Adam Zalepa

BLITZ BASIC

Programowanie od podstaw

Łódź 2017

ISBN 978-83-947369-6-5

Wydawca:
A2

Prawdziwy akt odkrycia
nie polega na odnajdywaniu
nowych lądów, lecz na patrzeniu
na stare w nowy sposób.

- Marcel Proust

SPIS TREŚCI

SPIS TREŚCI

SPIS TREŚCI

WPROWADZENIE

Niniejsza publikacja jest drugą książką z serii „Programowanie od podstaw". Poprzednia pozycja traktuje o języku „AMOS", który jest bardzo często krytykowany, głównie przez brak zgodności z systemem operacyjnym oraz małą prędkością działania. Jest to jednak dialekt niezmiernie łatwy do nauki i nie wymaga używania rozbudowanych konfiguracji Amigi. Z tego względu warto go poznać i przekonać się, jak szybkie i nieskomplikowane może być programowanie na swoim ulubionym komputerze.

Tym razem piszę o języku „Blitz Basic", do którego obsługi nie potrzeba nabywać wielu nowych umiejętności. Działa on jednak w formie systemowej – korzysta z elementów Workbencha i systemu operacyjnego. Pozwala na używanie w bezpośredni sposób typowego interfejsu użytkownika – ekranów, okien, różnych typów przycisków czy bibliotek. Oczywiście jest to tylko jeden z trybów działania. W książce skupiam się na nim, aby pokazać przede wszystkim jak pisać proste programy wyświetlające tekst czy grafikę we współdziałaniu z systemem operacyjnym.

Druga grupa omawianych poleceń wyłącza wielozadaniowość Amigi, aby uzyskać bezpośredni dostęp do funkcji układów specjalizowanych. Polecenia należące do tej grupy wprowadzam bardzo powoli, tłumacząc ich funkcje, samą realizację pozostawiając niejako na później. Nie chcę bowiem przytłaczać Czytelnika dużą ilością informacji, które z punktu widzenia początkującego programisty mogą wydawać się niespójne. Musisz być, drogi Czytelniku, świadomy różnych form

działania Twojego potencjalnego programu, ale wiadomości te będziemy przyswajać wolno, na razie bez konieczności analizy skomplikowanych listingów.

Siłą rzeczy wiele tematów wymaga kontynuacji, która nastąpi w kolejnej pozycji na temat programowania. Będzie ona przedstawiać inny, bardziej zaawansowany punkt widzenia, który wymaga pewnego obycia, nawet w świecie prostego języka jakim jest Basic, niezależnie od swojej odmiany.

Pamiętajmy, że programowanie to nie tylko – jak mówi definicja: proces tworzenia, projektowania i testowania programu komputerowego. W moim przekonaniu jest to przede wszystkim sztuka doskonalenia własnych umiejętności, które nie muszą odpowiadać trendom panującym w środowisku użytkowników określonego sprzętu. Powinny one natomiast „pasować" do naszych celów, ambitnych, wielkich lub małych, aby napisać program codziennego, prywatnego użytku. Tak więc kreatywność na pierwszym miejscu.

Wbrew panującym dziś przekonaniom chcę pokazać, że programować może każdy, niezależnie od wieku i ogólnych zainteresowań. Nabyte doświadczenia możemy wykorzystać w ramach różnych systemów operacyjnych, a napisane programy będą świadczyć o naszej wyobraźni i umiejętności abstrakcyjnego myślenia.

Książka opiera się na klasycznej wersji „Blitz Basica", czyli drugiej generacji działającej na komputerach Amiga produkcji Commodore. Wystarczy nam procesor 68000, 1-2 MB pamięci RAM i twardy dysk. Teoretycznie możemy się obejść również bez tego

ostatniego, ale nie polecam takiej „zabawy”, bowiem szybko może okazać się zniechęcająca. Tak więc nasze programy będą działać na zwykłej Amidze 500, 600 czy 1200. Trzeba jednak pamiętać, że nie jest to jedyna możliwość wykorzystania języka programowania, choć bardzo polecam zacząć od prostej konfiguracji komputera oraz nieskomplikowanych programów. Możemy wtedy przekonać się na własnej skórze, na ile prawdziwy jest slogan związany z Amigą i powtarzany często w ostatnim okresie:

Remember when computing was fun?

Dziś, po 30 latach od premiery Przyjaciółki nadal możemy cieszyć się jej możliwościami. Dla osoby zaczynającej programować będzie to cały wielki świat, który nie wyczerpie się zbyt szybko. Życzę więc moim Czytelnikom, aby pisali swoje programy, gry czy dema przede wszystkim dla przyjemności. Później zapraszam do pochwalenia się osiągnięciami w Internecie lub na jednym z wielu organizowanych imprez związanych z retro komputerami. Nie trzeba ciągle szukać nowego, czasem wystarczy spojrzeć na to, co mamy z innego punktu widzenia.

- Wrzesień 2017 r.

PODSTAWY

POBIERANIE Z INTERNETU

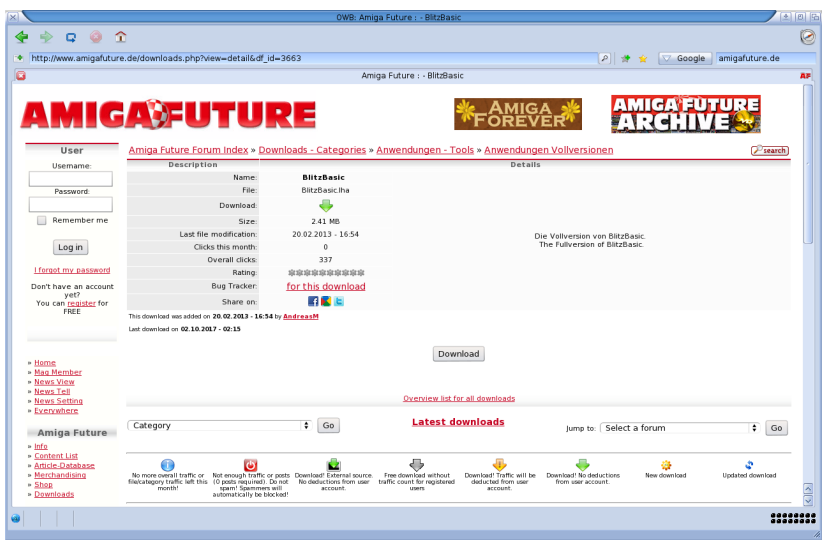
Blitz Basic można zdobyć na jednej z wielu stron internetowych, między innymi serwisu:

amigafuture.de

Przechodzimy do działu „Downloads", następnie „Anwendungen – Tools" oraz „Anwendungen Vollversionen" i wyszukujemy plik „BlitzBasic". Można także przejść bezpośrednio na stronę, gdzie znajdziemy właściwe archiwum. Oto pełny adres:

http://www.amigafuture.de/downloads.php?view=detail&df_id=3663

Tak wygląda strona po wczytaniu do przeglądarki OWB:



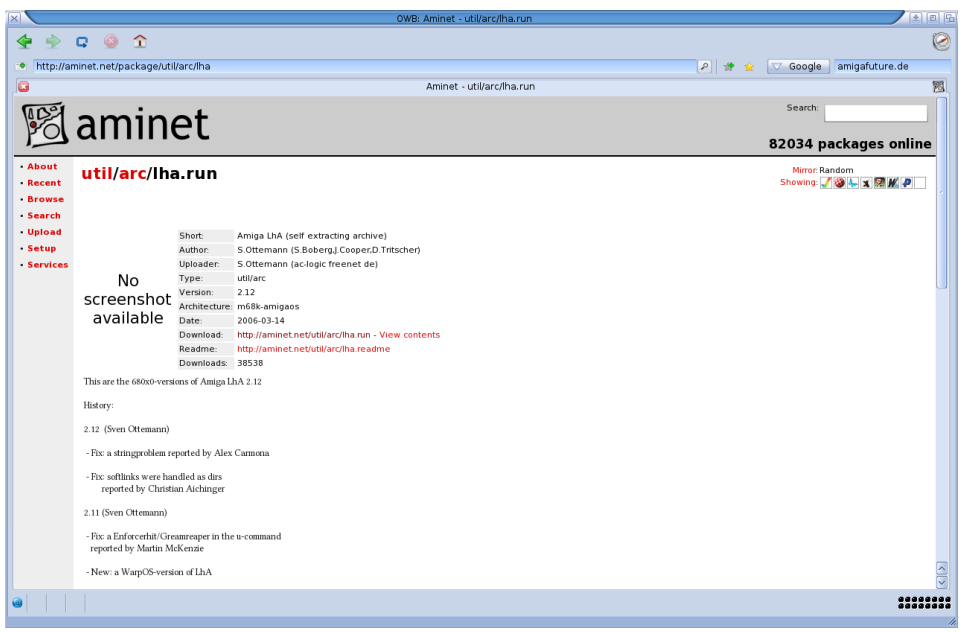
Teraz klikamy odnośnik „Download” i pobieramy plik na dysku, do dowolnego katalogu. Archiwum należy rozpakować za pomocą programu LHA, który można z kolei pobrać z serwisu:

aminet.net

Tym razem wyszukujemy frazę „lha.run”, aby odnaleźć plik, który można rozpakować automatycznie, bez instalowania kolejnych narzędzi. Powyższą nazwę trzeba wprowadzić w polu „Search” lub skorzystać z poniższego adresu:

http://aminet.net/package/util/arc/lha

Strona powinna wyglądać tak:



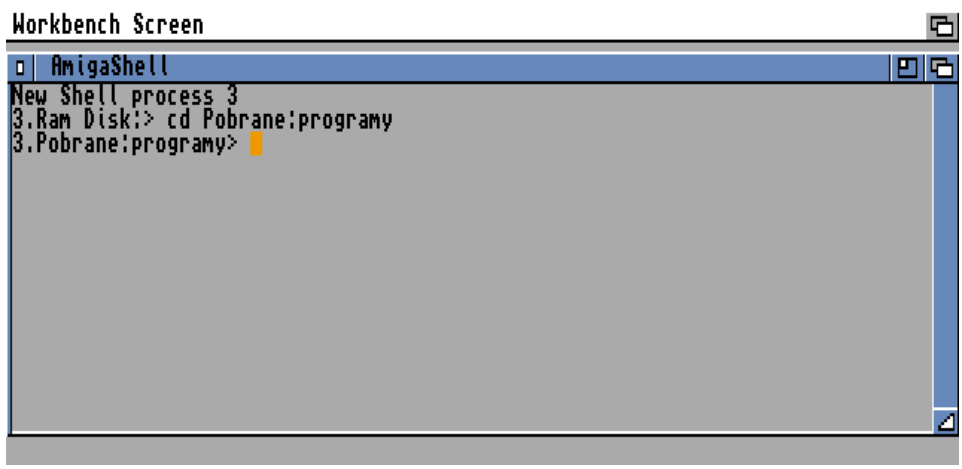
Kliknij nazwę widoczną obok napisu „Download:” zaznaczoną ramką na ilustracji. Plik zostanie pobrany na dysk. Na Workbenchu wywołaj opcję „Wykonaj polecenie...” (ang. „Execute Command”) z menu górnego o nazwie „Workbench” Wpisz z nim:

```
newshell
```

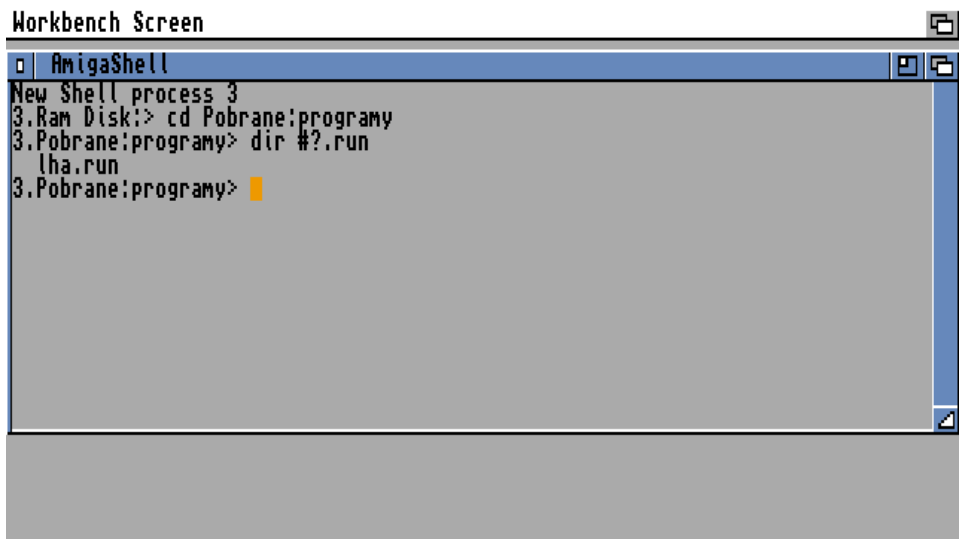
i naciśnij ENTER. Na ekranie pojawi się większe okno podpisane „AmigaShell”. Teraz musisz zmienić katalog bieżący na ten, do którego pobrałeś pliki. W tym celu wpisz polecenie CD, dalej naciśnij SPACJĘ i wpisz pełną ścieżkę dostępu. Na przykład, jeśli pliki zostały zapisane w katalogu „programy” na dysku „Pobrane”, cała linia powinna mieć następującą formę:

```
cd Pobrane:programy
```

Po naciśnięciu klawisze ENTER napis znajdujący się przed kursorem powinien zmienić się, tak jak na ilustracji:



Oczywiście jeśli Twoje katalogu będą miały inne nazwy, zobaczysz inny napis. Zawsze jednak musi on wskazywać na katalog, w których zapisałeś pobrane wcześniej pliki. Dla pewności możesz wyświetlić zawartość katalogu za pomocą polecenia DIR. Wystarczy wprowadzić jego nazwę oraz symbol „#?.run". Dzięki temu w oknie zobaczysz tylko pliki kończące się rozszerzeniem „.run". Po naciśnięciu ENTER informacja powinna wyglądać tak:



Gdy jesteśmy pewni, że mamy dostęp do plików, możemy przystąpić do dalszej pracy. W oknie „Shell" wpisz:

```
lha.run
```

i naciśnij ENTER. Po chwili pojawią się nowe komunikaty, a większość linii powinna rozpoczynać się nazwą „LHA". Ostatnia linia powinna zawierać napis:

```
Operation successfull
```

Jeśli jest inaczej oznacza to, że plik nie został poprawnie zapisany i utracił tak zwane atrybuty AmigaDOS. Dzięki nim plik posiadają określone cechy, między innymi za ich pomocą system stwierdza, czy dany plik może być rozpakowany przez wpisanie jego nazwy – tak jak w tym przypadku. Aby naprawić problem musisz wprowadzić jeszcze jedną linię, tym razem z poleceniem PROTECT. Wpisz:

```
protect lha.run +rwed
```

Teraz plik uzyska odpowiednie atrybuty i będziesz mógł z niego skorzystać. Jeśli wszystko przebiegło prawidłowo należy jeszcze skopiować jeden z plików do systemowego katalogu „C". W tym celu wprowadź kolejną linię:

```
copy lha_68k C:lha
```

po naciśnięciu klawisza ENTER kursor powinien przejść do następnej linii bez wyświetlania dodatkowych informacji. Oznacza to, że wszystko jest w porządku. W ten sposób zainstalowałeś program LHA na Twoim dysku systemowym. Za jego pomocą rozpakujesz archiwum z językiem Blitz Basic.

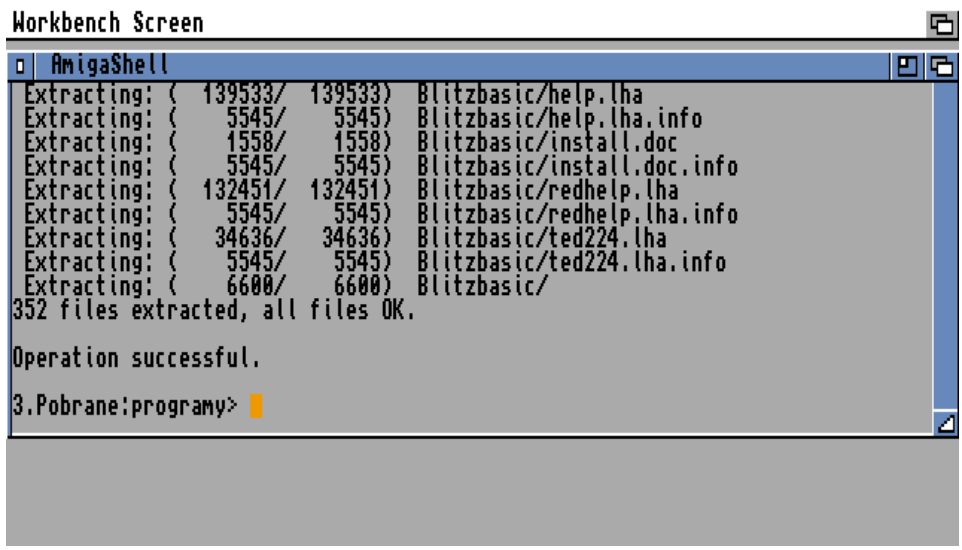
Aby tak się stało należy wpisać linię, która spowoduje rozpakowanie pliku „BlitzBasic.lha" na dysk twardy lub do pamięci komputera, czyli „Ram Dysku". Najprościej skorzystać z tego samego miejsca, w którym są zapisane pliki. Dlatego w następnej kolejności wpisz linię:

```
lha x BlitzBasic.lha
```

i potwierdź – tak jak wcześniej – klawiszem ENTER. W oknie będzie przewijać się dużo więcej plików, ale ostatnią linią znowu będzie napis:

```
Operation successful
```

Sytuacja powinna być podobna do poniższej:



Wszystkie podstawowe pliki są rozpakowane tak, aby zainstalować Blitz Basic na twardym dysku. Jak to zrobić dowiesz się z lektury następnego punktu.

INSTALACJA NA TWARDYM DYSKU

Proces instalacji języka Blitz Basic jest dość skomplikowany, bowiem nie dołączono do niego programu, który mógłby to zrobić za nas automatycznie. Wszystkie operacje możesz wykonać za pomocą jednego z menadżerów plikowych jak „Directory Opus" czy „File Master", ale my zrobimy to bez instalacji dodatkowego oprogramowania. Będzie to procedura nieco mniej wygodna, ale za to możesz ją zastosować na każdym komputerze, nawet jeśli dopiero zainstalowałeś system operacyjny bez żadnych rozszerzeń.

Na początek musimy utworzyć nowy katalog, w którym zapiszemy wszystkie pliki niezbędne do działania. Może być do miejsce na dowolnym dysku, jak również nazwa katalogu nie ma znaczenia. My użyjemy krótkiej nazwy „Blitz", aby ułatwić sobie kolejne operacje. Dane będziemy zapisywać na dysku systemowym, w dodatkowym katalogu „Tools", który należy do standardowych pozycji po zainstalowaniu systemu operacyjnego.

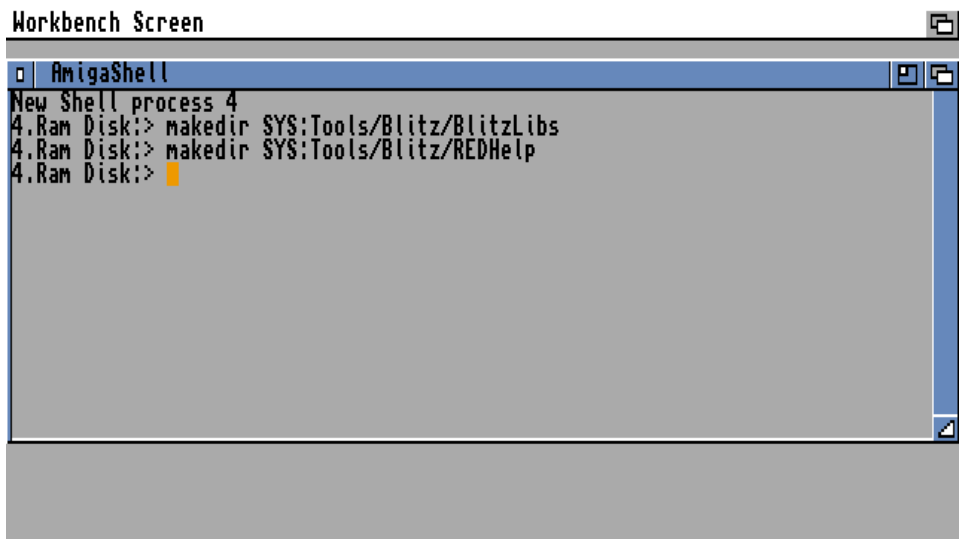
Tworzenie katalogu wykonamy za pomocą polecenie MAKEDIR. Wpisz poniższą linię:

```
makedir SYS:Tools/Blitz
```

i naciśnij ENTER. Kolejnym krokiem będzie utworzenie dwóch kolejnych katalogów wewnątrz „Blitz". Aby to osiągnąć wprowadź dwie następne linie:

```
makedir SYS:Tools/Blitz/BlitzLibs
makedir SYS:Tools/Blitz/REDHelp
```

Zwróć uwagę, że po wykonaniu każdej linii w oknie „Shell” nie powinien pojawić się żaden dodatkowy komunikat. W tym wypadku oznaczałoby to błąd, a tak wiemy, że wszystko przebiegło poprawnie. Tak jak na ilustracji:



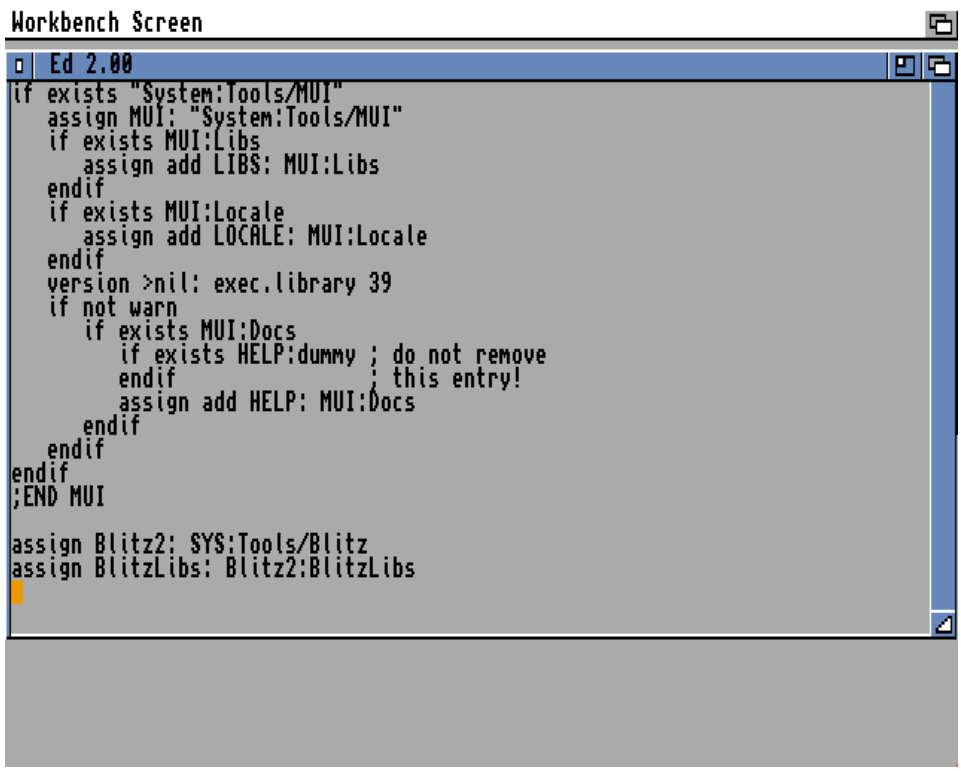
Teraz do utworzonych katalogów należy przyporządkować dwa urządzenia logiczne, o nazwach „Blitz2:” oraz „BlitzLibs:”. W tym celu musisz dokonać edycji pliku „user-startup”, który znajduje się w systemowym katalogu „S”. Dlatego wpisz następną linię:

```
ed S:user-startup
```

i naciśnij ENTER. Na ekranie pojawi się duże okno z wczytaną treścią pliku. Przejdź kursorem na koniec i dopisz dwie linie:

```
assign Blitz2: SYS:Tools/Blitz
assign BlitzLibs: Blitz2:Blitzlibs
```

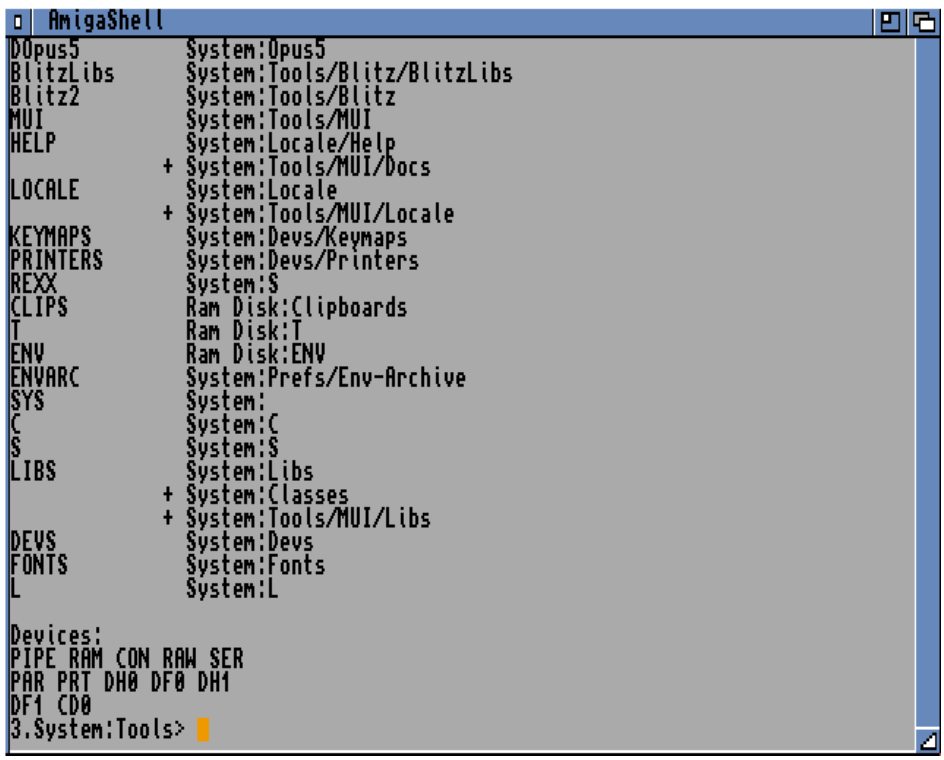
Musisz to zrobić bardzo dokładnie, aby nie pomylić żadnego znaku, inaczej nasze „przypisania" nie zadziałają. Wybierz opcję „Save" z menu górnego „Project", aby zapisać plik. Okno programu „Ed" powinno wyglądać tak:



Zamknij okno programu, zresetuj Amigę i wczytaj ponownie Workbencha. Jeżeli na ekranie nie pojawią się komunikaty o błędach oznacza to, że wszystko jest w porządku. Dla pewności możesz sprawdzić, czy nowe urządzenia logiczne są aktywne. Znowu otwórz okno „Shell", wpisz w nim:

```
assign
```

i naciśnij ENTER. Spowoduje to wyświetlenie listy dostępnych urządzeń. Rozszerz okno tak, aby widzieć całą listę i zwróć uwagę na pozycje rozpoczynające się słowem „Blitz". Tak to wygląda u nas:



Jeśli obie pozycje są u Ciebie widoczne, możesz kontynuować instalację. Zmień katalog bieżący na ten sam, co wcześniej – u nas jest to ścieżka „Pobrane:programy'. Wcześniej rozpakowaliśmy archiwum LHA, tak więc możemy od razu skorzystać z zawartych w nich plików.

Za pomocą poniższej linii zmień katalog na „Blitzbasic" zapisany wewnątrz:

```
cd Blitzbasic
```

Jeśli teraz wyświetlisz zawartość zawartość za pomocą polecenia DIR zobaczysz kolejne pliki z rozszerzeniem „.lha". Ich zawartość należy rozpakować do dwóch konkretnych katalogów, o których zaraz powiemy. Musisz to zrobić tak samo jak wcześniej, to znaczy wpisując linię według schematu:

```
lha x <PLIK> <KATALOG>
```

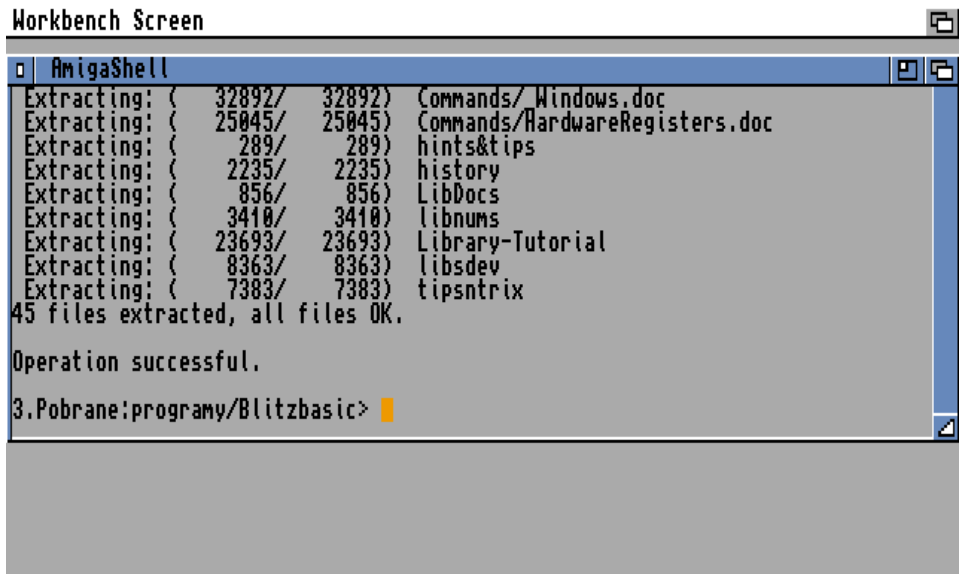
Oczywiście zamiast „<PLIK>" i „<KATALOG>" trzeba podać nazwy plików LHA i ścieżki dostępu. Oto ich lista:

<PLIK>	<KATALOG>
acidlibs.lha	Blitz2:
blitz210.lha	Blitz2:
blitzlibs.lha	Blitz2:Blitzlibs/
defdebug.lha	Blitz2:
deflibs.lha	Blitz2:
help.lha	Blitz2:
redhelp.lha	Blitz2:REDHelp/
ted224.lha	Blitz2:

Dla każdego pliku musisz wprowadzić osobną linię. Innymi słowy, aby rozpakować pierwszy plik wpisz:

```
lha x acidlibs.lha Blitz2:
```

Za każdym razem w oknie pojawią się nowe informacje, podobnie jak wcześniej. Przykładowo tak:



W rezultacie, w urządzeniu logicznym „Blitz2:” powinieneś uzyskać odpowiednią strukturę katalogów, a w nich zapisane pliki. Aby to sprawdzić zmień katalog bieżący:

```
cd Blitz2:
```

i wyświetl jego zawartość za pomocą polecenia DIR. Powinieneś zobaczyć następujące pozycje:

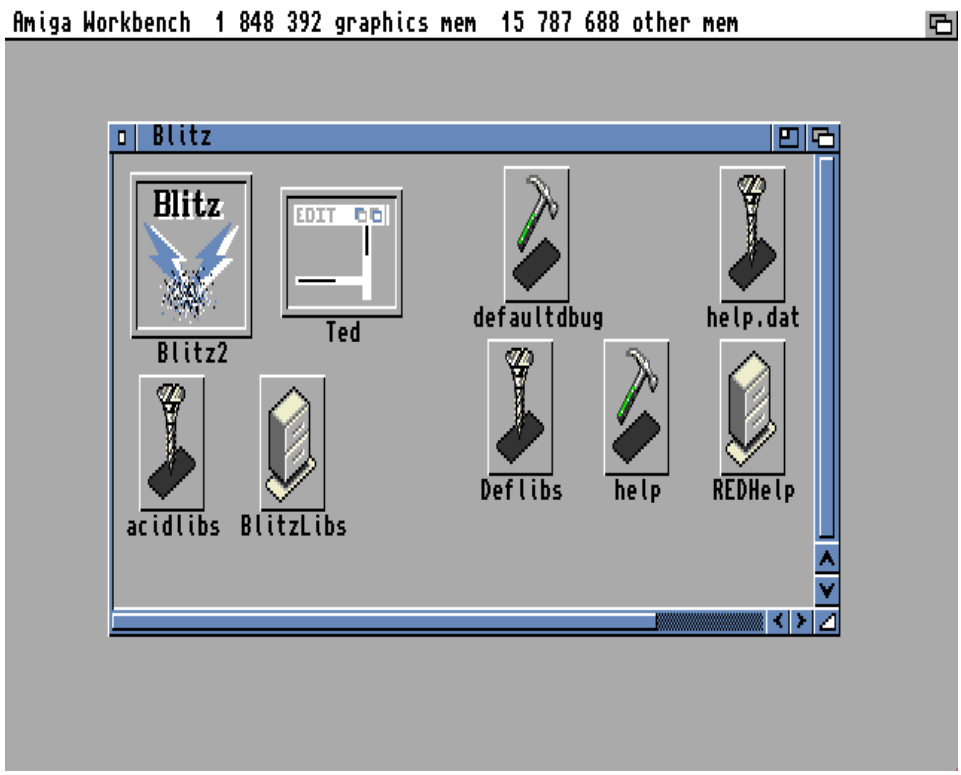
Na koniec należy skopiować bibliotekę systemową „ted.library” do urządzenia „LIBS:”. Wpisz kolejną linię:

```
copy ted.library LIBS:
```

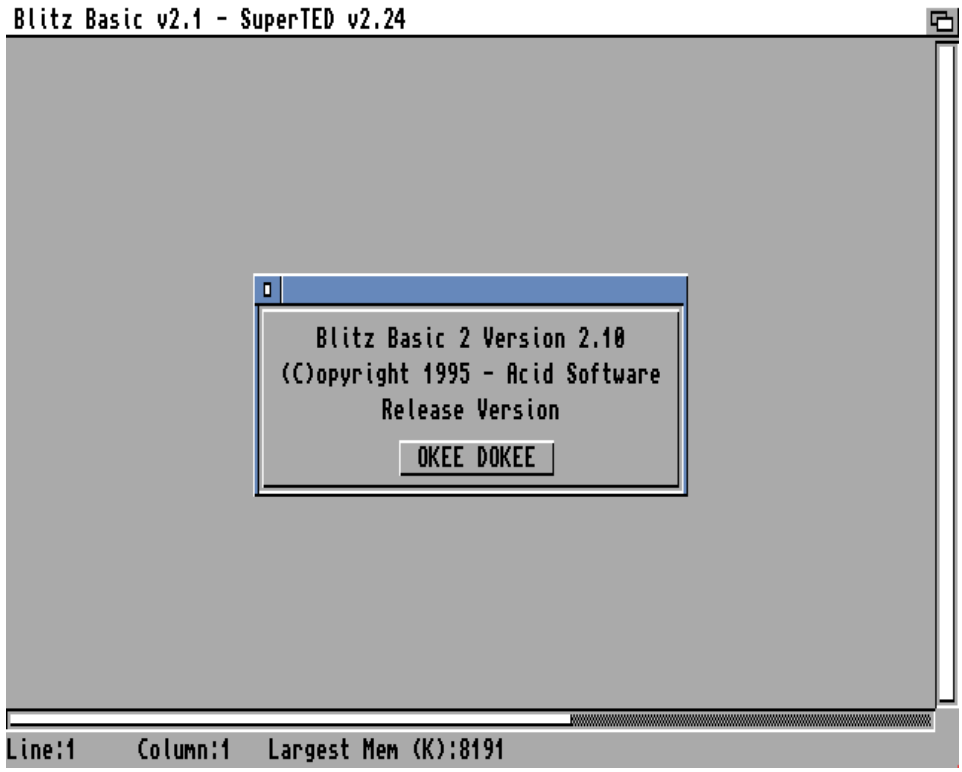
i naciśnij ENTER. Kursor powinien przeskoczyć do następnej linii bez dodatkowych komunikatów.

Zresetuj Amigę i wczytaj ponownie Workbencha. Odczytaj zawartość dysku systemowego, potem katalogu „Tools” oraz kolejnego katalogu „Blitz”, który utworzyliśmy. Nie posiada on swojej ikony, dlatego nie zobaczysz go od razu. W oknie katalogu „Tools” wywołaj menu górne „Okna”, wybierz pozycję „Wyświetl...”, a następnie opcję „Wszystkie pliki”.

Przewiń zawartość okna i odszukaj ikonę podpisaną „Blitz”. Wykonaj na niej zwykły dwuklik, aby odczytać zawartość. Zobaczysz nowe okno, a w środku kilka ikon. Powinno to wyglądać jak na poniższej ilustracji:



Najedź wskaźnikiem na ikonę „Blitz2” i naciśnij dwukrotnie lewy przycisk myszki. Uruchomisz w ten sposób edytor, w którym będziemy piać programy w języku Blitz Basic. Znowu spójrz na ilustrację:



Wybierz przycisk „OKEE DOKEE”. Edytor jest teraz gotowy do pracy. Zanim jednak zaczniesz pisać program powinieneś zapoznać się z podstawowymi funkcjami programu widocznego na ekranie. Przeczytasz o nich w rozdziale zatytułowanym „Edytor”.

URUCHOMIENIE Z DYSKIETKI

Praca wyłącznie ze stacją dyskietek nie należy do wygodnych. Jeśli Twoja Amiga ma możliwość podłączenia twardego dysku, nie wahaj się kupić choćby najmniejszy napęd. Szybkość odczytywania i zapisywania plików oraz wygoda użytkowania plików o dużych objętościach jest nie do przecenienia.

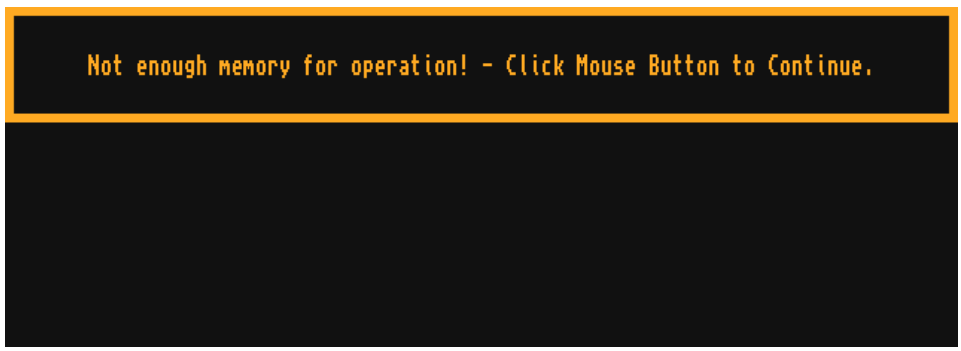
Jeśli jednak z różnych powodów musisz ograniczyć się do dyskietek, zacznij od wykonania kopii dyskietki systemowej, z której uruchamiasz Workbencha. Usuń wszystkie niepotrzebne pliki, czyli na przykład zawartość poniższych katalogów:

```
System
Rexx
Utilities
```

Na dyskietce pozostanie miejsce na zapisanie tylko podstawowych plików Blitz Basica. Jeśli Twoja Amiga posiada przynajmniej ok. 2 MB wolnej pamięci możesz skopiować niezbędne pliki do „Ram Dysku" i używać takiego zestawu alternatywnie w stosunku do wczytywania całości z twardego dysku.

Wszystkie czynności musisz wykonać analogicznie do opisu z poprzedniego punktu. Pliki możesz też zapisać na płycie CD, wtedy do pliku „startup-sequence" dopisz linie z poleceniem ASSIGN, a wcześniej aktywuj sterownik napędu optycznego.

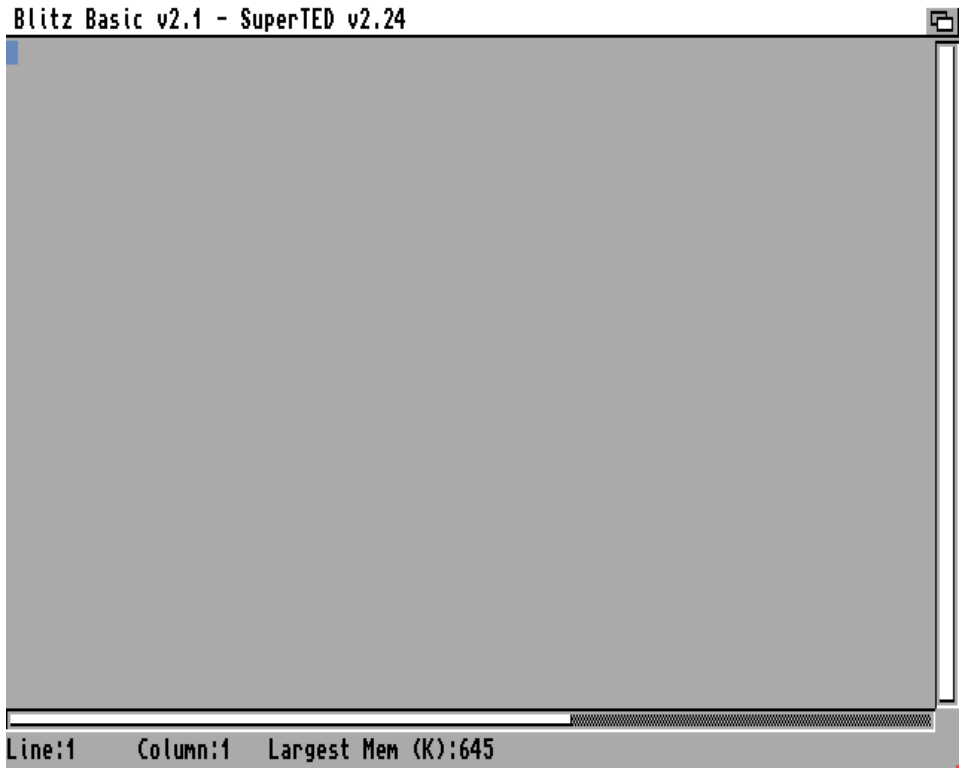
W praktyce jednak taka praca nie będzie miała większego sensu, bowiem w sytuacjach problematycznych większość procedury będziesz musiał powtarzać. Ponadto możesz zobaczyć komunikat wskazujący na brak odpowiedniej ilości wolnej pamięci do pracy:



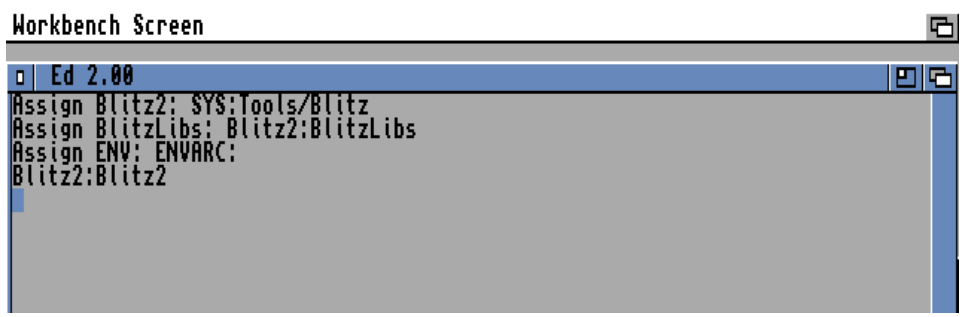
Dlatego, tak jak w przypadku innych komputerów, jeżeli chcesz zająć się bardziej poważnym wykorzystaniem Amigi musisz wyposażyć ją w kontroler dysku oraz odpowiedni napęd. W takiej sytuacji nawet pamięć o wielkości 1-2 MB nie ograniczy możliwości wczytania edytora. Będziesz musiał jednak uruchomić komputer bez wykonywania sekwencji startowej oraz ręcznie wykonać „przypisanie" do urządzenia logicznego „ENV". Najprościej możesz to zrobić za pomocą linii:

```
assign ENV: ENVARC:
```

Dodatkowo musisz utworzyć urządzenia logiczne o nazwach „Blitz2" i „BlitzLibs", tak samo jak wcześniej, ale poza plikiem „user-startup". W ten sposób zaoszczędzisz pamięć i spowodujesz, że dostępny pozostanie obszar ok. 600 kilobajtów. Tak jak na naszej ilustracji:



Dla ułatwienia zobacz zmodyfikowany plik „startup-sequence”, który wczytuje tylko te składniki, które są absolutnie niezbędne do działania:



Twój plik powinien wyglądać podobnie, bowiem umieszczone w nim linie tworzą jedynie wspomniane już urządzenia logiczne, a następnie uruchamiają edytor Blitz Basica. Jak go obsługiwać powiemy w kolejnym rozdziale.

EDYTOR

WPROWADZANIE PROGRAMU

Edytor, który widzisz na ekranie ma nazwę „SuperTED" i stanowi rozwiniętą wersję programu „Ted". Jest to główny element, w którym piszemy program oraz wykonujemy inne czynności związane z testowaniem i modyfikacją poleceń. Pamiętaj, że nie musisz od razu korzystać ze wszystkich możliwości, najlepszą drogą jest poznawanie tych funkcji, których w danej chwili potrzebujesz.

„Ted" jest zwykłym edytorem tekstu, tak więc możesz go użyć do zapisywania dowolnych plików typu ASCII. Obszar roboczy nie różni się zasadniczo od innych podobnych programów. Widoczne jest standardowe okno, suwaki przewijania i inne typowe elementy. Wszystkie działają analogicznie do okien innych programów uruchamianych na ekranie Workbencha.

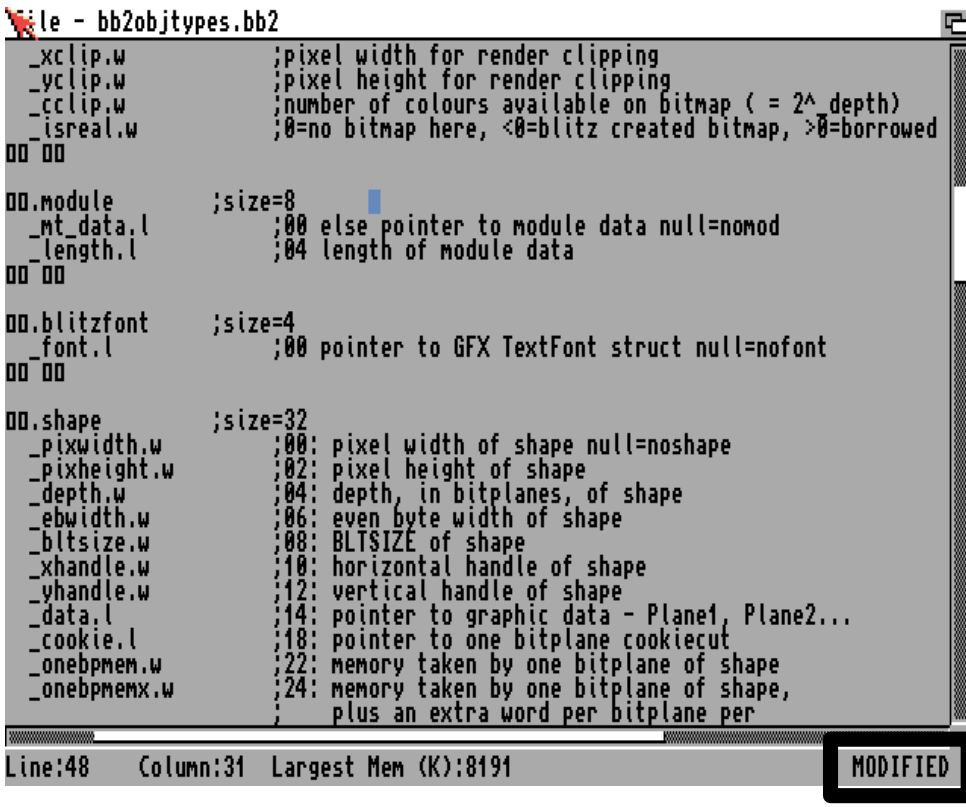
Poszczególne opcje związane z językiem Blitz Basic znajdują się w menu górnym programu. Zanim dowiesz się jak z nich korzystać dodajmy, że warto poznać skróty klawiaturowe, za pomocą których możliwe jest łatwiejsze lub szybsze wywoływanie funkcji.

Zwróć uwagę, że program przypomina systemowy edytor „Ed", ale jest bardziej rozbudowany. W dolnej części ekranu znajdują się informacje o pozycji kursora - „Line" oraz „Column". Oznaczają linię i kolumnę, w której aktualnie znajduje się kursor.

Obok widać liczbę wskazującą na maksymalną ilość pamięci z jakiego możesz korzystać. Jest to wartość wyrażona w kilobajtach, a więc zapis „8191" oznacza ok. 8 megabajtów. Na ilustracji możesz zauważyć

dużo większą liczbę, bowiem wszystko zależy od konkretnej ilości pamięci komputera. Nie możesz wykorzystać 100% pamięci, bowiem program musi mieścić się w ciągłym obszarze.

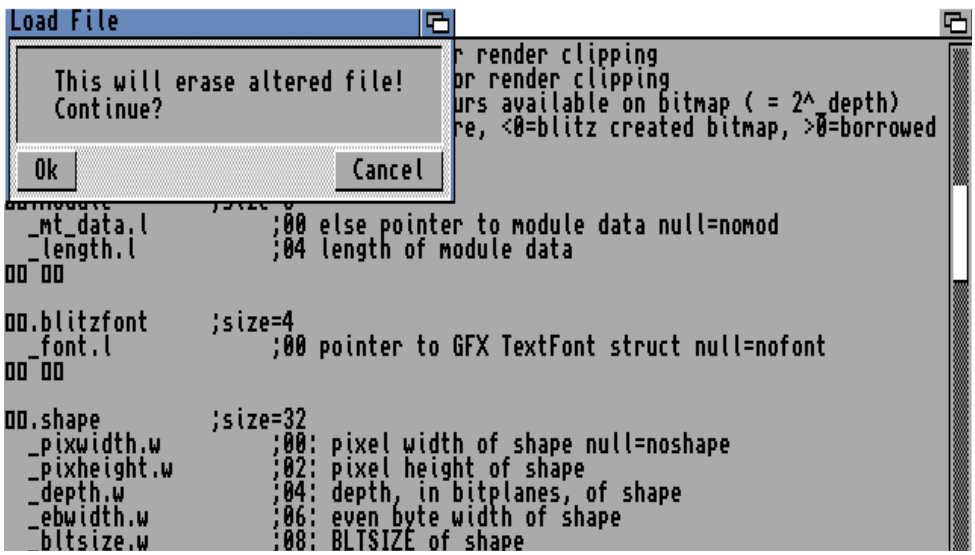
Zwróć uwagę, że w prawym dolnym rogu co jakiś czas pojawia się napis „MODIFIED”:



Jeśli go widać, plik został zmieniony w stosunku do wersji zapisanej na dysku. Jeśli zapiszesz go w dowolnym miejscu, napis zniknie. Dzięki temu zawsze wiesz, czy zawartość programu jest taka

sama jak na dysku, czy też została już zmodyfikowana. Wbrew pozorom jest to bardzo często istotny komunikat, dlatego warto o nim pamiętać.

Gdy spróbujesz wczytać inny plik lub zamknąć edytor, a nie zapisałeś treści programu, w górnym lewym rogu zobaczysz małe okno, jak poniżej:

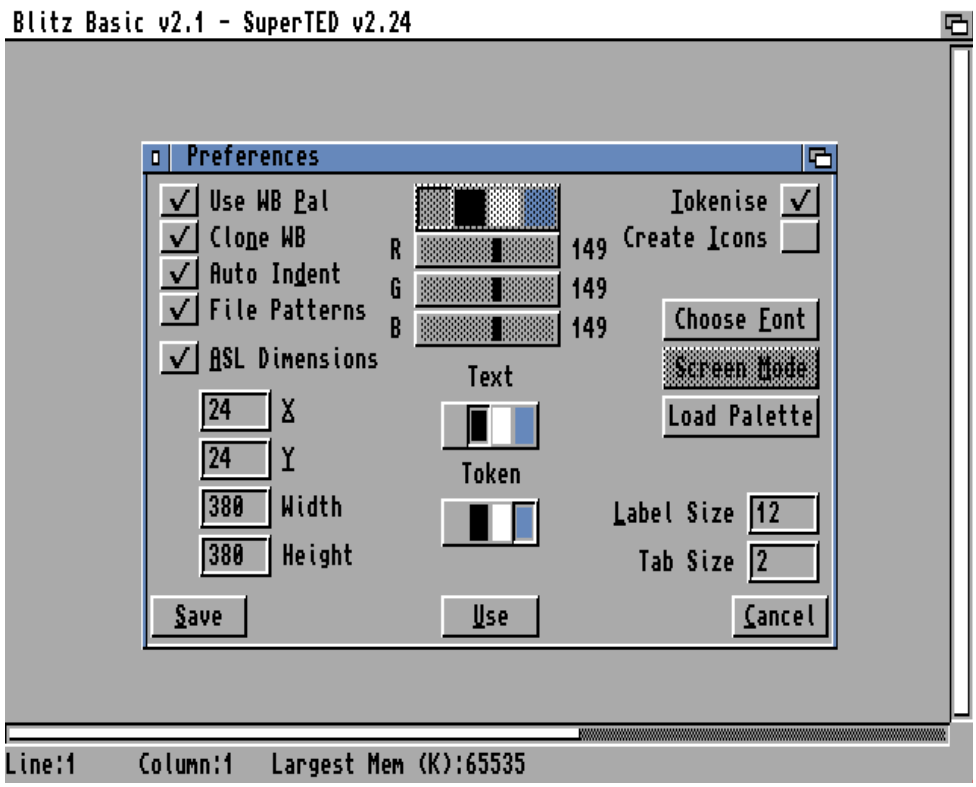


Tym razem edytor informuje Cię, że stracisz swój program. Jeżeli nie jest on ważny użyj przycisku „Ok” i wykonaj dalsze operacje, na przykład załaduj inny plik. Jeśli jednak chcesz zachować zawartość edytora, wybierz pole „Cancel” i zapisz plik na dysk za pomocą opcji „Save As...”. Znajdziesz ją w menu górnym, o którym teraz powiemy szerzej.

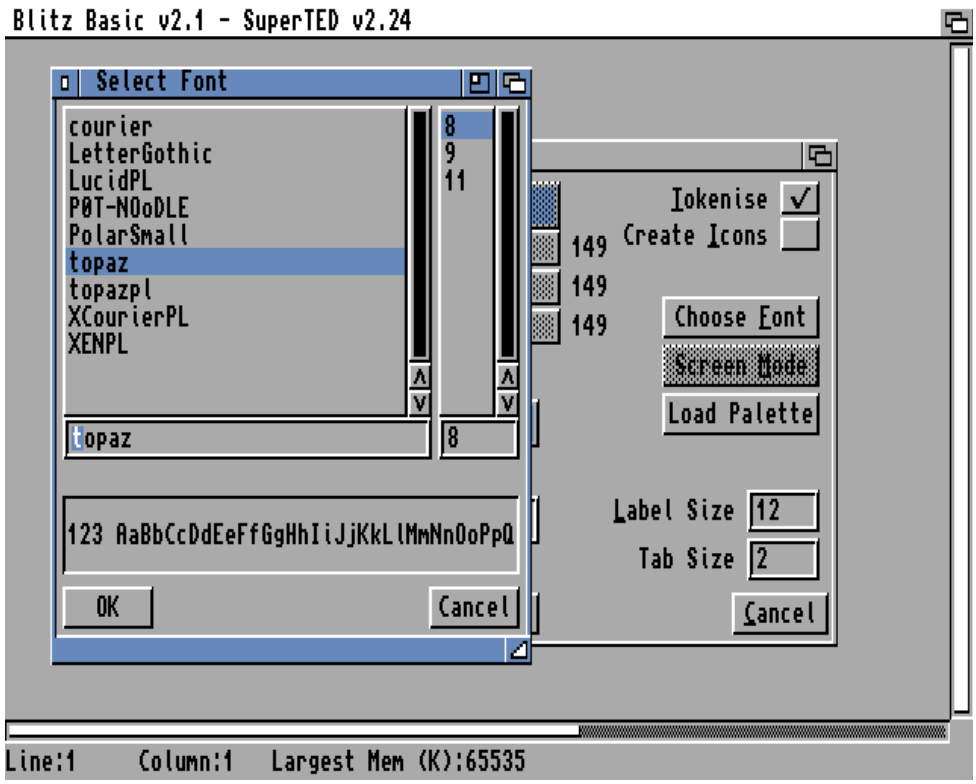
MENU GÓRNE

Najbardziej charakterystyczną sprawą w edytorze „Ted” jest menu górne, które zawiera szereg funkcji związanych bezpośrednio z działaniem Twojego programu. Pierwsze z nich o nazwie „Project” zawiera pozycje pozwalające odczytywać i zapisywać pliki na dysku. Są to typowe opcje jakie można znaleźć w większości programów dla Amigi.

Szczególną funkcją jest „Prefs...”, dzięki której możesz zmieniać ustawienia programu. Okno konfiguracji wypełnia prawie cały ekran:



Nie musisz zwracać uwagę na wszystkie opcje, ale warto poznać kilka najważniejszych. Za pomocą przycisku „Choose Font” możliwe jest wybranie czcionki, która będzie używana dla całego edytora. Na ekranie zobaczysz standardowe okno wyboru czcionek z napisem „Select Font”, jak na poniższej ilustracji:



Wskaż interesujący Cię krój i potwierdź wybór za pomocą pole „OK”. Ekran edytora zostanie zaktualizowany i odtąd wszystkie jego elementy będą korzystać z nowej czcionki. Zostanie to uwzględnione zarówno dla linii informacyjnej w dolnej części, jak i treści programu.

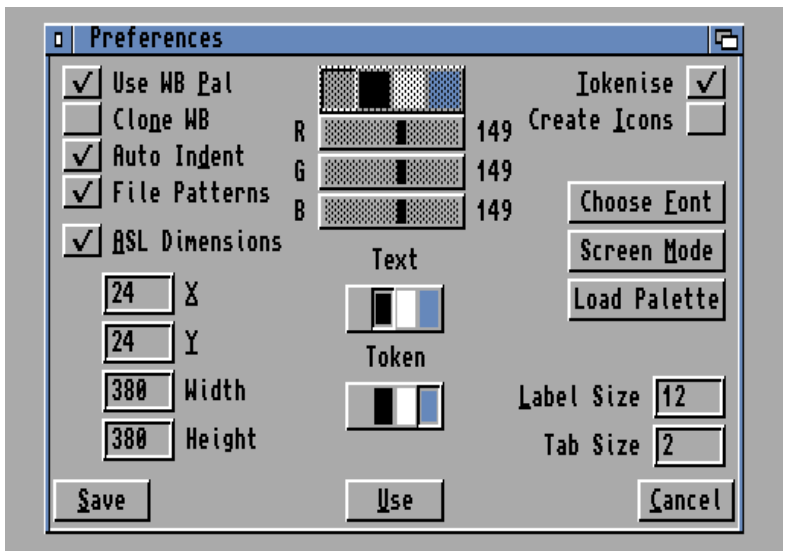
Zwróć jednak uwagę, że okno konfiguracji nadal będzie korzystać ze standardowych czcionek „Topaz". Dzięki temu nie musisz przejmować się, czy wybierasz odpowiedni krój, bowiem nawet jeśli użyjesz bardzo egzotycznych znaków, zawartość okna „Preferences" będzie zawsze czytelna.

Kolejną ważną opcją jest pole oznaczone jako „Create Icons". Aby je aktywować po prostu najedź wskaźnikiem na kwadrat i naciśnij lewy klawisz myszki. W polu znajdzie się charakterystyczny znak zakreślenia. Teraz przy zapisywaniu plików w edytorze, będą automatyczne dodawane odpowiednie ikony. Dzięki temu wystarczy, że wykonasz na nich dwuklik w oknie Workbencha, aby wczytać zarówno program, jak i treść programu zapisaną w pliku.

W centralnej części okna widocznych jest kilka większych kolorowych pól, a nad nimi napis „Text". Dzięki nim możesz zmieniać barwę jaką będzie wyświetlany Twój program. Oczywiście dotyczy to wyłącznie treści listingu w edytorze. Wystarczy kliknąć jeden z kwadratów, a następnie użyć przycisku „Use" lub „Save" na dole. Dla porządku dodajmy, że pierwszy zapamiętuje ustawienia do czasu ponownego uruchomienia programu, a drugi zapisuje zmiany na stałe.

Inny kolor tekstu może się przydać, gdy piszesz długi program. Wzrok szybko przyzwyczają się do nowych warunków pracy, ale po jakimś czasie możesz odczuć zmęczenie oczu. Jednak cztery podstawowe kolory mogą Ci nie wystarczyć, dlatego autorzy „Teda" przewidzieli możliwość uruchomienia go na oddzielnym ekranie.

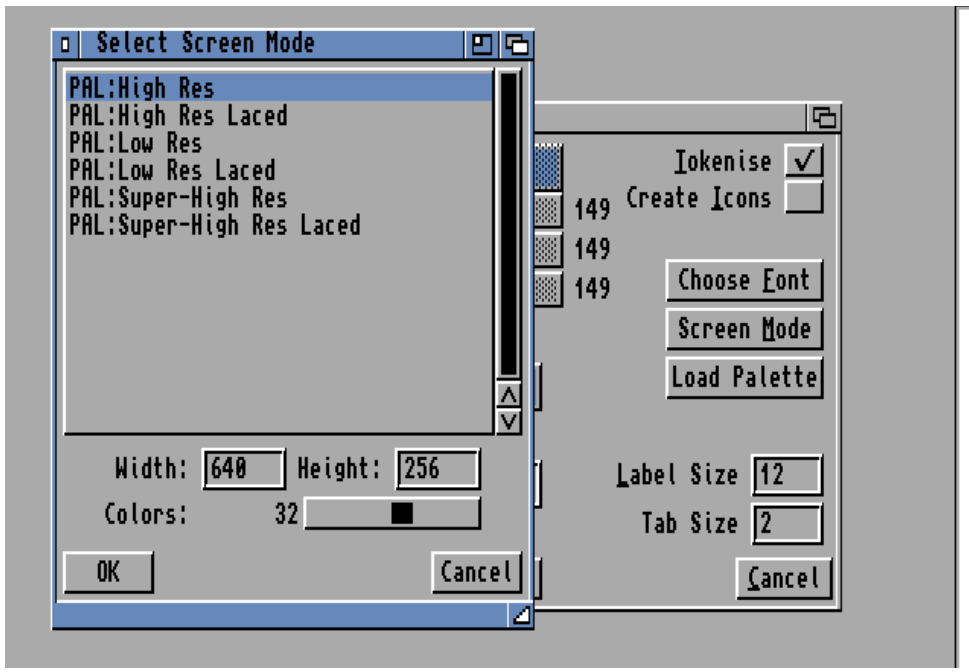
W tym celu musisz wykonać dwie czynności. Po pierwsze wyłącz opcję oznaczoną jako „Clone WB” po lewej stronie okna. Gdy pole będzie nieaktywne, aktywny stanie się przycisk „Screen Mode” po prawej stronie. Zobacz jak to powinno wyglądać:



Teraz musisz wybrać tryb wyświetlania, który będzie używany na oddzielnym ekranie przypisanym do edytora. Wskaż wspomniany przycisk, aby wyświetlić listę dostępnych trybów. Zwróć uwagę, że poza ich nazwami, na dole widać suwak „Colors”, za pomocą którego zmienisz ilość dostępnych kolorów. Przesuń go w prawo, aby uzyskać przynajmniej 16 barw.

Pamiętaj, że konkretne możliwości zależą od układów zamontowanych w Twojej Amidze oraz użytego sterownika ekranu. Standardowo jest to „PAL”, ale równie dobrze możesz użyć trybu „Multiscan” czy „Euro72”. Trzeba je tylko włączyć na Workbenchu.

My mamy Amigę z układami AGA, więc ustawimy 32 kolory. Okno powinno więc wyglądać następująco:



Wybierz przycisk „OK”, a następnie „Use” lub „Save” w głównym oknie konfiguracji. Ekran zostanie przełączony i teraz „Ted” znajdzie się na ekranie o innych parametrach. Wybierz ponownie opcję „Prefs...” z menu górnego, aby się o tym przekonać.

Nie zmieniliśmy rozdzielczości, więc tekst będzie wyglądał tak samo, ale w środkowej części okna zobaczysz większą ilość kolorowych kwadratów. Ich funkcja pozostanie identyczna, natomiast wyboru należy dokonać bardziej precyzyjnie.

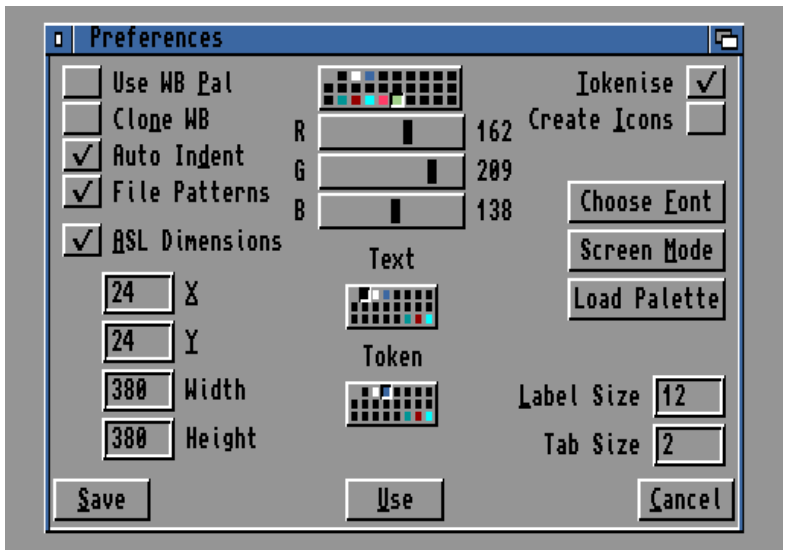
Nie będzie tutaj widać wszystkich możliwych barw, bowiem edytor korzysta maksymalnie z 24 kolorów. Spójrz na różnicę:

Taki wybór jest całkowicie wystarczający. Teraz możesz swobodnie decydować o kolorystyce Twojego listingu. Pamiętaj jednak, że im więcej kolorów na ekranie, tym więcej pamięci zajmuje edytor. Dlatego z tej opcji korzystaj przede wszystkim wtedy, gdy masz Amigę wyposażoną w dodatkową pamięć Fast, która w momencie uruchomienia programu jest w dużej części wolna.

Jeśli chcesz uzyskać barwę, której nie widać możesz zmienić paletę kolorów. Domyślnie program przyjmuje paletę zbieżną z ekranem Workbencha. Wystarczy jednak „wyłączyć” przycisk o nazwie „Use WB Pal”, aby sytuacja uległa zmianie. W tym wypadku bezpośrednio po aktywacji pola zmieni się kolorystyka ekranu – bez potrzeby

potwierdzania operacji. Jeśli chcesz przywrócić poprzedni stan, po prostu jeszcze raz kliknij pole „Use WB Pal”.

Wraz z tą zmianą aktywne będą suwaki umieszczone w centralnej części okna, opisane jako „R”, „G” oraz „B”:

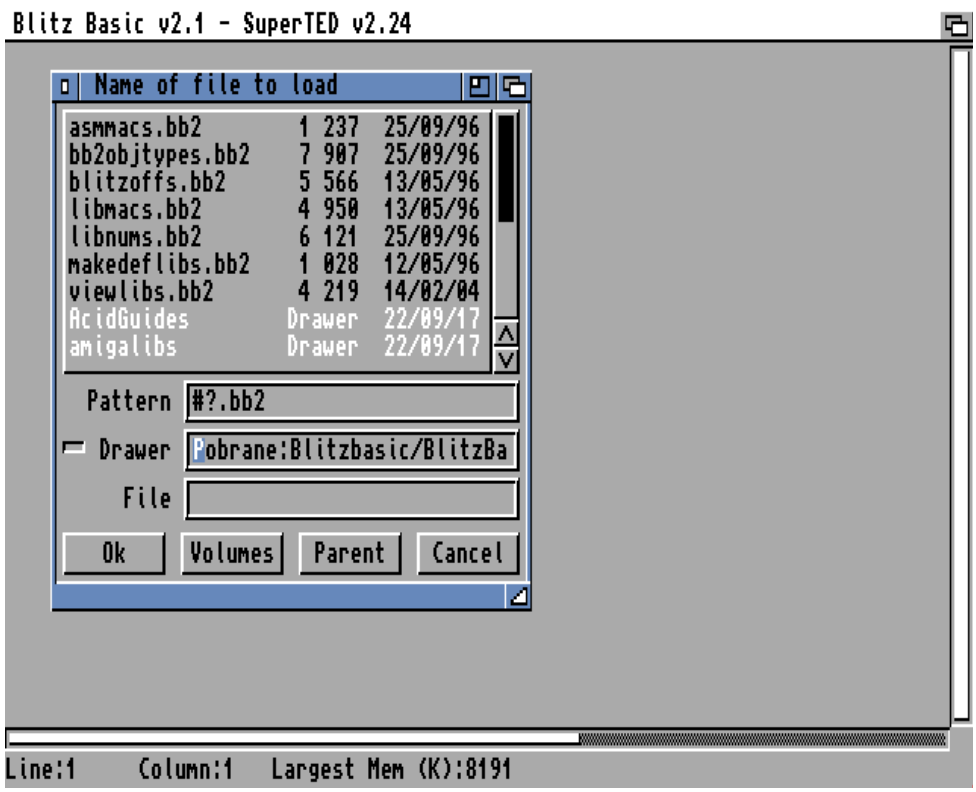


Teraz możesz zmodyfikować wszystkie barwy. Wystarczy tylko kliknąć na jeden z kwadratów na samej górze – nad suwakami – i zmienić stan jednej ze składowej: czerwonej (R), zielonej (G) lub niebieskiej (B). Wszystko razem powoduje, że ekran edytora możesz dostosować nie tylko do konfiguracji swojej Amigi, ale również monitora. Najlepiej ustawić kolory, które nie będą męczyły wzroku, ale nie przeszkodzą także w analizie programu. Listing musi zawsze pozostać czytelny.

Ważną funkcją jest także pole oznaczone jako „File Patterns”. Jeśli jest aktywne, w oknach wyboru plików – po wywołaniu opcji

odczytywania lub zapisywania – pojawi się dodatkowe pole „Pattern”. Dzięki niemu możesz stosować tak zwane filtry AmigaDOS, czyli znaki zastępujące nazwy, na przykład „#?”. Jest to przydatne, gdy w jednym katalogu na dysku masz zapisanych dużą ilość plików i chcesz wyświetlić tylko określoną grupę. Nie każde okno wyboru daje taką możliwość, w przypadku edytora „Ted” możesz o tym decydować.

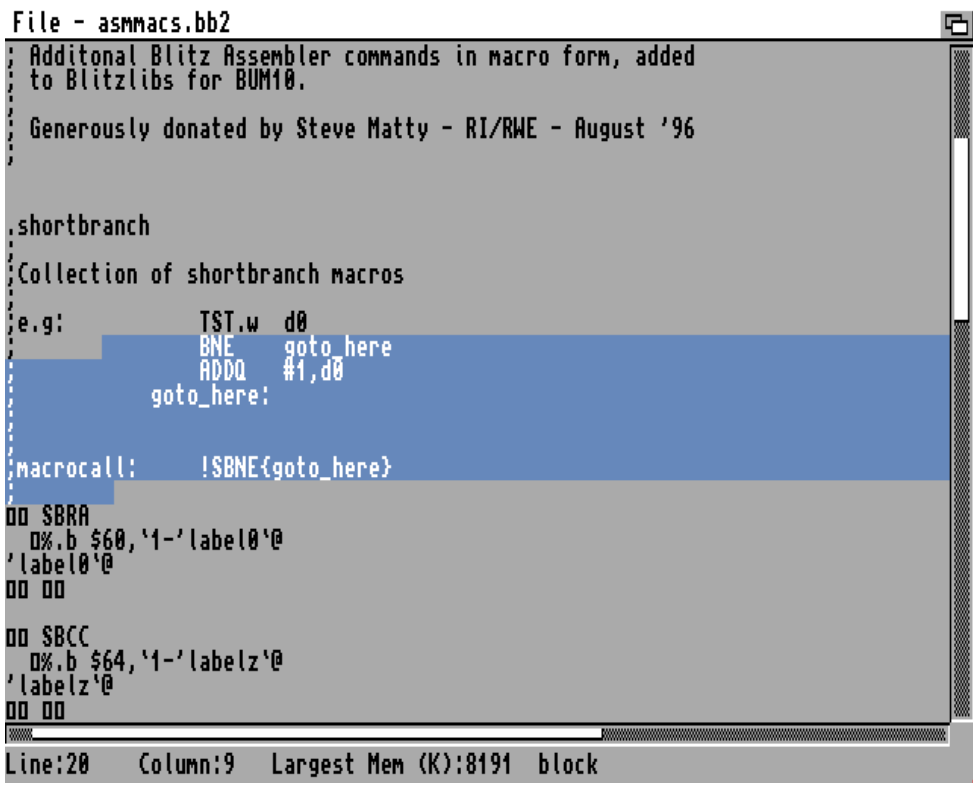
Jeśli więc w katalogu chcesz zobaczyć tylko pliki z rozszerzeniem „.bb2” charakterystycznym dla Blitz Basica, zastosuj wpis jak na poniższej ilustracji:



Jeżeli pole „File Pattens” zostanie wyłączone, okno wyboru zostanie pozbawione dodatkowego pola, a na liście widoczne będą wszystkie zapisane pozycje.

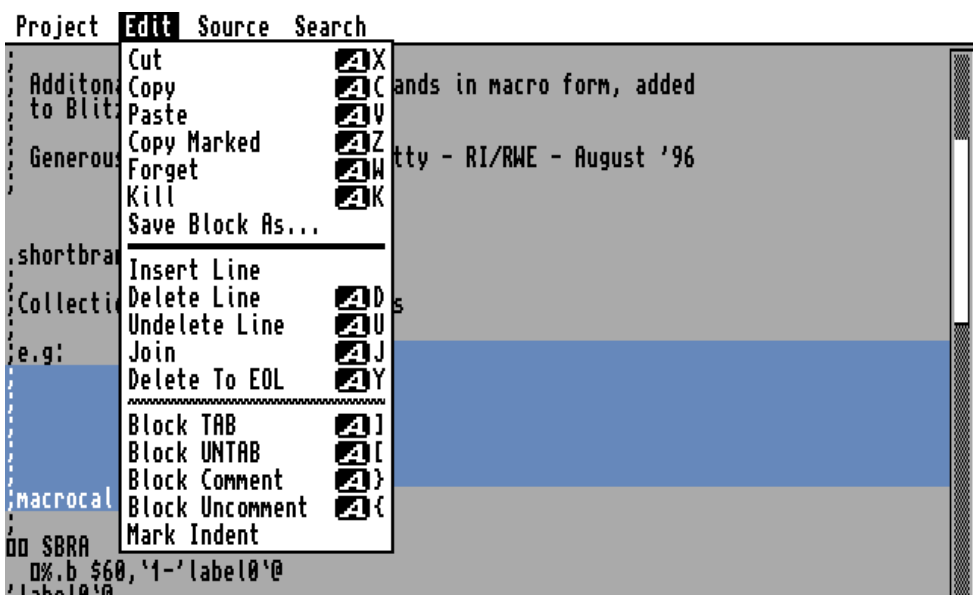
OPERACJE NA BLOKACH

Podczas edytowania programu często trzeba korzystać z bloków tekstu, aby przenosić lub kopiować większe fragmenty. Bloki zaznaczamy tak samo jak w innych edytorach, przy pomocy lewego klawisza myszki. Tutaj jednak możesz to robić za pomocą samej klawiatury. Wystarczy umieścić kursor na początku bloku, następnie nacisnąć klawisz Fl, a potem zmienić pozycję kursora i użyć klawisza F2. Zaznaczony w ten sposób blok wygląda tak:



Możesz mieć wrażenie, że na chwilę znika kursor, ale w rzeczywistości jedynie zlewa się z końcem bloku, ma bowiem ten sam kolor. Wystarczy użyć klawiszy kursora i przesunąć go w dół, aby przekonać się, że jest ustawiony na końcu wybranego bloku. Wadą tego rozwiązania jest fakt, że podczas zaznaczania – zanim naciśniesz F2 – nie widać fragmentu, który będzie wchodzić w skład bloku. Dopiero później całość jest podświetlana. Podczas zaznaczania tekstu myszką taki efekt nie występuje i cała zawartość jest aktualizowana na bieżąco.

Zaznaczony blok korzysta oczywiście z możliwości systemowego Schowka (czyli Clipboardu). Możesz wykonywać na nim wiele operacji, które są dostępne w menu górnym o nazwie „Edit". Zauważ, że przed wybraniem bloku większość z nich jest nieaktywna. Ponadto widzimy tutaj dużo większą ilość funkcji niż zwykle, w podobnych programach. Sytuacja po zaznaczeniu bloku powinna wyglądać tak:



Pierwsze trzy pozycje to typowe funkcje, które pozwalają wycinać, kopiować i wstawiać wybrane fragmenty tekstu w różnych miejscach. Możemy też usunąć wybór za pomocą opcji „Forget" lub skasować blok z treści programu używając „Kill".

Jeśli chcesz zapisać wybrany fragment na dysku, skorzystaj z funkcji „Save Block As...". Działa ona podobnie do zapisywania całego pliku, a więc na ekranie pojawi się zwykłe okno wyboru plików. Tym razem jednak zapiszesz jedynie fragment programu.

Możliwe jest też kasowanie całych linii lub wstawianie pustych. Służą do tego dwie funkcje widoczne niżej, odpowiednio „Delete Line" oraz „Insert Line". Gdy usuniesz linię przypadkowo możesz ją odzyskać przy pomocy opcji „Undelete Line". Funkcja działa jednak tylko w stosunku do ostatnio skasowanej linii. Jeśli wywołasz ją wielokrotnie uzyskasz wiele takich samych linii w kolejnych wierszach.

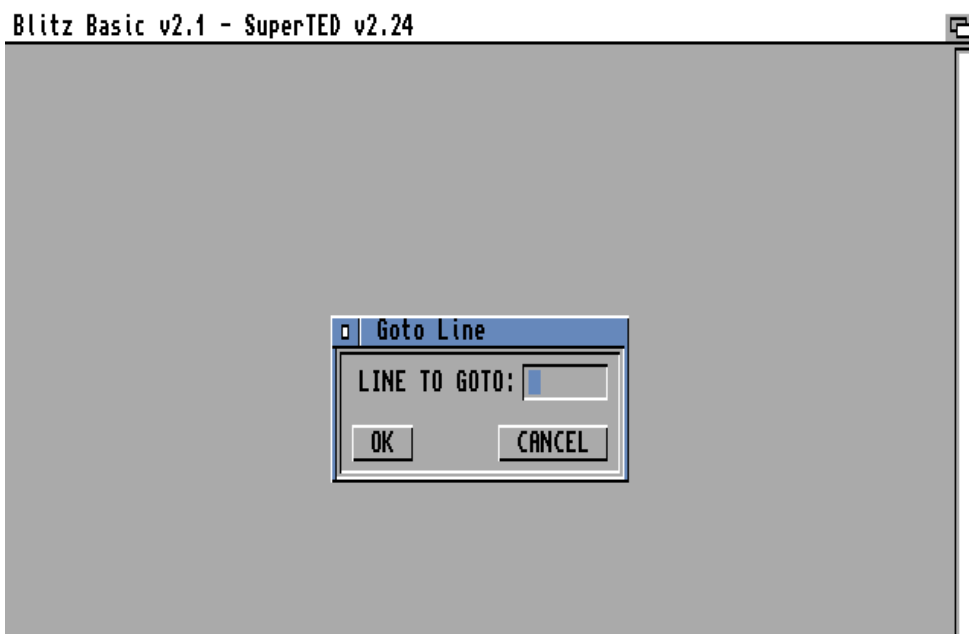
Możesz również połączyć ze sobą linię, w której znajduje się kursor z następną. Robimy to przy pomocy pozycji „Join". Zwróć uwagę, że w tym przypadku nie ma znaczenia jaka jest pozioma pozycja kursora, a jedynie pionowa. Linie zawsze są łączone przy uwzględnieniu ich pełnej zawartości, tak więc jeśli na początku lub na końcu znajdą się znaki SPACJI, nowa dłuższa linia również będzie zawierała odstępy.

Dzięki funkcjom „Block TAB" oraz „Block UNTAB" możesz natomiast kontrolować poziomą pozycję linii wchodzących w skład aktualnie wybranego bloku. Pierwsza z nich przesuwa wszystkie zaznaczone linie na prawo, druga – w przeciwnym kierunku. Domyślnie wielkość „marginesu" wynosi 2 znaki, możesz jednak to zmienić

wpisując nową wartość w polu „Tab Size” dostępnym w oknie konfiguracji.

Kursor możesz przemieszczać po ekranie za pomocą myszki lub klawiatury. Dostępne jest także menu górne o nazwie „Source”, w którym znajdziesz 3 ważne funkcje. Pierwsza - „Top” - przesuwa kursor na początek pliku wczytanego do edytora. Druga opcja - „Bottom” - działa odwrotnie, mianowicie przesuwa kursor do ostatniej linii pliku.

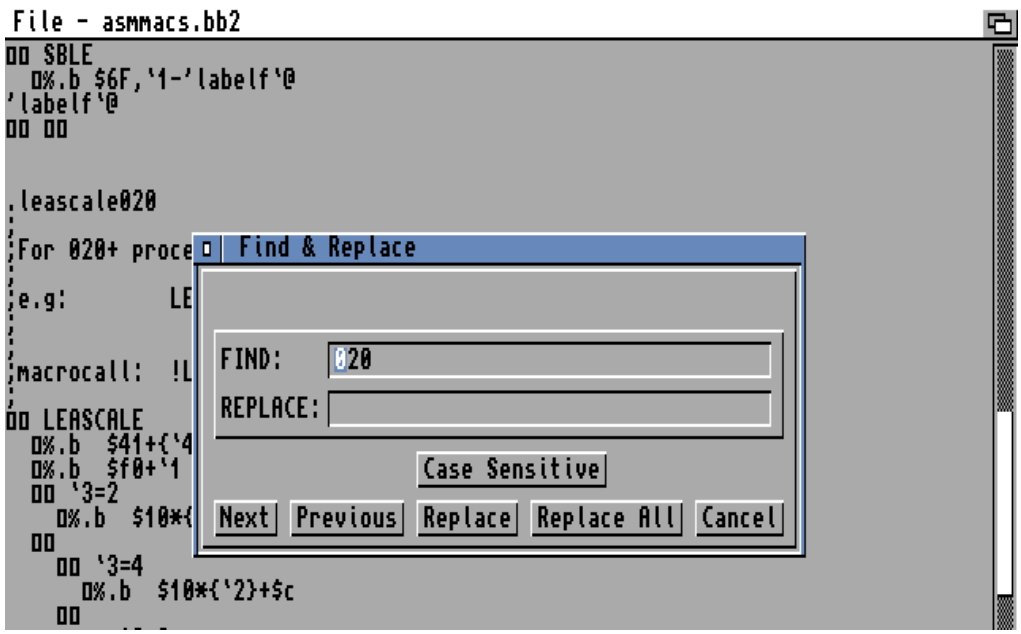
Poniżej widoczna jest jeszcze jedna opcja „Goto...”, dzięki której możesz umieścić kursor w określonej linii programu. Jest to wygodne szczególnie podczas edycji dłuższych listingów. Po wybraniu tej funkcji na ekranie pojawi się małe okno z polem tekstowym:



Wpisz numer pożądanej linii i naciśnij ENTER lub wybierz przycisk „OK”. Program zmieni pozycję kursora i automatycznie przewinie tekst tak, aby wskazany w ten sposób fragment był widoczny na ekranie. Oczywiście korzystanie z tej możliwości nie wyklucza „normalnej” zmiany pozycji kursora.

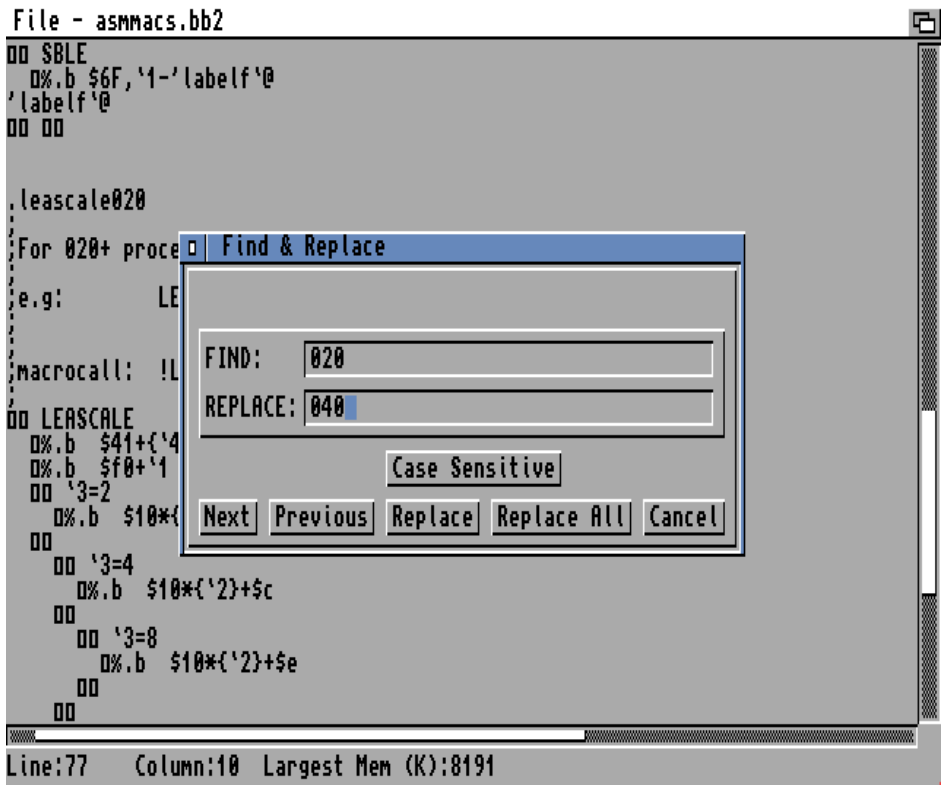
PRZESZUKIWANIE PROGRAMU

Jeżeli chcesz odnaleźć określony tekst w całym programie, skieruj się do menu górnego o nazwie „Search”. Pierwsza pozycja to „Find...”, której wybór powoduje pojawienie się nowego okna. Na przykład tak jak poniżej:



W polu „FIND” należy wpisać tekst jaki chcemy wyszukać. My użyliśmy wpisu „020”, bo interesuje nas odnalezienie fragmentów programu przeznaczonych dla procesora 68020. Dalej wskaż przycisk „Next”. Teraz zostanie przeszukany program w edytorze od pozycji kursora „w dół”. Jeśli chcesz uzyskać to samo w przeciwnym kierunku – od pozycji kursora „do góry” - użyj pola „Previous”.

Bardzo podobnie działa zastępowanie fragmentów tekstu. Aby zmienić nasz wpis „020” na „040”, należy uzupełnić pole „REPLACE”. Powinien znaleźć się w nim tekst, którym chcemy zastąpić poprzedni. U nas całość będzie wyglądała tak:



Teraz użyć przycisku „Replace”, aby odszukać i zastąpić jedno, najbliższe wyrażenie zgodne z polem „FIND”. Jeżeli natomiast chcesz zmienić wszystkie pasujące fragmenty skorzystaj z funkcji „Replace All” widocznej obok. W tym ostatnim przypadku na ekranie pojawi się dodatkowe okno, jak poniżej:

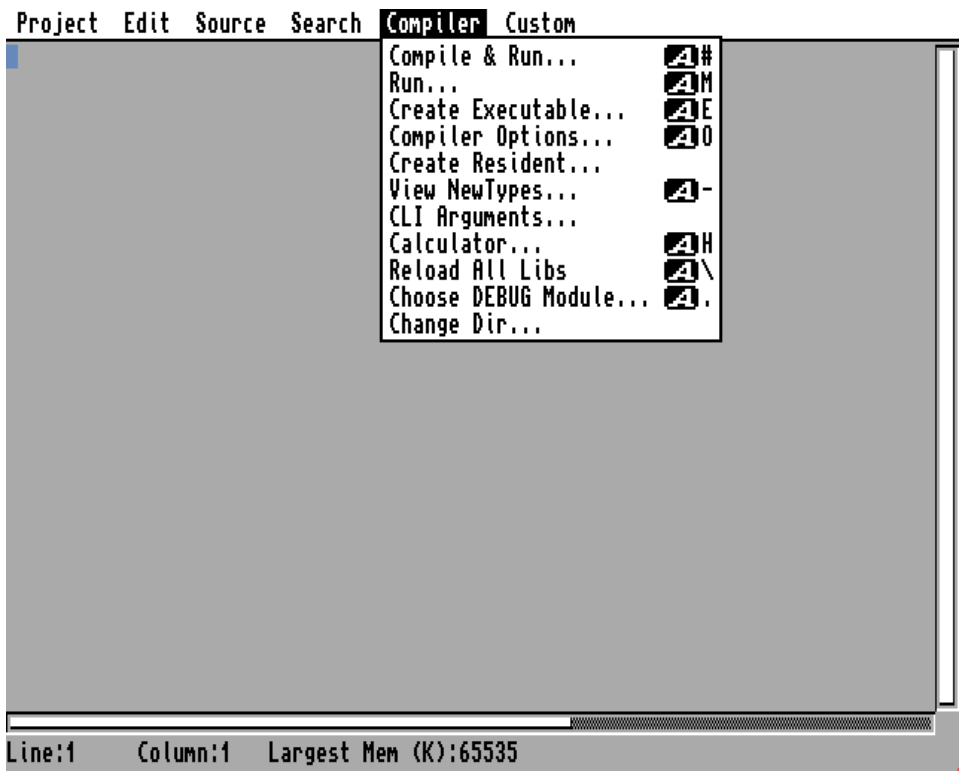
Przycisk „FROM HERE” spowoduje, że funkcja będzie działała od aktualnej pozycji kursora do końca tekstu. Pole „FROM TOP” natomiast to znak, aby przeszukać plik od początku, niezależnie od ustawienia kursora. Jeżeli chcesz anulować operację skorzystaj z trzeciego przycisku opisanego jako „Cancel”.

Zwróć uwagę, że niezależnie od wybranej opcji kursor zawsze będzie ustawiany na początku szukanego albo zastępowanego tekstu. Nie będzie on podświetlony, ale dzięki odpowiedniej pozycji kursora nie musisz zastanawiać się, gdzie znajduje się właściwy ciąg tekstowy.

Te same funkcje, co w oknie znajdziesz również w samym menu górnym „Search”. Możesz z nich korzystać analogicznie, bowiem działają tak samo jak przyciski w oknie. Dzięki nim nie zmienisz szukanego tekstu, za to uruchomisz wyszukiwanie lub zastępowanie szybciej niż z poziomu okna „Find & Replace”.

KOMPILACJA
I TESTOWANIE PROGRAMU

Do tej pory mówiliśmy o czysto edycyjnych funkcji „Teda". Za jego pomocą będziesz także uruchamiał programy, a więc musisz zapoznać się z opcjami widocznymi w menu górnym o nazwie „Compiler". Jest ono dużo bardziej skomplikowane w użyciu, dlatego najpierw spójrz na wszystkie pozycje:

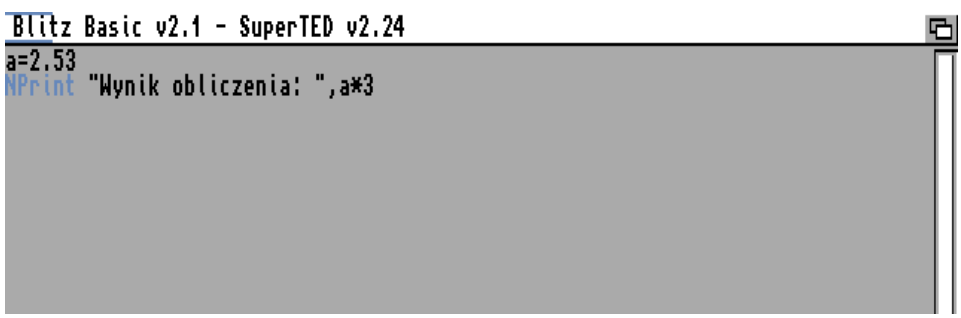


Nazwa menu wskazuje na czynność zwaną „kompilacją” programu (ang. Compile). Jest to proces, który polega na przekształceniu treści widocznej w edytorze na język maszynowy, czyli zestaw rozkazów przeznaczonych dla procesora, który można już uruchomić.

Dlatego nazwa opcji, z której będziemy korzystać brzmi „Compile & Run...”, co oznacza, że wykonane zostaną dwie główne czynności: kompilacja i uruchomienie programu. Oczywiście wszystko dzieje się pod warunkiem, ze listing nie zawiera błędów, w przeciwnym razie zobaczysz komunikat o problemie, który należy rozwiązać. Więcej na ten temat przeczytasz w rozdziale pod tytułem „Obsługa błędów”.

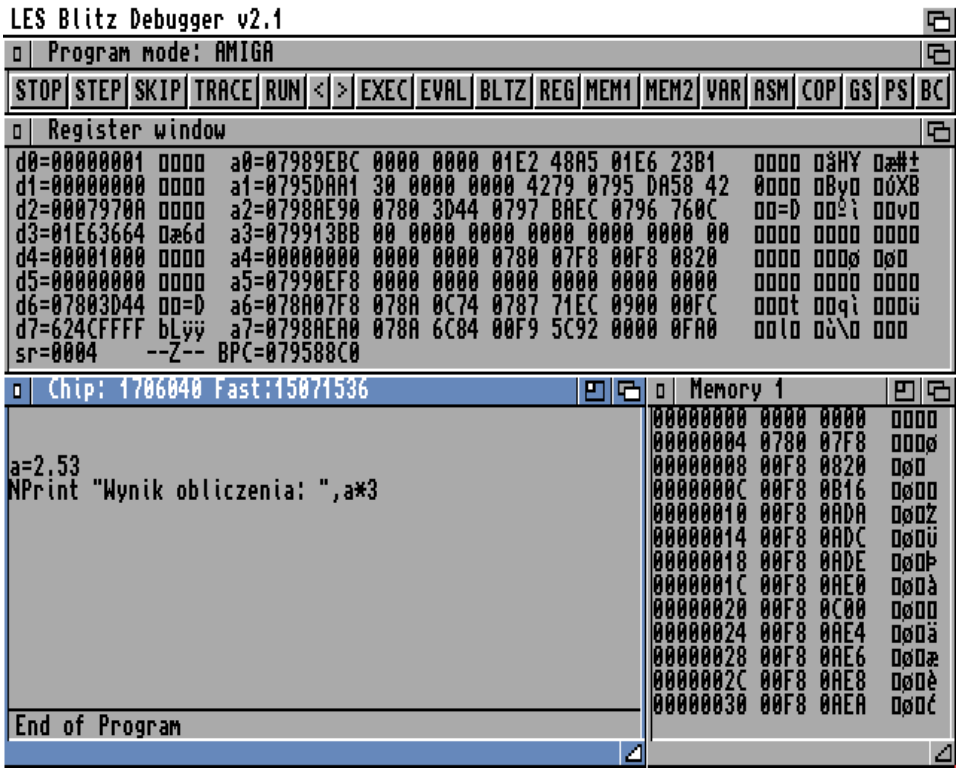
- Uruchamianie programu

Aby zobaczyć jak to działa w praktyce, spróbujemy uruchomić bardzo prosty program. Spójrz na nasz przykład:



Są to tylko dwie linie – deklaracja zmiennej oraz wyświetlenie wyniku obliczenia iloczynu dwóch wartości. Zastosowaliśmy czytelny zapis, który z pewnością nie sprawi Ci problemów.

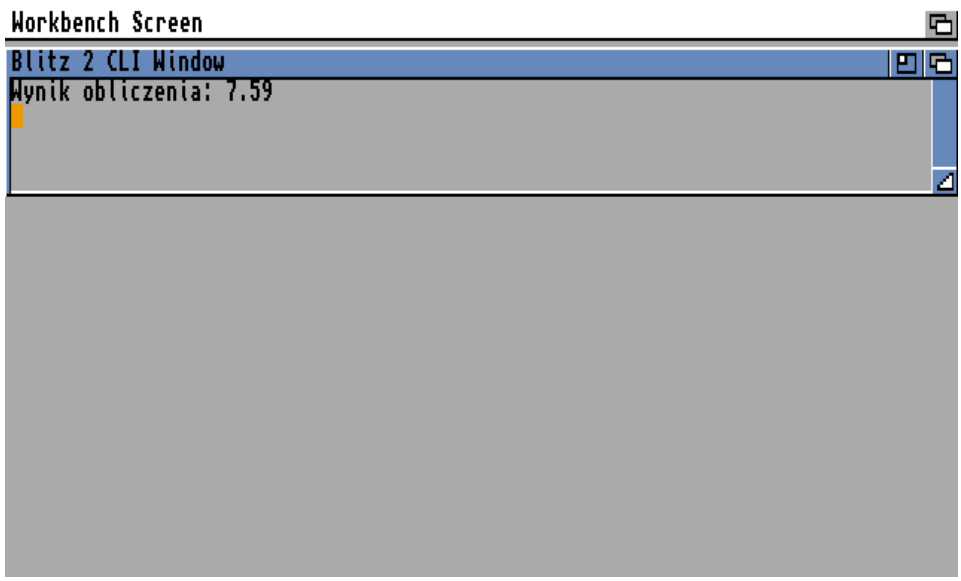
Wybierz teraz wspomnianą opcję „Compile & Run...” z menu górnego „Compiler”. Po chwili ekran zostanie przełączony i zobaczysz treść podobną to następującej:



Trzeba przyznać, że sytuacja zrobiła się dużo bardziej skomplikowana. To, co widać to tak zwany „debugger”, czyli moduł przeznaczony do analizy działania Twojego programu. Jedno z okien wyświetla znaną Ci treść, drugie zawartość pamięci, a trzecie – stan tak zwanych „rejestrów” procesora. Są to komórki pamięci, które służą do przechowywania informacji, na przykład wyników obliczeń lub innych tymczasowych danych.

Na same górze ekranu dostępny jest pasek narzędzi, za pomocą których możliwe jest wywoływanie różnych funkcji. O tych elementach szczegółowo będziemy mógł później, teraz ważne jest tylko, abyś wiedział jak sprawdzać działanie programu.

Wbrew pozorom jest to bardzo łatwe, wystarczy przełączyć ekran na blat Workbencha. Powinieneś zobaczyć poniższe okno:

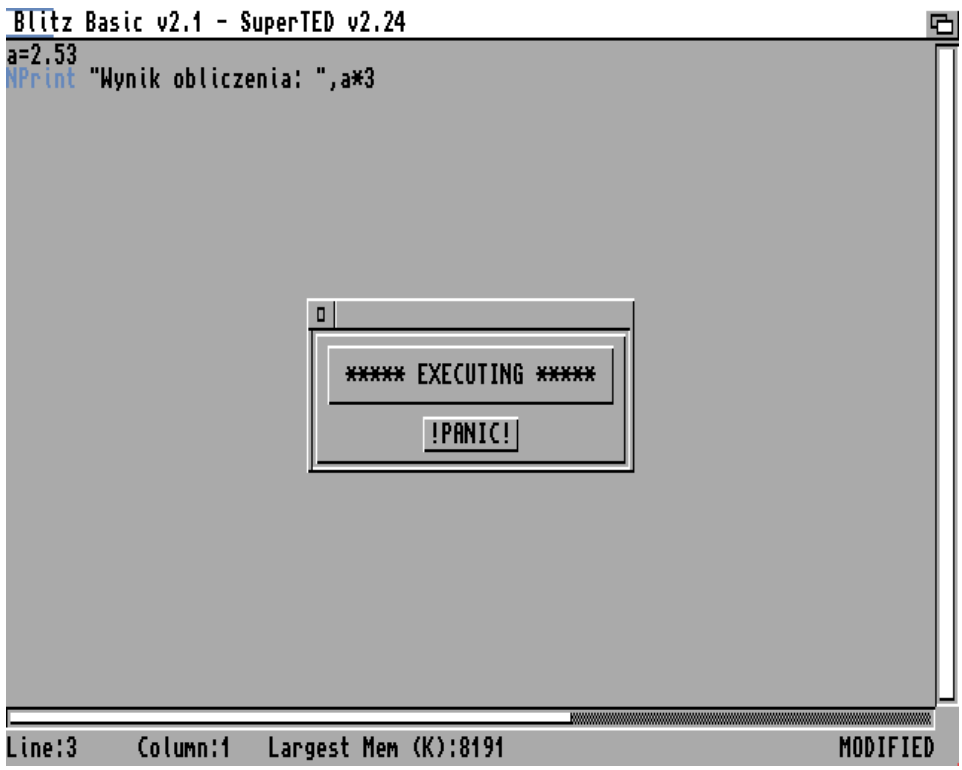


Jest to okno „CLI”„ które otwiera Blitz Basic przy uruchamianiu programu. To w nim widoczny jest rezultat jego działania. Tak się dzieje, bowiem nie wprowadziliśmy poleceń nakazujących otworzyć nowy ekran lub okno, ani utworzyć innych elementów interfejsu graficzne. Dlatego wynik naszego polecenia NPRINT został wypisany w domyślnym miejscu – w oknie AmigaDOS.

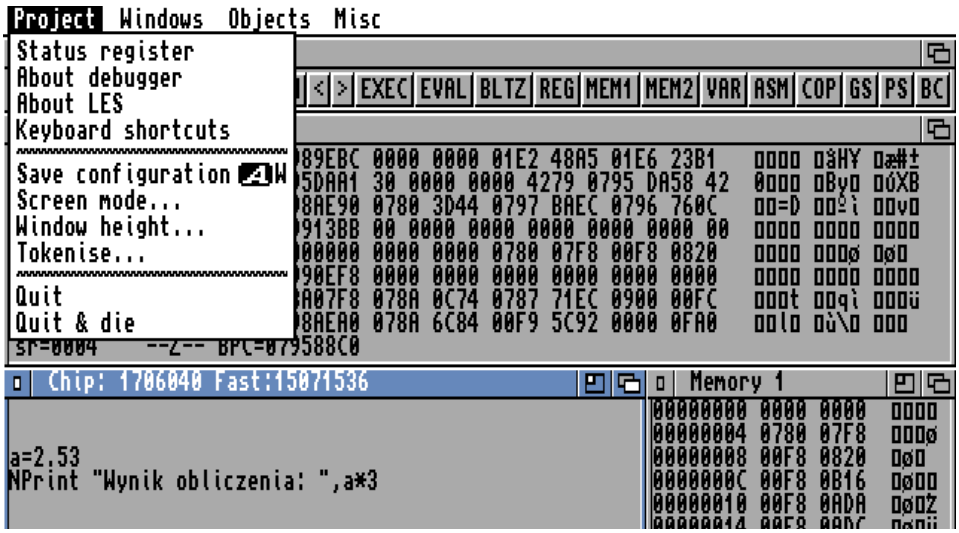
Gdy przełączysz ekran z powrotem do edytora „Ted”, w centralnej części widoczne będzie okno zawierające napis:

```
EXECUTING
```

oraz mniejszy przycisk poniżej. Zobacz kolejną ilustrację:



Tak się dzieje, bowiem formalnie listing nie został w żaden sposób zakończony. Aby przerwać działanie i powrócić do edycji programu przełącz ekran jeszcze raz do „debuggera” i wywołaj jego menu górne o nazwie „Project”. Zobaczysz listę opcji:

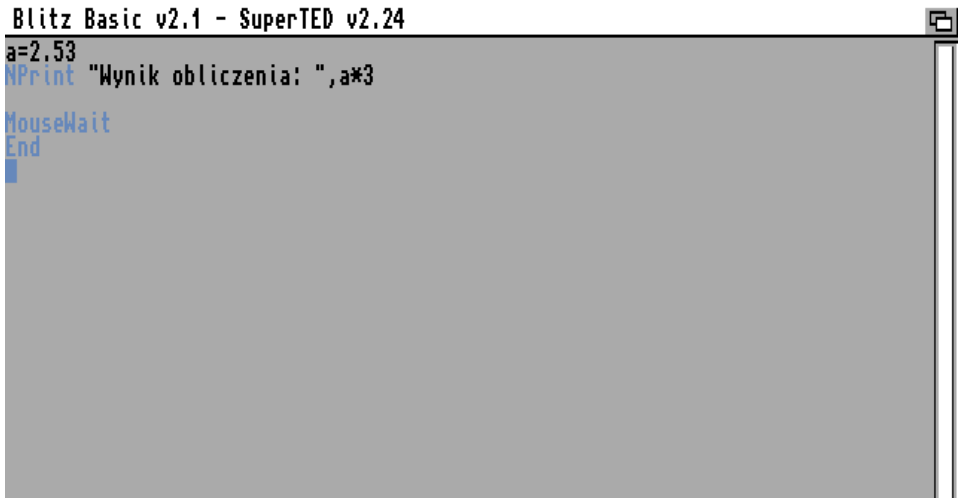


Wskaż pozycję „Quit”, aby kontynuować pracę nad programem. Jeśli chcesz uniknąć tego typu problemów wystarczy, że na końcu programu umieścisz linię z poleceniem END. Dzięki temu zobaczysz rezultat działania w oknie „CLI”, a następnie Blitz Basic automatycznie przełączy ekran z powrotem do edytora.

W tym wypadku jednak nie masz kontroli nad czasem, którym wyświetlane będzie okno „CLI”. Może się to stać na tyle szybko, że nie zdążysz zaobserwować rezultatów działania programu, a okno zostanie już zamknięte. Jest na to proste rozwiązanie. W przedostatniej linii, przed poleceniem END, wpisz:

```
MouseWait
```

Razem powinno to wyglądać podobnie do zapisu widocznego na ilustracji:



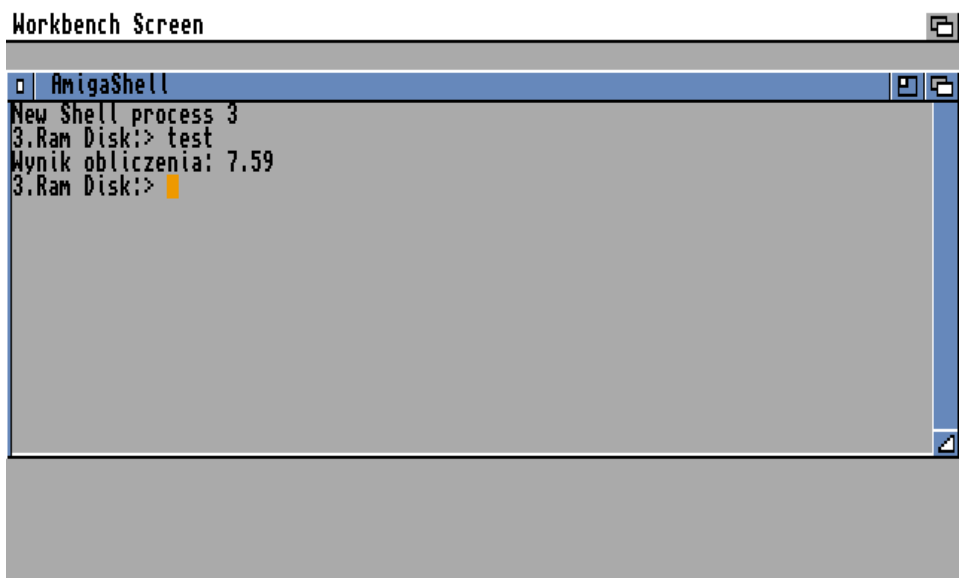
Teraz program będzie czekał, aż naciśniesz lewy przycisk myszki, dopiero później okno zostanie zamknięte i powrócisz do edytora. Jest to bardzo często wykorzystywany sposób testowania działania programów, ale warto dodać jedną ważną uwagę. Polecenie MOUSEWAIT nie powinno być używane w innych celach, bowiem powoduje zwolnienie pracy pozostałych programów działających jednocześnie. Używaj je wszędzie tam, gdzie chcesz sprawdzić rezultaty wprowadzonych słów, ale z gotowego programu należy je usunąć.

Zwróć też uwagę na fakt, iż poprawnie wpisane polecenia są podświetlane w edytorze na niebiesko. Jednocześnie po naciśnięciu klawisza ENTER poprawiana jest pisownia. Dzięki temu masz pewność, że nie popełniłeś błędu, a program staje się bardziej czytelny. Nazwy poleceń są wizualnie oddzielone od argumentów i zmiennych. Oczywiście nie ma to wpływu na działanie programu i efekt ten jest widoczny tylko w edytorze Blitz Basica.

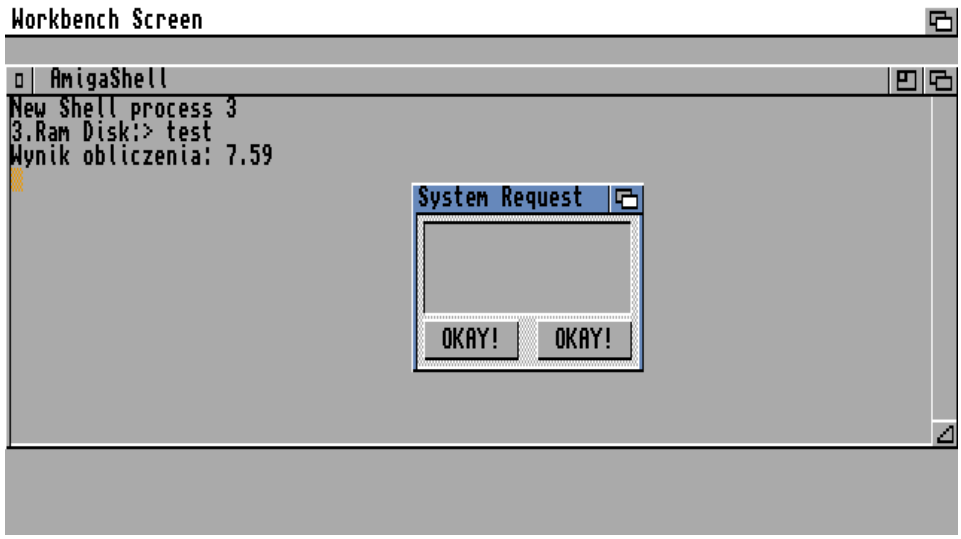
- Tworzenie programów wykonywalnych

Za pomocą opcji menu górnego „Compiler” możesz również zapisać program gotowy do uruchomienia poza edytorem. Wystarczy wybrać opcję o nazwie „Create Executable...”. Za jej pomocą utworzysz tak zwany plik wykonywalny, czyli program możliwy do uruchomienia w oknie AmigaDOS.

Po wskazaniu tej opcji na ekranie pojawi się okno wyboru, w którym należy wybrać katalog w jakim chcesz zapisać plik oraz wpisać jego nazwę. Po użyciu przycisku „Ok” w katalogu docelowym zobaczysz nową pozycję. Teraz otwórz okno „Shell” i wpisz nazwę pliku. U nas ma on nazwę „test”, a wynik działa wygląda tak jak poniższej ilustracji:



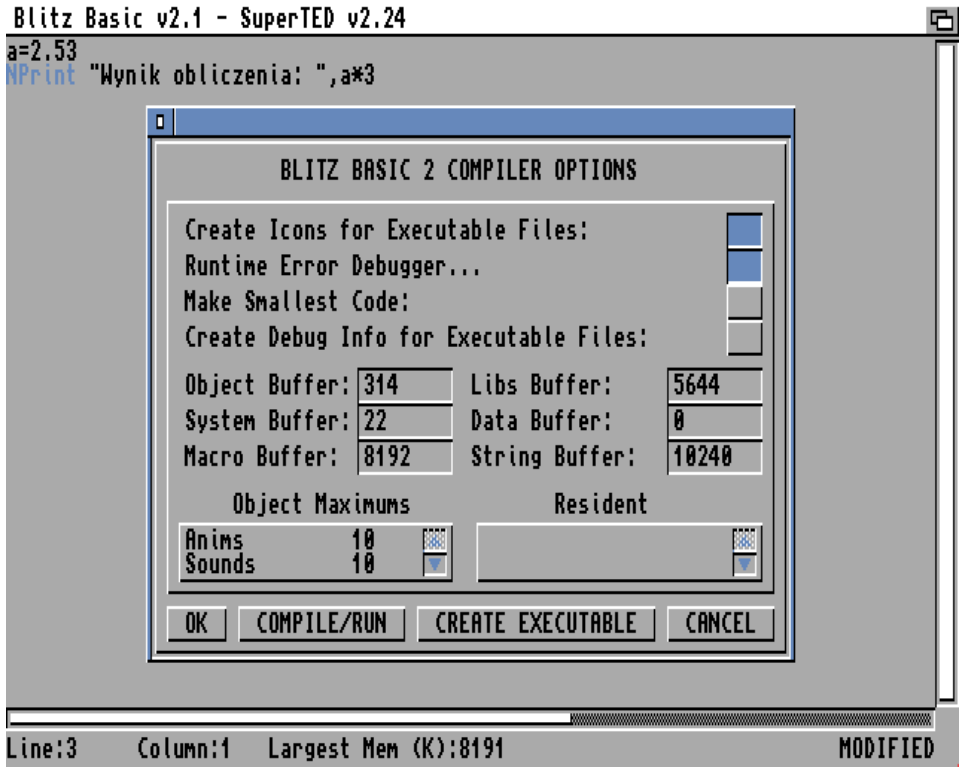
Program zachowuje się dokładnie tak, jak po uruchomieniu poprzez edytor Blitz Basica, czyli wyświetla komunikat i oczekuje na naciśnięcie lewego klawisza myszki. Możesz również zobaczyć mniejsze okno z przyciskami „OKAY”, jak poniżej:



Wybierz jeden z nich, aby zakończyć program. W ten sposób możesz zapisywać swoje programy. O bardziej zaawansowanym użyciu omówionych funkcji będzie pisać później.

- Parametry kompilacji

Domyślna kompilacja przyjmuje określoną grupę parametrów, których nie widać. Czasem zajdzie potrzebna ich zmiany. Jest to możliwe przy użyciu kolejne opcji umieszczonej w menu górnym „Compiler”. Tym razem wybierz pozycję „Compiler Options...”. Na ekranie pojawi się duże okno:



Pierwsza funkcja dotyczy tworzenia ikon do plików wykonywalnych. Domyślnie jest aktywna, dlatego wraz z Twoim programem w katalogu docelowym zapisany będzie plik z rozszerzeniem „.info". Na razie piszemy programy przeznaczone do uruchomienia w oknie AmigaDOS, dlatego funkcję „Create Icons for Executable Files" możesz wyłączyć.

Zwróć też uwagę na opcję umieszczoną poniżej, czyli „Runtime Error Debugger". Dzięki niej po uruchomieniu program pojawia się ekran „debuggera". Jeśli nie chcesz go oglądać wyłącz również tę funkcję.

Musisz jednak pamiętać , że „debugger” jest modułem, który nie tylko wskazuje ważne informacje na temat działania programu, ale również pomaga w sytuacji, gdy wystąpi błąd. Jeśli go wyłączysz, Twój program może się zawiesić i nie zobaczysz komunikatu o błędzie. Gdy piszesz proste programy może nie mieć to większego znaczenia, ale w miarę rozbudowy listingu powinieneś korzystać z tej funkcji. Działaniem i ustawieniami „debuggera” zajmiemy się też później.

Może Ci się to przydać także wtedy, gdy używasz Amigi z małą ilością pamięci, bowiem „debugger” powoduje jej większe zużycie. Pamiętaj jednak, że w przypadku bardziej skomplikowanych programów moduł ten będzie jedna przydatny, na przykład do „śledzenia” kolejno wykonywanych poleceń. Będziemy o tym mówić później.

Zainteresuj się dodatkowo funkcją opisaną jako „Make Smallest Code”. Gdy ją włączysz, kompilator będzie starał się tworzyć zawsze pliki o najmniejszej objętości. Nie będzie to miało większego znaczenia przy uruchamianiu prostych listingów, ale po wczytaniu bardziej rozbudowanego programu możesz uzyskać ciekawe rezultaty.

W tym wypadku bardzo wiele zależy od sposoby pisania programu, jego poszczególnych części oraz użytych algorytmów, dlatego oszczędność miejsca nie zawsze będzie możliwa.

Pozostałe pola w oknie potraktuj czysto informacyjnie. Poniżej wspomnianych przycisków widać wartości oznaczające bufory dla różnego rodzaju operacji i zmiennych. Na tym etapie nie musisz się nimi przejmować.

Zwróć jednak uwagę, że za pomocą przycisków umieszczonych w dolnej części okna możesz uruchomić program lub utworzyć plik wykonywalny bez potrzeby ponownego wywoływania menu górnego. Służą do tego pola oznaczone odpowiednio:

- COMPILE/RUN,
- CREATE EXECUTABLE.

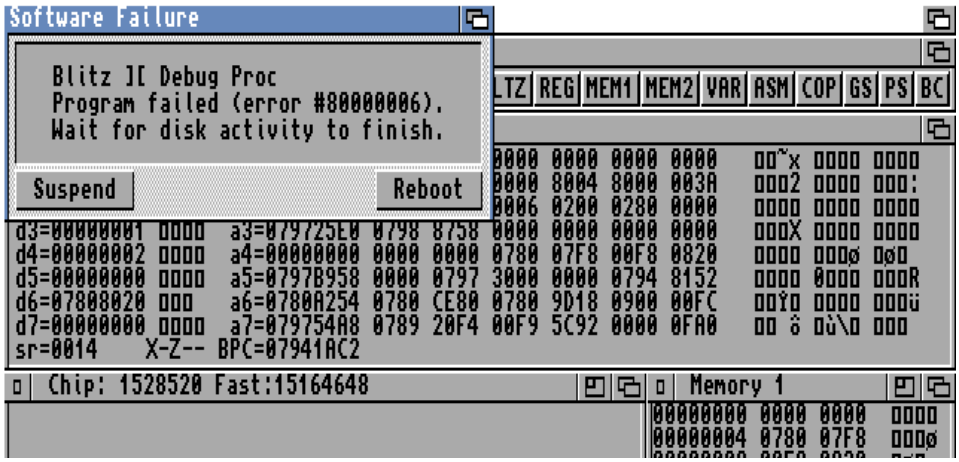
Ich obsługa jest identyczna jak funkcje menu „Compiler".

Jeśli chcesz wyłącznie zmienić konfigurację, aby zamknąć okno wskaż przycisk „OK". Gdy Twoją intencją jest anulowanie zmian, wybierz „CANCEL". Pod tym względem okno działa analogicznie do funkcji konfiguracyjnych, które można znaleźć w wielu innych programach.

Kompilacja i uruchomienie programu jest procesem złożonym, o którym będziemy pisać w dalszej części książki. W tym momencie powinieneś umieć wykonać kompilację programu oraz ustawić podstawowe opcje do uruchomienia.

Pamiętaj, że Blitz Basic operuje na składnikach systemu operacyjnego i w wielu przypadkach Twój program może doprowadzić do zawieszenia komputera lub pojawienia się okna „Software Failure". Ponadto nie powinieneś uruchamiać drugi raz programu bez wyłączenia „debuggera".

Jeśli więc na ekranie zobaczysz okno podobne do poniższego:



oznacza to, że popełniłeś błąd i musisz zresetować komputer. Następnie powinieneś określić, czy problem został wywołany błędnym stanem poleceń w programie, czy tylko nieprawidłowo uruchomiłeś funkcję dostępną w edytorze czy module „debuggera”. W tym celu możesz utworzyć plik gotowy do uruchomienia w oknie AmigaDOS lub zmienić opcje konfiguracji kompilatora.

SKRÓTY KLAWIATUROWE

Niektóre funkcje „Teda" mogą być niewygodne do wywoływania za pomocą myszki. Jest to podstawowy sposób obsługi, ale nic nie stoi na przeszkodzie, aby korzystać z odpowiednich kombinacji klawiszy. Zwykle w programach dla Amigi opcje z menu górnego można uruchamiać za pomocą klawisza AMIGA i odpowiedniego znaku. Nie inaczej jest w tym przypadku.

Oto lista dostępnych funkcji:

AMIGA + A
Powoduje wywołanie funkcji zapisywania pliku, a więc w ten sposób zapiszesz program pod nową nazwą. Jest to odpowiednik opcji „Save As..." z menu „Project".

AMIGA + B
Umieszcza kursor w ostatniej linii treści programu.

AMIGA + D
Usuwa linię tekstu, na którym znajduje się kursor.

AMIGA + F
Wywołuje funkcję przeszukiwania zawartości programu (opcja „Find..." w menu górnym o nazwie „Search").

AMIGA + G
Umieszcza kursor w określonej linii programu, podanej przez użytkownika.

AMIGA + I

Pozwala wczytać inny plik i wstawić jego zawartość w aktualnej pozycji kursora. Pozostała treść znajdująca się pod kursorem jest przesuwania w dół.

AMIGA + J

Pozwala połączyć dwie następujące po sobie linie programu.

AMIGA + L

Wywołuje funkcję wczytania pliku z dysku, z utratą aktualnej zawartości..

AMIGA + N

Pozwala wyszukać kolejny fragment podany wcześniej w oknie „Find...".

AMIGA + Q

Wyłącza edytor Blitz Basica. Powrócisz do okna lub ekranu, z którego uruchomiłeś program. Zwykle jest to ekran Workbencha lub okno AmigaDOS.

AMIGA + R

Wywołuje funkcję zastępowania fragmentów dwóch ciągów tekstowych. Jest to odpowiednik opcji „Replace" znajdującej się w menu górnym o nazwie „Search".

AMIGA + S

Pozwala zapisać aktualną treść programu na dysku, ale pod tą samą nazwą. Wyjątkiem jest sytuacja, w której plik jest

zapisywany pod raz pierwszy na dysku. Jest to odpowiednik opcji „Save” z menu o nazwie „Project”.

AMIGA + T
Umieszcza kursor na początku programu.

AMIGA + W
„Wyłącza” podświetlenie zaznaczonego fragmentu tekstu, gdy używane są operacje na blokach. Możesz o nich przeczytać także w oddzielnym punkcie.

AMIGA + Y
Usuwa zawartość linii w pozycji kursora, aż to napotkania znaku końca linii (ang. End-Of-Line). Z tego względu może kasować większą lub mniejszą część programu, bowiem znak EOL jest standardowo niewidoczny.

AMIGA + ?
Wywołuje okno informacji o edytorze, w którym znajdziemy między innymi komunikat o nazwie tak zwanego ekranu publicznego, na którym działa program. Ponadto wskazany zostanie port język ARexx, o ile jest dostępny. W przeciwnym razie wyświetlony zostanie symbol „N/A” (ang. Not Available) wskazujący na brak aktywnego portu ARexxa.

AMIGA +]
Przemieszcza wybrany blok w prawo, czyli powoduje utworzenie lub zwiększenie tak zwanego „wcięcia”. Przed jej użyciem należy podświetlić blok, na przykład za pomocą myszki.

AMIGA + [
Funkcja odwrotna do poprzedniej, pozwala zmniejszyć „wcięcie”.

AMIGA + }
Umożliwia automatycznie dodanie znaku średnika na początku linii „zaznaczonego” bloku tekstu. Dzięki temu linie te będą traktowane jak komentarze i nie będą wykonywane po uruchomieniu programu.

AMIGA + [
Funkcja odwrotna do poprzedniej, usuwa znaki średnika, czyli ponownie „aktywuje” wskazane linie w bloku. Pozostała zawartość nie jest modyfikowana.

PIERWSZY PROGRAM

ZMIENNE

W programach mamy do czynienia z danymi tekstowymi i liczbowymi. Możemy je definiować jako tak zwane "zmienne", a także wykonywać na nich wiele różnych operacji. Można to przyrównać do danych z jakich korzystamy przy rozwiązywaniu zadań matematycznych, podstawiając określone wartości do wzorów. Podobnie będzie to działać w Twoim programie.

Aby wpływać na działanie programu zachodzi konieczność użycia wielu różnych zmiennych, które zwykle będą przyjmować różne wartości liczbowe lub tekstowe. Blitz Basic obsługuje 5 standardowych typów liczbowych o różnym zakresie i dokładności. Oto ich lista:

```
- Byte    (.b)      +-128           liczby całkowite
- Word    (.w)      +//-32768       liczby całkowite
- Long    (.l)      +/-2147483648   liczby całkowite
- Quick   (.q)      +/-32768.0000   1/65536
- Float   (.f)      +/-9*10/\18     1/10^18
```

Zwróć uwagę, że korzystanie z różnych typów zmiennych powoduje nie tylko zmianę dokładności przechowywanych informacji, ale także szybkości wykonywania operacji.

Zmienna jest przypisana do określonego typu poprzez dodanie odpowiedniego przyrostka (sufiksu) do jego nazwy Trzeba to zrobić tylko na początku, a kolejne odniesienia mogą być mniej szczegółowe, chyba że będzie to zmienna tekstowa. Szczególnym zbiorem zmiennych są tak zwane „tablice”, o których możesz przeczytać w oddzielnym punkcie pod tym samym tytułem.

Blitz Basic umożliwia także stosowanie wielu rozszerzeń swoich funkcji, możliwe jest między innymi definiowanie nowych typów zmiennych, które będą zbiorem kilku standardowych typów.

Trzeba jednak pamiętać, aby przy tworzeniu zmiennych i tablic przestrzegać kilku podstawowych zasad. Ich nazwy mogą mieć w zasadzie dowolną długość, ale powinny jasno opisywać działanie. Zbyt długie symbole będą niewygodne w użyciu. Każda nazwa powinna rozpoczynać się literą lub znakiem podkreślenia oraz zawierać wyłącznie litery i cyfry. Ponadto nazwy nie mogą pokrywać się z pozostałymi poleceniami Blitz Basic, bowiem zaburzone zostałoby działanie programu.

Ciekawostką jest fakt, iż rozróżniana jest również pisownia, a więc wielkość liter. Na przykład dwie poniższe nazwy:

```
punktjeden
```

oraz

```
PunktJeden
```

będą potraktowane jako dwie osobne pozycje. O tych zasadach należy zawsze pamiętać, w przeciwnym razie Twój program nie będzie działał prawidłowo. Zwykle w takich sytuacjach bez zmiany nazw zmiennych nie będziesz w stanie wyeliminować wszystkich błędów.

DEKLAROWANIE ZMIENNYCH

Każda zmienna posiada swój symbol oraz przyporządkowaną wartość. Aby ją utworzyć musisz wpisać linię podobną do poniższej:

```
a=10
```

Spowoduje ona , że zmienna liczbowa "a" będzie zawierać wartość 10. Jak widzisz nie jest to skomplikowane, każdą nazwę trzeba jednak wpisać bardzo dokładnie. Jeśli Twoja zmienna ma przyjąć liczby inne niż całkowite musisz zastosować się do listy wymienionej w punkcie „Zmienne”. Na przykład, po wprowadzeniu linii:

```
a.q=1.3
```

zmienna „a.q” przyjmie liczbę 1,3, a więc ułamek. Wynika to z typu zmiennej, który wybraliśmy wpisując końcówkę „.q”. Zmienne mogą tez zawierać wynik obliczenia:

```
a.q=50/13
```

Teraz nowa zmienna „a.q” będzie zawierała wartość 3,8461. Jest to wygodny sposób wprowadzania skomplikowanych liczb, bowiem nie musisz ich ręcznie obliczać, a precyzja z jaką będą przechowywane w programie zależą od typu zmiennej.

Podobnie możliwe jest utworzenie zmiennej zawierającej tekst. W tym wypadku jej symbol musi być uzupełniony o znak "dolara" ($), czyli zamiast zmiennej "a" musisz zastosować wpis "a$", przykładowo tak:

```
a$="Amiga"
```

Teraz nowa zmienna tekstowa "a$" będzie zawierała wyraz "Amiga". Zwróć uwagę, że w ten sam sposób do zmiennej tekstowej możesz przyporządkować liczbę:

```
a$="10"
```

Będzie ona traktowana cały czas jako ciąg tekstowy, dlatego nie będziesz mógł wykonywać na niej obliczeń. Możliwe będzie jednak wykonywanie innych działań, charakterystycznych dla zmiennych tekstowych. Powiemy o tym za chwilę.

OPERACJE
NA ZMIENNYCH LICZBOWYCH

Możesz sprawić, aby komputer zmienił wartość zmiennej liczbowej według określonego schematu, na przykład dodał lub odjął określoną wartość. Wystarczy użyć następującej linii:

```
a=a+1
```

lub

```
a=a-1
```

Może ona także zawierać więcej niż jedną operację, bez problemu użyjesz również nawiasów. Oto kolejny przykład:

```
a=(a+2)*10
```

Oczywiście zachowana zostanie zwykła kolejność działań matematycznych. Zwróć uwagę, że po znaku równości, oprócz wzoru, zapisaliśmy sam symbol zmiennej „a". Jak to działa? Komputer bierze pod uwagę poprzednio przypisaną wartość i wykonuje obliczenie według formuły. W przypadku pierwszego zapisu, do aktualnej wartości zmiennej „a" jest dodawana lub obejmowana liczba 1, a całość zapisywana w ramach tej samej zmiennej. Aby to lepiej zrozumieć operację możesz utworzyć drugą zmienną i zastosować zapis w postaci:

```
b=a+1
```

Jednak program nie powinien zawierać zbyt dużej ich ilości bez wyraźnej potrzeby. Ponadto taki zapis nie ma sensu, jeśli chcesz zmieniać wartości zmiennych wielokrotnie, bo za każdy razem musiałbyś tworzyć kolejne pozycje.

Wynik obliczenia może być też uzależniony od większej ilości zmiennych, a nie tylko jednej. Nasz poprzedni wzór możemy zamienić na taki:

```
c=(a+b)*10
```

Wcześniej należy oczywiście zadeklarować dodatkową zmienną „b". Cały program może więc wyglądać następująco:

```
a=5
b=8

c=(a+b)*10
```

Pamiętaj, że na razie tylko określamy wartości zmiennych. Jak wyświetlać je na ekranie dowiesz się między innymi w punkcie „Polecenia i składnia".

Warto dodać, że potęgowanie jest uzyskiwane poprzez znak „^". Jest to typowy zapis dla dialektu języka Basic, weź jednak pod uwagę, że w przypadku rozbudowanych wzorów będziesz musiał często stosować pojedyncze lub podwójne nawiasy. Może to wyglądać na przykład tak:

```
c=2^(a+b)*((15*b)-5)
```

Jest to oczywiście tylko przykład, ale dobrze obrazuje sposób działania. Nawiasy powinny być stosowane wszędzie tak, gdzie chcesz uzyskać większą czytelność odnośnie kolejności wykonywania obliczeń.

W programach możesz używać ułamków, zapisując je w formie dziesiętnej, przykładowo:

```
a=2.36
```

Ułamek możesz bardzo łatwo zaokrąglić w razie potrzeby, wystarczy użyć funkcji INT(). Cały program może wyglądać w następujący sposób:

```
a=2.36
b=Int(a)

NPrint a
NPrint b
```

Zwróć uwagę, że tworzymy tu nową zmienną „b", ale nie jest to konieczne. Jeśli nie zależy Ci na zachowaniu obu wyników, możesz usunąć ostatnią linię, a następnie zamienić zaokrąglenie na wpis:

```
a=Int(a)
```

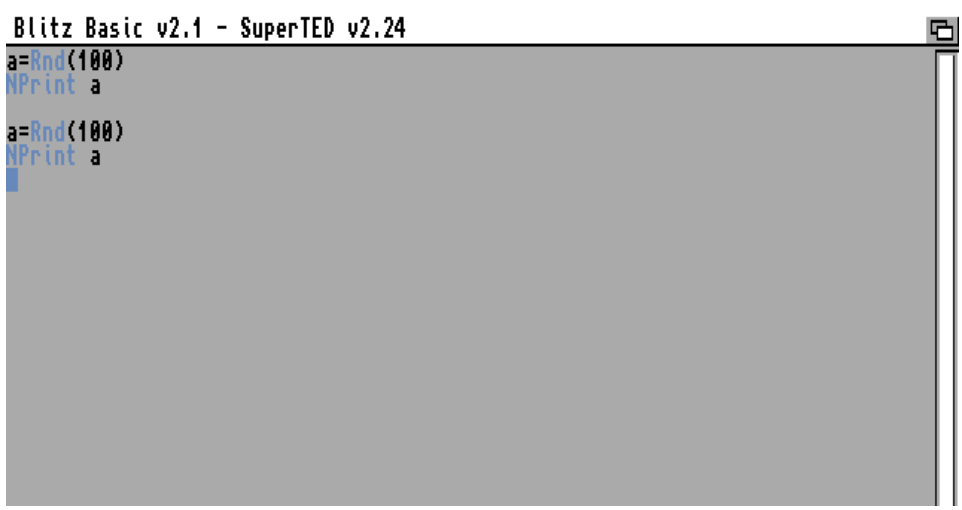
Na ekranie zobaczysz tę samą liczbę, czyli - u nas „2". Pamiętaj, że funkcja INT nie zachowuje się tak jak typowe zaokrąglenie „w dół" lub „w górę", a jedynie odcina ułamkową część wartości.

W razie potrzeby możemy również wygenerować liczbę losową korzystając z funkcji RND(). Jako argument należy podać zakres wartości, który będzie brany pod uwagę. Przykładowo:

```
a=Rnd(100)
```

spowoduje, że maksymalną dopuszczalną wartością będzie liczba 100, a przy każdym wywołaniu funkcji możesz uzyskać inny wynik. Tak się dzieje niezależnie od tego, czy program uruchomisz dwa razy, czy powtórzysz kilka linii w jednym listingu i dopiero później uruchomisz całość.

Pamiętaj jednak, że liczba „losowana” jest w momencie wykonania linii z zapisaną nazwą funkcji, tak więc nie wystarczy wprowadzić dwa razy polecenia NPRINT. Prawidłowy przykład przedstawiamy na następnej ilustracji:



Więcej na temat funkcji oraz ich sposobu używania możesz przeczytać w punkcie im poświęconym.

OPERACJE
NA ZMIENNYCH TEKSTOWYCH

W ramach zmiennych możesz zapisać dowolny tekst. Możesz na nich także wykonywać innego rodzaju operacje, niedostępna dla liczb. Najprościej połączyć dwie zmienne, bowiem wykonujemy to identycznie jak dla liczb, czyli na przykład:

```
a$="Amiga"
b$="1200"

wynik$=a$+b$
```

Jedyną różnicą jest fakt użycia znaku dolara ($). Taki zapis jest uniwersalny, co oznacza, że możesz go stosować zarówno podając nazwy zmiennych, jak i wpisywać tekst bezpośrednio. Może to wyglądać następująco;

```
wynik$="Amiga"+b$
```

lub

```
wynik$="Amiga"+"1200"
```

Na liczbie zapisanej w takiej formie nie wykonasz żadnego obliczenia. Możesz natomiast wykonać konwersję ciągu tekstowego na liczbę. Aby było to możliwe trzeba użyć funkcji o nazwie VAL(). Odnosząc to do naszego przykładu, linie:

```
b$="1200"
b=Val(b$)

NPrint b
```

spowodują, że zmienna „b” będzie zawierała liczbę 1200, co sprawdzamy za pomocą typowego polecenia NPRINT. Czasem możesz potrzebować wykonać operację odwrotną, na przykład łącząc dwa ciągi tekstowe. Do Twojej dyspozycji pozostaje kolejna funkcja STR$(). Należy ją używać w analogiczny sposób, podając zmienną tekstową jako parametr. Na przykład:

```
b=1200
b$=Str$(b)

NPrint b$
```

Jeszcze raz zwracamy uwagę, że zmienne poddawane operacjom muszą mieć ten sam rodzaj. Dlatego dostosowanie ich rodzaju „w obie strony” może być bardzo przydatne, choć początkowo możesz tego nie przewidywać.

Każdą zmienną tekstową możesz również zmodyfikować tak, aby tekst był pisany małymi lub WIELKIMI literami. Służą do tego funkcje o poniższych nazwach

```
UCase$()
```

oraz

```
LCase$()
```

Korzystamy z nich w identyczny sposób, podając w nawiasie nazwę zmiennej. Przykładowo, aby nasz napis „Amiga” być pisany WIELKIMI literami trzeba zastosować taki program:

```
a$="Amiga"
b$=UCase(a$)

NPrint b$
```

Nazwy funkcji pochodzą od angielskich słów „Lowercase” oraz „Uppercase”, co powinno Ci ułatwić zapamiętanie ich działania.

Ciągi tekstowe może również powielać. Jeśli chcesz utworzyć drugą zmienną zawierającą określoną ilość powtórzeń naszego słowa „Amiga”, wystarczy wpisać linie podobne do poniższych:

```
a$=”Amiga”
b$=String$(a$,5)

NPrint b$
```

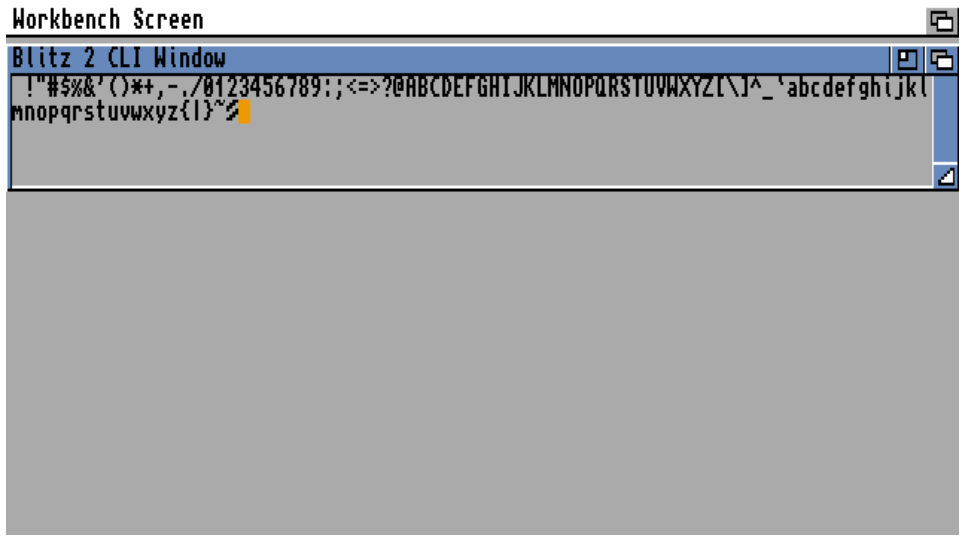
Jak łatwo można się domyślić, pierwszy parametr funkcji STRING$() to nazwa zmiennej źródłowej, a drugi – ilość powtórzeń. Zwróć jednak uwagę, że pomiędzy kopiami tekstu nie będzie zachowany żaden odstęp. Czasem możesz chcieć uzyskać taki efekt, ale najczęściej jednak potrzebujemy utrzymać czytelność tekstu.

Możesz w tym celu umieścić znak SPACJI na końcu kopiowanego słowa lub zdefiniować drugą zmienną zawierającą separator. Następnie połącz zmienne przed wykonaniem duplikatu.

Ręczne wprowadzanie odstępu nie jest jednak wygodne, ponadto nie będziesz mógł automatycznie zmienić znaku odstępu, chyba ze znowu wpiszesz go z klawiatury. Aby ułatwić sobie zadanie i uczynić Twój program bardziej uniwersalny proponujemy poznanie kolejnej funkcji CHR$(). Za jej pomocą można zapisać znak posiadający określoną wartość w tak zwanej tablicy ASCII. Jest to kod przyporządkowujący liczby literom alfabetu, cyfrom i innym znakom, które niekoniecznie są dostępne bezpośrednio z klawiatury. To ostatnie jest bardzo ważne, jeżeli nie chcesz nakładać ograniczeń na swój program.

Rzecz jasna teraz powinieneś sprawdzić, jakie liczby są przypisane do konkretnych znaków. Dlatego napiszemy krótki program realizujące tę funkcję. Wykorzystamy poznane do tej pory polecenie PRINT, pętlę FOR … NEXT oraz deklaracje kilku zmiennych. Oto nasza propozycja:

Zobacz jak wygląda efekt uruchomienia tego listingu:



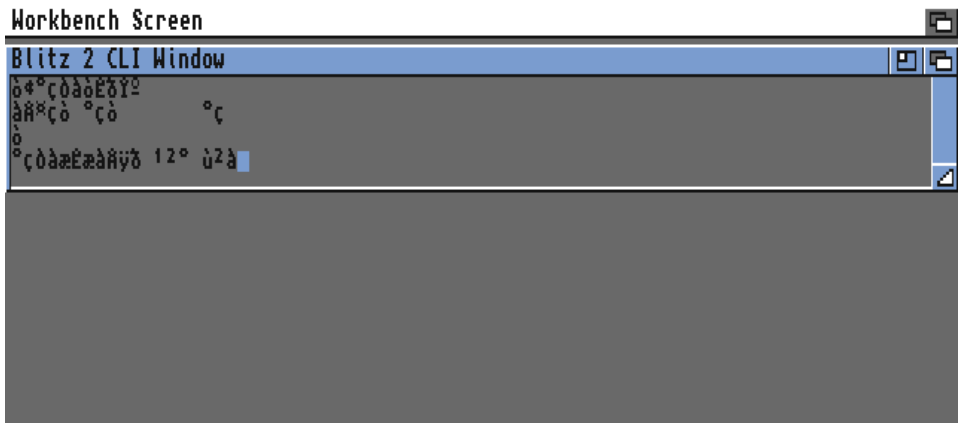
Teraz bez problemu zastosujesz odpowiednie znaki, aby wstawić separator w środku tekstu. Możesz to zrobić w poniższy sposób:

```
a$="Amiga"+Chr$(32)
```

Taki zapis spowoduje, że po nazwie „Amiga" znajdzie się odstęp (SPACJA) bez żadnych dodatkowych znaków.

Operacje na zmiennych tekstowym mogą być jeszcze bardziej skomplikowane. Co zrobić, gdy utworzymy długi tekst i będzie on zapisany jako jedna pozycja? Taka sytuacja może mieć miejsce na przykład podczas odczytywania danych z dysku. Jak to zrobić przeczytasz w punkcie pod tytułem „Obsługa plików".

Rezultatem może być zmienna zawierająca fragment pliku tekstowego. Po wczytaniu zawartości do zmiennej i wyświetleniu jej zawartości zobaczysz niezbyt czytelną formę. W ekstremalnej sytuacji możesz zobaczyć coś takiego:



W takiej sytuacji najlepszym rozwiązaniem jest odczytywanie krótkich fragmentów pliku i analiza zawartości. Aby pogrupować informacje będziesz potrzebować „pociąć” zmienną na bardziej zrozumiałe części. Później możesz na przykład posortować polecenia, wykryć ich argumenty i stworzyć program analizujący działanie poleceń AmigaDOS. Oczywiście to tylko przykład, możliwości zastosować jest mnóstwo.

Nie zmienia to faktu, iż musisz w pewien sposób podzielić zawartość zmiennej na wiele mniejszych części. Najprościej zrobić to używając tablic, o których mówimy w oddzielnym punkcie. Gdy je zadeklarujesz musisz wiedzieć, jak „wyciągnąć” fragmenty tekstu. Przydadzą Ci się tutaj funkcje o poniższych nazwach:

- MID$(),
- LEFT$(),
- RIGHT$(),
- REPLACE$()

oraz

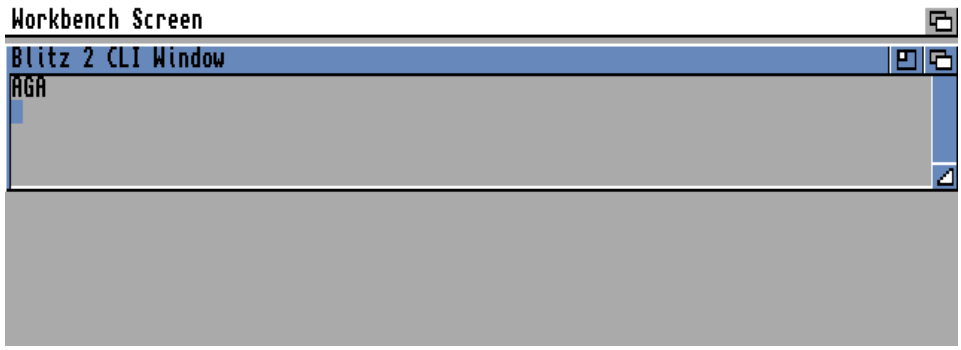
- LEN().

Ostatnią podajemy osobno, bowiem ma inny charakter od pozostałych. Cała czwórka umożliwia wykonywanie operacji wycinania fragmentów ze zmiennych tekstowych, choć w różny sposób. Najbardziej uniwersalna jest opcja wybrania znaków o określonych numerach lub z określonego zakresu. To cecha pierwszej funkcji MID$().

Wystarczy podać nazwę zmiennej, pierwszy znak oraz długość jaką chcemy uzyskać. Na przykład, jeśli wpiszesz taki program:

```
a$="Amiga AGA"
b$=Mid$(a$,7,3)

NPrint b$
```

przekonasz się, że w nowej zmiennej zapisany zostanie tekst jak na następnej ilustracji:



Możesz nie podawać ostatniego parametru, ale wtedy uzyskasz fragment od znaku o określonym numerze aż do końca całego tekstu.

Możesz zrobić to samo, ale rozpoczynając zawsze od początku lub końca zawartości zmiennej. Innymi słowy wybieranie znaków będzie miało miejsce od „lewej" lub „prawej" strony. Dlatego też funkcje pozwalające wywołać te opcje mają nazwy LEFT$ (ang. lewo) lub RIGHT$() (ang. prawo). W tym przypadku należy jednak podać tylko długość tekstu jaki chcesz uzyskać. Może to wyglądać tak:

```
b$=Left$(a$,5)
```

lub tak:

```
b$=Right$(a$,4)
```

W ten sposób możesz podzielić zmienną na wiele mniejszych fragmentów, które będziesz poddawał dalszej obróbce. Same linie z powyższymi funkcjami nie spowodują efektu, który chcemy uzyskać.

Możesz natomiast użyć pętli, która będzie sprawdzać kolejne znaki zmiennej, a następnie instrukcji warunkowej nakazującej podział według określonych długości.

Aby to było możliwe powinieneś nauczyć się jeszcze obliczać długość zmiennej. Nie jest to trudne, ale wymaga wprowadzenia kolejnej funkcji o nazwie LEN(). Umieściliśmy ją na końcu naszej listy, bowiem nie powoduje ona zmian w tekście, lecz jedynie wskazuje na jego długość. Sposób użycia jest następujący:

```
a=Len(a$)
```

W rezultacie otrzymasz liczbę, którą możesz sprawdzać długości zmiennych do określonych znaków i według nich dzielić zawartość. Jeśli niektóre części nie będą pasować do schematu umieszczonego w ramach pętli możesz zamienić automatycznie znaki.

Wystarczy użyć ostatniej funkcji z listy, czyli REPLACE$(). Nasz program może teraz wyglądać tak:

```
a$="Amiga AGA"
b$="AGA"
c$="ECS"

d$=Replace(a$,b$,c$)
NPrint b$
```

W ten sposób napis „Amiga AGA” zostanie zamieniony na „Amiga ECS”. Tak się dzieje, bowiem tym razem parametry należy podać według następującej kolejności:

- ciąg, który chcemy zmienić (tutaj – a$)
- szukany fragment ciągu (tutaj – b$)
- ciąg na jaki ma być zamieniona aktualna treść (tutaj - c$)

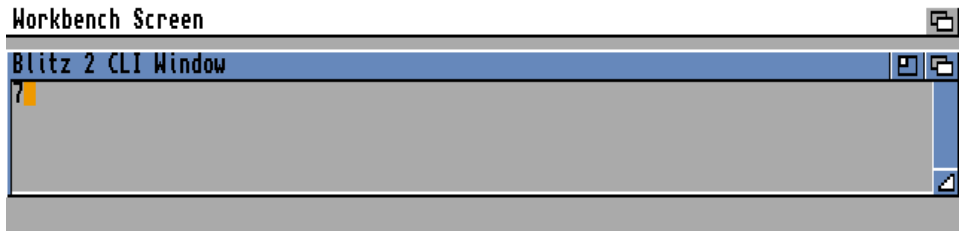
Jeśli jako ostatni parametr wprowadzisz tekst, który nie występuje w zmiennej źródłowej, żadna zmiana nie zostanie wykonana. Zwróć uwagę, że mimo to poszczególnie funkcje w pętli będą nadal wykonywane – przypisywanie zmiennej i wywoływanie funkcji REPLACE$(). Gdy interesuje Cię jedynie zamiana fragmentu ciągu, możesz ograniczyć się do tej funkcji.

Jednak nie możesz sprawdzić, czy tekst został odnaleziony czy też nie. Ponadto program nie będzie działał zbyt szybko Możesz go przyspieszyć sprawdzając, czy zmienna zawiera szukany tekst, a dopiero potem wykonywanie funkcji. W przeciwnym razie linie zastępujące ciąg mogą być pominięte.

Aby uzyskać taki efekt musisz użyć funkcji o nazwie INSTR(). Jej działanie jest trudniejsze do zrozumienia, bowiem w odpowiedzi podaje liczbę, mimo że cały czas operuje na zmiennych tekstowych. Zobacz następny program:

```
a$="Amiga AGA"
b$="AGA"

c=Instr(a$,b$)
NPrint c
```

Gdy go uruchomisz zobaczysz taki rezultat:



Może być to dla Ciebie dziwne, bowiem zobaczysz liczbę. Tak właśnie działa funkcja INSTR() - podaje pozycję pierwszego znaku, który pasuje do szukanego ciągu. W naszym przypadku jest to liczba 7 bo fragment „AGA” rozpoczyna się od siódmego znaku zmiennej „a$”.

Funkcja ta jednak zachowuje się nieco inaczej, gdy nie zostanie odnaleziony szukany fragment. Uzyskasz mianowicie wartość logiczną 0 (zero), co oznacza fałsz, a więc brak interesującego Cię tekstu. O operacjach logicznych możesz przeczytać między innymi w punkcie zatytułowanym „Operatory”.

Nie poruszaliśmy do tej pory problemu wielkości liter. Funkcje REPLACE$() oraz INSTR() mogą brać pod uwagę pisownie lub nie. Domyślnie jest ona sprawdzana, ale nie zawsze będzie to wygodne. Za kontrolę sprawdzania małych i WIELKICH liter odpowiada polecenie CASESENSE. Jeśli chcesz ją wyłączyć, należy zastosowac linię:

```
CaseSense Off
```

Później możliwe jest oczywiście ponowne włączenie kontroli zap pomocą argumentu ON, czyli razem:

```
CaseSense On
```

Pamiętaj, że zmiana następuje w momencie wykonania linii. Kontrola pisowni nie zostanie przywrócona do stanu pierwotnego po użyciu jednej z omawianych funkcji. Zarówno „włączenie”, jak i „wyłączenie” kontroli pisowni musisz ustawiać samodzielnie, biorąc pod uwagę potrzeby oraz – w szczególności – kolejność zastosowanych pętli i tablic.

OPERATORY

Podczas porównywania zawartości różnych zmiennych, wprowadzania instrukcji warunkowych i innych operacji możesz korzystać ze znaków takich jak „plus" czy „minus. Symbole te nazywamy operatorami. Można je podzielić na wiele rodzajów, na przykład:

- operatory arytmetyczne,
- operatory służące do deklaracji zmiennych,,
- operatory porównywania,
- operatory logiczne

i inne. W całej treści książki wielokrotnie spotkasz się przykładami i omówieniem ich działania. Poniżej przedstawiamy listę ważniejszych operatorów, z których możesz korzystać w edytorze.

Pierwsza grupa to zwykłe operacje matematyczne, które nie wymagają komentarza:

`+`	- dodawanie
`-`	- odejmowanie
`*`	- mnożenie
`/`	- dzielenie
`^`	- potęgowanie

Kolejne symbole dotyczą tak zwanych funkcji logicznych, których wynikiem działania jest wartość PRAWDA (ang. True) lub FAŁSZ (ang. False). W pierwszym wypadku oznacza to liczbę -1, natomiast w drugim otrzymasz 0 (zero).

Podstawowe funkcje logiczne mogą być zapisywane za pomocą następujący znaków:

&	- funkcja logiczna AND
|	- funkcja logiczna OR

Funkcja AND przekaże wartość PRAWDA wtedy, gdy dwie zmienne biorące udział w operacji będą posiadały także wartość PRAWDA. Funkcja OR jest inna, bowiem wystarczy, aby jedna ze zmiennych wskazywała na PRAWDĘ, aby cała funkcja przekazała taką właśnie wartość. Więcej na ten temat będzie mówić także w dalszych rozdziałach.

Możesz też korzystać z tak zwanych operatorów porównania, które są stałym elementem instrukcji warunkowych. Większość zapisujemy standardowo, czyli:

=	- oznacza równość zmiennych
<>	- oznacza nierówność zmiennych
<	- oznacza mniejszą wartość zmiennej,
>	- oznacza większą wartość zmiennej

Czasem jednak chcemy użyć jednocześnie znaku „=” oraz „<” lub „>”. W tym wypadku znak równości należy zapisać na końcu, czyli:

<=	- zmienna równa lub mniejsza,
>=	- zmienna równa lub większa.

Nie są to oczywiście wszystkie możliwości, ale używane najczęściej. Przykłady zastosowania powyższych symboli znajdziesz w całej treści książki. Aby lepiej zrozumieć ich działanie przejdź do punktu „Instrukcje warunkowe”.

TABLICE

W swoich programach możesz mieć potrzebne operowania na grupach liczb lub tekstów, które będą w pewien sposób powiązane ze sobą. Takie operacje umożliwiają tak zwane „tablice”.

Wyobraź sobie sytuację, w której chcesz utworzyć zmienne zawierające 10 numerów określających tę samą cechę, na przykład współrzędne poziome na ekranie. W zwykły sposób musiałbyś zastosować 10 różnych zmiennych. Już ich tworzenie nie byłoby wygodne, nie mówiąc o zmianach ich wartości.

Dzięki tablicom wszystko możesz umieścić w ramach jednej zmiennej, która jest uzupełniona o numer indeksu, czyli liczbę w nawiasie. I tak, zamiast zmiennej „a” możesz użyć:

```
a(1)
a(2)
a(3)
a(4)
a(5)
…
a(10)
```

Każda z pozycji jest niezależna i może przyjmować dowolne wartości, ale możesz je dużo łatwiej deklarować oraz zmieniać ich zawartość. Z pewnością w wielu przypadkach będziesz chciał zastosować pętle, o których piszemy w punkcie pod tym samym tytułem.

Definiowanie tablic wygląda jednak nieco inaczej niż deklarowanie zwykłych zmiennych. Na początek musisz określić, ile pozycji indeksu będzie mogła przyjmować tablica. W tym celu używamy słowa DIM podając obok nazwę zmiennej i numer maksymalnego indeksu. Na przykład:

```
Dim a(10)
```

Oznacza to, że zmienna „a" będzie mogła posiadać 11 indeksów, bowiem są one liczone od zera. Przypisywanie im konkretnych wartości działa już analogicznie do tworzenia zwykłych zmiennych. Musisz tylko pamiętać, aby każdy indeks wpisywać w nawiasie po nazwie zmiennej. Zobacz kolejne przykłady:

```
a(1)=5
a(2)=6
a(9)=22
```

Na tablicach możesz wykonywać obliczenia, tak samo jak omawialiśmy to wcześniej. Wprowadzanie wzorów nie zmienia się, jednak linie mogą wyglądać dość egzotycznie. Na przykład, aby dodać do siebie wartość zmiennych z jednej tablicy należy zastosować formułę podobną do następującej:

```
a(5)=a(1)+a(2)
```

lub

```
a(1)=a(1)+1
```

Jak widzisz nie zmienia się ogólna zasada. We wzorach mogą występować różne zmienne, jak również możliwe jest dodawanie określonych wartości do tylko jednej pozycji. Dzięki swojej rozbudowanej formie tablice mogą mieć wiele bardziej skomplikowanych zastosowań, o których powiemy w dalszej części książki.

FUNKCJE

W Blitz Basicu, jak w każdym języku programowania, możesz korzystać z funkcji matematycznych. Mogą one przybierać różne formy i mieć wiele zastosowań. Na początek warto nauczyć się z nich korzystać. Zwykle po nazwie funkcji zapisujemy wartość lub kilka wartości, podobnie jak obok nazwy poleceń. W tym wypadku jednak muszą być one ujęte w nawiasie.

Ogólny schemat zapisu każdej funkcji wygląda tak:

```
FUNKCJA()
```

Jest to uniwersalny sposób używany na różnych komputerach i językach programowania. W związku z tym, jeśli chcesz obliczyć wartość funkcji SINUS, musisz wpisać poniższą linię:

```
Sin(90)
```

Korzystamy z funkcji SIN() (czyli Sinus), a jako argument podajemy liczbę 90. Bezpośredni zapis jest niezbyt wygodny jeśli piszesz większy program, dlatego najlepiej funkcję przypisać do zmiennej. Robimy to tak samo jak wcześniej, przykładowo:

```
a=Sin(90)
```

lub w bardziej rozbudowanej formie:

```
a=Sin(x*t)
```

Zwróć uwagę, że w ramach wzoru funkcji możliwe jest używanie kolejnych zmiennych. Możesz też stosować innego rodzaju funkcje, nie tylko czysto matematyczne. Pozwalają one uzyskać wiele różnych informacji, na przykład za pomocą następującej funkcji:

```
Asc()
```

sprawdzisz wartość liczbową przyporządkowaną do konkretnego znaku. Niektóre funkcje odczytują informacje o Twoim komputerze lub wykonują operacje na zmiennych. Funkcje są szerokim zagadnieniem, dlatego będziemy je wprowadzać stopniowo. Jednak ich cechą wspólną jest zapis oraz konieczność stosowania zmiennych, w których przechowywane są wartości, czyli rezultaty działania funkcji.

POLECENIA I SKŁADNIA

W każdym programie będziesz używał wielu poleceń, czyli słów wywołujących określoną operację. Każde z nich musi być wprowadzone we właściwy sposób. Ten sposób zapisu nazywamy składnią, bowiem wymaga on stosowania właściwej kolejności słów, argumentów i innych elementów niezbędnych do działania. Polecenia mają różną składnię, wymagają także podania różnego rodzaju argumentów, na przykład tekstu lub liczby.

Najprostszym poleceniem jest słowo PRINT, które wyświetla informacje na ekranie. W programie można zrobić to w następujący sposób:

```
Print ”Amiga”
```

lub

```
a$=”Amiga”
Print a$
```

Jak widzisz wymagane jest tutaj podanie tylko jednego argumentu. Może być on wpisany bezpośrednio obok nazwy polecenia, możesz także użyć poznanych wcześniej zmiennych. Oczywiście linie mogą być zdecydowanie bardziej rozbudowane. Spójrz na kilka przykładów:

```
Print "Witaj",a$
Print a$(i)
Print 3,"CARS",a,a*7+3
```

Najbardziej przydatnym poleceniem służącym do wyświetlania tekstu jest jednak nie PRINT, a NPRINT. Różnicę stanowi fakt, że każda informacja znajdzie się w osobnej linii, bowiem program automatycznie doda znak nowej linii. W przypadku słowa PRINT, kolejne wartości będą pojawiać się obok siebie, co może być również przydatne. Warto znać tę różnicę.

W dalszej części książki będziemy mówić o wielu różnych operacjach jakie możesz wywołać. Na razie ważne jest, abyś nauczył się rozpoznawać poszczególne części powyższych linii. Zwróć uwagę, że wszystkie argumenty są rozdzielone znakiem przecinka, natomiast ciągi tekstowe zawsze wpisujemy w cudzysłowie. Argumentem może być zarówno tekst, liczba, tablica, jak również całe wyrażenie.

Niezależnie od funkcji konkretnego polecenia, schemat użycia jest zawsze ściśle określony i należy się do niego stosować. W przeciwnym wypadku Twój program nie będzie działał lub wykona nieprzewidziane operacje.

Zwróć też uwagę, że do tej pory wszystkie polecenia wpisywaliśmy do oddzielnych linii. Możesz je łączyć poprzez wykorzystanie znaku dwukropka, czyili zamiast pisać:

```
a=1
a=a+1

NPrint a
```

napisz po prostu:

```
a=1:a=a+1:NPrint a
```

Taki sposób nie jest jednak zalecany, bowiem utrudnia analizę listingu. Ponadto nie pozwala na stosowanie odstępów lub pustych linii, które mogą być przydatne podczas rozbudowy programu. Mimo wszystko, efekt działania będzie identyczny, tak więc wiele zależy od Twojego przyzwyczajenia.

INSTRUKCJE WARUNKOWE

W programach będziesz musiał uzależniać działanie od poszczególnych elementów, na przykład stanu zmiennych lub wyboru dokonanego przez użytkownika. Jest to możliwe przy zastosowaniu tak zwanych instrukcji warunkowych.

Obok nich należy podać „warunek", po którego spełnieniu wykonane zostaną określone operacje. W przeciwnym razie zostaną pominięte lub uruchomiona zostanie inna operacja.

IF … THEN

Warunek może być wprowadzony w jednej linii. Służy do tego konstrukcja typu IF … THEN. Po pierwszym słowie wprowadzamy warunek, natomiast po drugim musi znaleźć się polecenie do wykonania. Na przykład:

```
If b=30 Then Print a
```

Powyższa linia spowoduje wypisanie zawartości zmiennej „a" za pomocą słowa PRINT pod warunkiem, że zmienna „b" osiągnie wartość 30. Takich linii możesz umieścić więcej, jedna pod drugą lub w określonych częściach programu, aby uzyskać różne działanie w sytuacji zmiany wartości zmiennych.

IF … ENDIF

W wielu przypadkach będziesz potrzebował wywołać bardziej skomplikowane działania, które nie będą mogły być zapisane w jednej linii. Dlatego warunek można zapisać w formie wielu linii.

Aby zmienić w ten sposób nasz poprzedni przykład należy użyć słów IF oraz ENDIF. Zobacz jak może to wyglądać:

```
If b=30
     Print a
EndIf
```

Na razie działanie będzie identyczne. Jaki mamy z tego zysk? Pomiędzy linią rozpoczynająca i kończącą warunek możemy umieścić dowolne inne polecenia, w wielu liniach. Dzięki temu, po spełnieniu warunku, program może wykonać bardziej skomplikowaną operację.

Ten zapis daje dodatkowy zysk w postaci możliwości zapisywania kolejnych warunków w ramach jednej konstrukcji IF … ENDIF. Schematycznie możemy to zrobić tak:

```
If WARUNEK
     POLECENIE1
     POLECENIE2
     POLECENIE3
          While WARUNEK
               If WARUNEK
                    PĘTLA
```

```
Endif
            Wend
Endif
```

Jest to oczywiście tylko przykład, ale pokazuje jak bardzo rozbudowany może być warunek, który na początku wprowadzony był jako jedna prosta linia.

Zapisywanie kolejnych instrukcji warunkowych w ramach poprzedniej nosi nazwę „zagnieżdżania" i wbrew pozorom nie jest wcale rzadko stosowane. Gdy będziesz stopniowo rozszerzał swój program, z pewnością stanie się bardziej złożony nie tylko pod względem ilości funkcji. Może okazać się, że działanie uzależnisz od coraz bardziej precyzyjnych wartości różnych zmiennych. Jeśli jedna z nich będzie miała osiągać liczby w określonych zakresach, możesz utworzyć wiele kolejnych warunków lub użyć wpisu podobnego jak w naszym przykładzie.

Często jest to łatwiejsze do analizy, poza tym program będzie krótszy niż wprowadzanie nowych konstrukcji IF … ENDIF i powtarzanie tylko niektórych poleceń. Pamiętaj, że nie powinieneś komplikować zapisu bez potrzeby.

IF … THEN … ELSE

W sytuacji braku spełnienia warunku zapisanego w linii IF, program przejdzie do wykonywania kolejnych linii lub zostanie zakończony. Możesz jednak sprawić, aby wywołana została

automatycznie inna instrukcja. Do tego celu służy słowo ELSE, które trzeba dodać na końcu linii. Oto przykład:

```
If a=3 Then Print "Amiga" Else Print "Atari"
```

Wszystko nadal zapisujemy w jednej linii, ale teraz po spełnieniu warunku zostanie wyświetlony inny napis niż w sytuacji, gdy warunek nie zostanie spełniony. Trzeba przyznać, że zapis ten jest krótki, ale mało wygodny i nie daje wielu możliwości dalszego działania. Dlatego to samo możesz wprowadzić w inny sposób, wykorzystując dodatkowo słowo ENDIF:

```
If a=3
     Print "Amiga"
Else
     Print "Atari"
EndIf
```

Działanie będzie identyczne, ale teraz – podobnie jak wcześniej – możesz wpisać większą ilość linii i wywołać bardziej rozbudowane funkcje. Pamiętaj, że w tym wypadku rozróżniane będą dwa człony:

1) od słowa IF do ELSE,
2) od ELSE do końcowego ENDIF.

Wszystkie linie umieszczone w tych „klamrach” zostaną potraktowane jako przeznaczone do wykonania po spełnieniu warunku (człon 1) oraz następnie - przy braku spełnienia (człon 2). Analiza takiego programu jest trudniejsza, ale daje nowe możliwości, bowiem w ramach

każdej z wymienionych części możesz umieszczać kolejne pętle, warunki i inne potrzebne elementy.

Można więc powiedzieć, że słowo ELSE jest uzupełnieniem i wywołuje kolejny warunek, przeciwny do pierwszego. Podobny efekt możesz uzyskać wpisując po prostu dwa warunki używając konstrukcji typu IF … ENDIF. Jest to bardziej zawiłe szczególnie, gdy Twoje instrukcje warunkowe będą rozbudowane i zapiszesz ich więcej. Dlatego sprawdź działanie instrukcji warunkowych na kilku przykładach, zanim zdecydujesz się za użycie bardziej złożonych struktur. Powiązania pomiędzy wartościami zmiennych mogą nastręczać trudności, nawet jeśli Twój program nie jest na początku zbyt długi.

Pamiętaj także o tym, że w niektórych wypadkach będziesz musiał zerować wartości, w przeciwnym razie warunki mogą być spełnione zawsze lub nigdy. Zmienne nie zmieniają stanu automatycznie po wykonaniu operacji zapisanej w ramach instrukcji warunkowej. Musisz o to zadbać samodzielnie, chyba że przypisane do nich liczby i teksty będą potrzebne w dalszym ciągu działania programu.

PĘTLE

Jeśli chcesz wykonać wiele razy tę samą operację, możesz użyć tak zwanych „pętli". Pozwalają one skrócić program, a wielu przypadkach także przyspieszyć jego działanie. Możesz powtarzać te same fragmenty programu, co nie oznacza, że za każdym obiegiem pętli rezultat działania będzie taki sam.

Istnieje kilka rodzajów pętli, ponadto w ich w ramach możesz zmieniać wartości zmiennych, a więc modyfikować parametry w trakcie wykonywania operacji. Pamiętaj jednak, że każdy obieg pętli zachowuje tę samą zasadę działania.

FOR … TO … NEXT

Najprostszy rodzaj pętli powoduje wykonanie poleceń umieszczonych pomiędzy kilkoma słowami, bez ustalania dodatkowych warunków. Pętla wymaga jedynie wprowadzenia zakresu działania oraz zmiennej, która będzie pełnić rolę licznika. Na przykład, za pomocą poniższej linii możemy sprawić, że wykonanych zostanie 100 obiegów pętli:

```
For i=0 To 100
```

Jak widać liczby będą przyjmować kolejne liczby całkowite od 0 do 100, a zmienna sterująca ma symbol „i". Taka linia nie będzie jednak działać samodzielnie, bowiem trzeba jeszcze zapisać jej koniec, a to robimy za pomocą słowa NEXT. Całość może mieć więc formę:

```
For i=0 To 100
      ...
Next
```

W miejscu wielokropka należy oczywiście umieścić fragment programu, który będzie powtarzany. Obieg pętli 100 razy możesz osiągnąć za pomocą wielu podobnych wyrażeń, ale mogą być one zapisane w inny sposób. Spójrz na dwa kolejne przykłady:

```
For abc=0 To 100
      ...
Next
```

lub

```
For p=514 To 614
      ...
Next
```

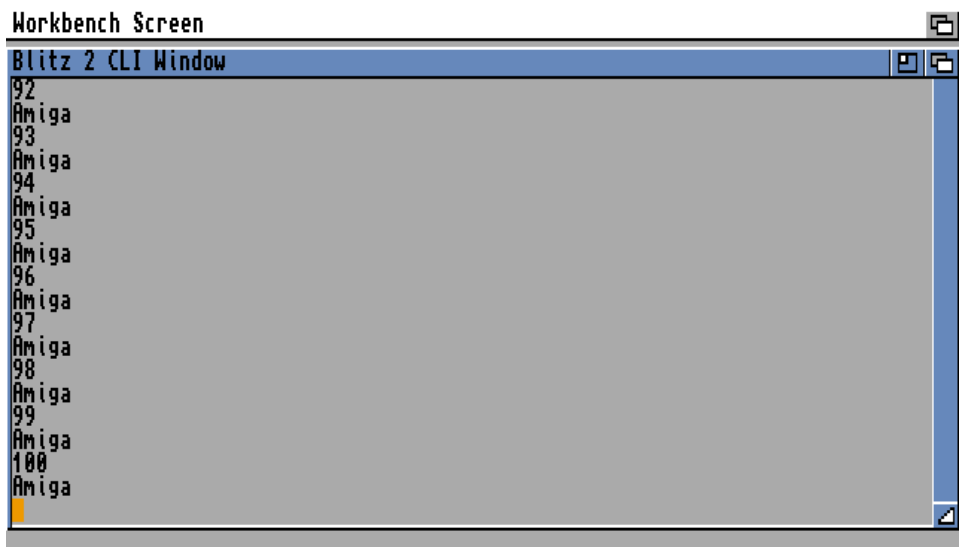
W pierwszym przypadku zmieniliśmy tylko nazwę zmiennej, więc jest to łatwe do zrozumienia. Drugi przykład pokazuje, że obieg pętli może wyglądać tak samo, mimo przyjmowania innych wartości zmiennych. Zamiast używać liczb od 1 do 100 licznik może mieć dowolne inne wartości.

Zwróć jednak uwagę, że zmienna stanowiąca licznik może być wykorzystywana analogicznie do innych zmiennych. W związku z tym powtarzany fragment wewnątrz pętli może korzystać z wartości „i", „abc" lub „p", choć oczywiście nie musi. Wartości przypisywanie tym

zmiennym nie mają znaczenia do czasu, w którym powiążesz je z poleceniami umieszczonymi w ramach pętli. Aby to lepiej zrozumieć prześledzimy działanie następującego programu:

```
For abc=0 To 100
      NPrint abc
      NPrint „Amiga”
Next
```

W ramach pętli umieściliśmy polecenie NPRINT, które powoduje wyświetlenie komunikatu – zawsze w nowej linii. Więcej na ten temat piszemy w punkcie zatytułowanym „Polecenia i składnia”. Powyższy zapis sprawi, że pojawią się kolejne wartości zmiennej „abc”. Druga linia to polecenie nakazujące wyświetlenie wyrazu „Amiga”, co zostanie wykonane za każdym razem po wypisaniu wartości zmiennej. W rezultacie uzyskasz efekt podobny do takiego:



Oczywiście ostatnią wartością będzie 100. Wyobraź sobie teraz, że chcesz wyłącznie wyświetlić 100 razy napis „Amiga”. W tym wypadku nie będzie miało znaczenia jakie wartości przyjmie zmienna „abc”. Jeśli jednak jest dla Ciebie ważne, aby w ramach formuły umieścić jej symbol musisz pamiętać, że wynik działania pętli będzie bezpośrednio związany z liczbami, jakie wygeneruje pętla.

Kolejna ważna kwestia to sposób powiększania się wartości zmiennej. Licznik nie musi przyjmować kolejnych wartości, ale możesz kontrolować jego wzrost. Służy do tego słowo STEP (ang. krok), które trzeba umieścić na końcu linii z poleceniem FOR. Obok zapisujemy wartość, o którą ma zwiększać się licznik, zamiast jedności. Jeśli zmodyfikujemy poprzedni przykład, może przyjąć poniższą formę:

```
For abc=1 To 100 Step 2
     NPrint abc
     NPrint „Amiga”
Next
```

Teraz licznik będzie zmieniał się od 1 do 100, ale pojedynczy krok wyniesie 2. Efektem będzie tylko 50 obiegów pętli, bowiem schemat będzie wyglądał tak:

```
1
Amiga
3
Amiga
5
Amiga
…
```

Odliczanie możesz także wykonywać wstecz, według tych samych wskazówek. Podobnie, należy wstawić słowo STEP, ale tym razem z ujemną liczbą, na przykład:

```
For abc=100 To 1 Step -2
```

Gdy pomylisz wartości lub ich znaki i obieg pętli nie będzie możliwy do wykonania, nie zostanie wywołany żaden błąd. Może to mieć miejsce wpisując „krok" jako liczbę dodatnią:

```
For abc=100 To 1 Step 2
```

Należy o tym pamiętać, bowiem można być zdziwionym, że program nie działa. W tej sytuacji musisz prześledzić wartości wprowadzonych zmiennych.

Pamiętaj, że ilość obiegów pętli jest bezpośrednio uzależniona od liczb, które zostały zapisane z linii FOR. Pozostałe wartości w ramach pętli będą się zmieniać wyłącznie wtedy, gdy wykonany zostanie choć jeden raz „odliczanie".

Pętla typu FOR … NEXT nie zawsze będzie przydatna, bowiem wykonuje operacje bez względu na zawartość linii umieszczonych pomiędzy linią początkową, a końcową. Ponadto zawsze wprowadza nową zmienną, co powiększa zapotrzebowanie programu na pamięć. Gdy utworzysz jedną pętlę nie będzie miało to większego znaczenia. Jeśli jednak Twój program stanie się rozbudowany, może się okazać, że każdy kilobajt będzie na wagę złota.

Ponadto w wielu przypadkach będziesz potrzebował uzależnić działanie obiegu pętli od zawartości różnych zmiennych. Można to osiągnąć za pomocą tak zwanych instrukcji warunkowych, o których piszemy w oddzielnym punkcie. Alternatywnie możliwe jest użycie innych rodzajów pętli, które nie wymagają wpisywania oddzielnych linii określających ramy działania.

WHILE … WEND

Jeżeli obieg Twojej pętli ma być wykonywany do czasu, gdy przestanie być spełniony konkretny warunek, użyj konstrukcji WHILE … WEND. Taką pętlę wpisujemy analogicznie do poprzedniej, ale obok pierwszego słowa należy podać warunek. Na przykład:

```
While a<3
     NPrint „Mniej niż 3”
Wend
```

Jak łatwo się domyślić, napis umieszczony obok polecenia NPRINT będzie wyświetlany cały czas, chyba że zmienna „a” uzyska wartość większą niż 3. Na własny użytek może to opisać w następujący sposób:

podczas gdy warunek a<3 JEST SPEŁNIONY - wykonuj pętlę

Jednak nasz program na razie nie zadziała, bo zmienna ma wartość zerową. Musimy teraz dodać linie deklarujące początkową wartość oraz schemat ich zmian. Może to wyglądać tak:

```
a=1

While a<3
     NPrint „Mniej niż 3”
     a=a+1
Wend
```

Teraz zmienna „a” uzyska wartość 1, a następnie – przy każdym obiegu pętli – zmieni się ona również o 1. Przy czwartym obiegu zmienna osiągnie wartość 4 i pętla zostanie zakończona.

W ten sam sposób możesz zapisać bardziej skomplikowane warunki, chyba że nie da się ich umieścić w jednej linii, obok słowa WHILE. Na przykład możesz powiązać działanie z wartościami dwóch zmiennych,. Wymaga to jednak wprowadzenia operatora w postaci słowa AND lub OR. Na czym polega różnica? Spójrz na kolejny przykład:

```
a=1
b=20
While a<3 AND b>10
      NPrint a
      a=a+1

      NPrint b
      b=b-2
Wend
```

Teraz będą wyświetlane wartości obu zmiennych, ale pierwsza będzie się zwiększać, druga – zmniejszać. Warunek ze słowem AND oznacza, że brane pod uwagę będą obie zmienne, choć każda w inny sposób. Innymi słowy uwzględnione zostaną jednocześnie oba warunki, cały czas zapisane w jednej linii. Nasz warunek można zapisać słownie tak:

podczas gdy zmienna „a” jest mniejsza od 3 ORAZ zmienna „b” jest większa od 10 - wykonuj pętlę

Wartości znowu będą się zmieniać, a obieg zostanie przerwany tylko wtedy, gdy wartości obu zmiennych będą pasować do podanego warunku. Ten sam program możesz zapisać tak, że pierwsza linia będzie zawierać słowo OR, czyli:

```
While a<3 OR b>10
      …
Wend
```

Taka pętla będzie wykonywana do czasu, gdy spełniony zostanie jeden lub drugi warunek, a nie oba w tym samy czasie. Spróbujmy po raz kolejny zapisać to słownie:

podczas gdy zmienna „a" jest mniejsza od 3 LUB zmienna „b" jest większa od 10 - wykonuj pętlę

Zwróć uwagę, że zmienia to całkowicie sposób działania pętli. Do jej działania nie potrzebujesz już zmiennej „a" i zmiennej „b", bowiem wystarczy tylko jedna z nich. Dzięki temu wartości drugiej mogą być modyfikowane w dowolny sposób i nie przerwie to wykonywania operacji umieszczonych w liniach pomiędzy WHILE i WEND.

Analogicznie można budować kolejne warunki, w których będzie występować zarówno słowo AND, jak i OR. Zmienne nie muszą być liczbowe, choć przy ich użyciu najłatwiej można zrozumieć działanie warunków. Na przykład:

```
While a<3 AND b>10 OR c$="Amiga"
```

Wprowadzamy kolejną zmienną, tym razem zawierającą ciąg tekstowy. Pętla wymaga obecności dwóch zmiennych lub tylko jednej, tak więc Twój program może zmieniać działanie w zależności od potrzeb, wszystko zależy od jego przeznaczenia.

Skomplikowane warunki przydają się wtedy, gdy chcesz uzyskać jak najkrótszy program. Nie musisz pisać kolejnych, podobnych fragmentów, lecz jedynie w umiejętny sposób ustawiać wartości zmiennych. Utrzymanie porządku jest ważne tym bardziej, im dłuższy jest program. Łatwo bowiem popełnić tak zwany błąd logiczny, czyli wprowadzić program, który co prawda działa, ale nie wykonuje operacji tak, jak wyglądało to w założeniach. Więcej na ten temat przeczytasz w punkcie pod tytułem „Obsługa błędów”.

REPEAT … UNTIL

Kolejna zmienna jest podobna do poprzedniej, ale warunek należy wpisać w ostatniej linii. Zmodyfikujmy nasz poprzedni przykład:

```
Repeat
      …
Until  a=3
```

Różnica nie polega tylko na użyciu innych słów. W tym wypadku warunek będzie sprawdzany po każdym obiegu pętli, a nie tak jak wcześniej – przed jej wykonaniem. Dzięki temu pętla typu REPEAT … UNTIL jest zawsze wykonywana co najmniej raz.

Dla ułatwienia opiszmy słownie działanie:

dopóki zmienna „a" nie osiągnie wartości 3 - wykonuj pętlę

W ten sposób podobnie działające fragmenty programu możesz zapisać w różny sposób, często używając prostszego lub krótszego zapisu. Pozostałe możliwości określania warunków nie zmieniają się. Pamiętaj jednak, że te same warunki będą działać inaczej, w zależności od zastosowania określonego typu pętli.

ETYKIETY

Kolejne części programu możesz oznaczać przy pomocy tak zwanych etykiet. W tym celu wystarczy podać nazwę, która będzie przypisane do określonego fragmentu programu i zakończyć ją znakiem dwukropka. Całość zapisujemy w oddzielnej linii, na przykład tak jak poniżej:

```
punkt1:
```

Może to być przydatne podczas analizy działania programu, ale nie tylko. Blitz Basic będzie traktował to miejsce jako „zakładkę", do której możesz później przejść korzystając z polecenia GOTO. Po prostu wpisz linię:

```
Goto punkt
```

aby wykonany został „skok". Zwróć uwagę, że tym razem nazwy nie kończymy dwukropkiem. Nie musisz tu także podawać żadnych dodatkowych argumentów. Program może mieć wiele różnych etykiet, a poszczególne fragmenty nie są w żaden sposób zakończone. Możemy to zapisać w formie kolejnego schematu:

```
Etykieta1:
      POLECENIE1
      POLECENIE2
      POLECENIE3
Etykieta2:
      POLECENIE1
      POLECENIE2
```

```
Goto Etykieta1
Etykieta3:
```

Jak widać, polecenie GOTO nie musi kończyć programu, masz pełna swobodę działania. Z jednej strony jest to zaleta, bo pracujemy szybciej, z drugiej jednak, trzeba pamiętać o specyficznym zachowaniu programu. O ile bowiem „skoczysz" do określonej części, wartości zmiennych oraz stan innych wykorzystanych elementów nie powrócą do pierwotnej formy. Jeśli na przykład wartości zmiennych zostaną powiększone, zapisane instrukcje warunkowe mogą zmienić działanie. Linia ze słowem GOTO to po prostu bardzo prosta pętla, do której trzeba dostosować działanie pozostałych części programu.

PODPROGRAMY

Do etykiet podobne są tak zwane podprogramy. One również wymagają oznaczenia określoną nazwą, ale muszą być zakończone słowem RETURN. Spójrz na kolejny przykład:

```
punkt2:
      POLECENIE1
      POLECENIE2
      POLECENIE3
Return
```

Wygląda to bardzo podobnie, oprócz ostatniej linii. Aby wywołać działanie podprogramu należy użyć polecenia GOSUB, podobnie jak wcześniej. W naszym wypadku będzie to wyglądało tak:

```
Gosub punkt2
```

Do tego miejsca nie ma różnic, bowiem wykonane zostaną linie pomiędzy nazwą, a słowem RETURN. Jednak później nie będą wywołane kolejne linie w listingu, a program powróci do miejsca, którego wykonywany był „skok".

Wszystko powoduje, że podprogramy powinny być umieszczone w jednym miejscu programu (zwykle na końcu), który jest niezależny od pozostałej części. Można to zobrazować następującym schematem:

```
Etykieta1:
      POLECENIE1
      POLECENIE2
```

```
      POLECENIE3
Etykieta2:
      POLECENIE1
      Gosub punkt2
      POLECENIE2
End

punkt2:
      POLECENIE 1
      POLECENIE 2
      POLECENIE 3
Return
```

Teraz pow wykonaniu POLECENIA1 należącego do „Etykiety2", program uruchomi instrukcje należące do podprogramu „punkt2", a następnie wywołane zostanie POLECENIE2, wciąż zapisane pod „Etykietą2".

Zwróć uwagę, że wprowadziliśmy tutaj jeszcze dwie zmiany. Na końcu programu, ale przed treścią podprogramu, została dodana linia END, która kończy działanie programu. Gdyby jej nie było, podprogram „punkt2" zostałby uruchomiony, a następnie wywołany byłby błąd. Tak się dzieje, bowiem polecenie RETURN musi „wiedzieć", do którego miejsca programu ma powrócić. Wywołanie podprogramu bez linii ze słowem GOSUB nie jest prawidłowe i nie powinno mieć miejsca. Stąd program musi zakończyć wcześniej swoje działanie.

Podprogram oddzieliliśmy dodatkową pustą linią od reszty listingu. Nie jest to konieczne, ale powoduje, że program jest bardziej

czytelny. Ponadto możesz stosować linie tak zwanych komentarzy, które będą pomijane w trakcie działania programu. Na przykład:

```
; Tutaj zaczynam
;
POLECENIE1
POLECENIE2

; Obliczenia
POLECENIE3
```

Wystarczy użyć znaku średnika, a po nim wpisać dowolny tekst komentarza. Możesz również wprowadzić sam średnik, nie ma to znaczenia. Ten sposób warto stosować w przypadku rozbudowanych programów, bowiem po pewnym czasie możesz nie pamiętać jak działają różne fragmenty. Ponadto całość jest łatwiejsza do analizy i przeszukiwania, a więc oszczędzamy czas podczas testowania i modyfikacji listingu. Trzeba tylko pamiętać, że program zajmuje wtedy nieco więcej miejsca w pamięci.

PROCEDURY

Podprogramy są wygodne, o ile nie stosujemy dużej ilości różnych zmiennych, które mają wielokrotnie wpływać na działanie programu. Musisz pamiętać, aby nazwy zmiennych się nie powtarzały, ponadto w wielu przypadkach będziesz musiał wprowadzać kolejne, podobne zmienne, aby poszczególne części programu były od siebie naprawdę niezależne.

Tych wszystkich problemów pozbawione są tak zwane „procedury", które możemy zastosować w zamian. Można powiedzieć, że jest to sposób na umieszczanie konkretnych funkcji programu w samodzielnych modułach. Mogą być one wywoływane z głównej części programu, a zmienne są przekazywane niezależnie od pozostałych wartości.

Należy podkreślić, że treść procedury musi znaleźć się przed linią, która będzie ją wywoływać. W przeciwnym razie program nie będzie możliwy do uruchomienia. Dlatego najlepiej wszystkie procedury umieścić w jednym ciągu, na początku programu – na przykład obok deklaracji zmiennych.

Procedury posiadają własny „obszar roboczy", dzięki czemu możemy mieć pewność, że reszta zmiennych nie wpłynie na działanie utworzonych funkcji. Używanie procedur jest jednak trudniejsze. Aby utworzyć nową musisz wprowadzić dwie linie ze słowami STATEMENT oraz END STATEMENT. Cała konstrukcja wygląda tak:

```
Statement moja1{n}
      …
      …
      …
End Statement
```

Tym razem musimy wpisać nawiasy klamrowe, w ramach których podajemy parametry procedury. Jest to konieczne, nawet jeśli nie wymaga ona podawania żadnych wartości.

Parametry to „argumenty", które wprowadzamy do procedury jako zmienne początkowe, czyli wejściowe. Możesz użyć maksymalnie sześciu zmiennych do przekazywania parametrów do procedury. Jeśli potrzebujesz więcej, dodatkowe parametry można umieścić w specjalnych zmiennych globalnych, które omawiamy w punkcie pod tytułem „Zmienne lokalne i globalne".

Oczywiście pomiędzy liniami STATEMENT i END STATEMENT umieszczamy polecenia do wykonania, podobnie jak wcześniej przy podprogramach. Aby później wywołać utworzoną procedurę, należy użyć linii z nazwą procedury w następujący sposób:

```
moja1{5}
```

Znowu korzystamy z nawiasów klamrowych, ale wewnątrz podajemy już konkretną wartość, która będzie używana jako zmienna w procedurze. Przykładowo, aby dodać określoną wartość do zmiennej i wyświetlić ją na ekranie możesz zastosować poniższą procedurę wraz z jej wywołaniem:

```
Statement moja1{n}
      a=1
      a=a+n

      NPrint a
End Statement

moja1{5}
```

Dzięki ostatniej linii zmienna „n” uzyska wartość 5, która zostanie dodana do początkowej wartości zmiennej „a”. Wynik obliczenia zostanie wypisany za pomocą polecenia NPRINT.

Jeśli wywołasz procedurę podając inną liczbę jako parametr, zostanie obliczona inna suma, według wzoru wewnątrz procedury. Może być ona dużo bardziej rozbudowana, dzięki czemu program będzie działał różnie po wpisaniu linii z jednym słowem FACT.

Należy pamiętać także o tym, że każda zmienna wewnątrz procedury jest deklarowana przy każdym wywołaniu, dlatego program może działać nieco wolniej niż przy użyciu podprogramów.

Procedury są niezależne od pozostałej części programu. Zmienne zachowują swoje wartości tylko we własnych „obszarach roboczych”. Dzięki temu każda procedura może być bez zmian przeniesiona do innego programu.

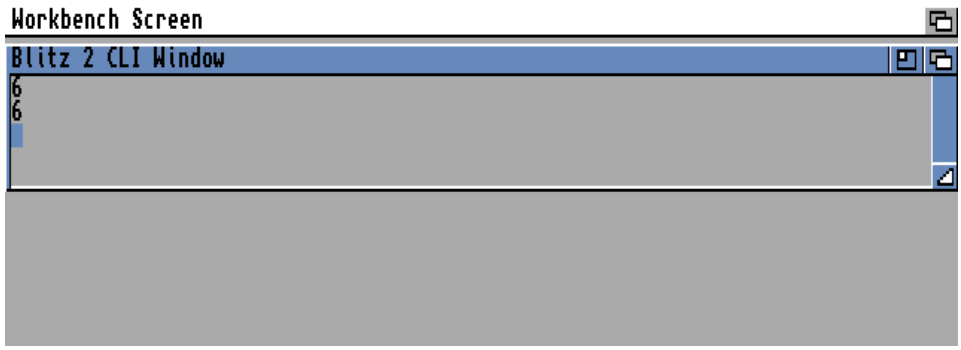
- Zmienne lokalne i globalne

W razie potrzeby możesz jednak zmienić zachowanie się zmiennych tak, aby ich zawartość mogła być odczytywana z poziomu głównego programu, czyli poza procedurą. W tym celu należy zastosować polecenie SHARED wewnątrz konstrukcji STATEMENT … END STATEMENT.

Odnosząc to do poprzedniego przykładu może to wyglądać następująco:

```
Statement moja1{n}
      Shared a
      a=1
      a=a+n

      NPrint a
End Statement

moja1{5}
NPrint a

End
```

Obok słowa SHARED podajemy po prostu nazwę zmiennej. Oto rezultat działania powyższego programu:



Zwróć uwagę, że pierwszy wynik jest wyświetlany z poziomu procedury, a drugi - głównego programu. Jeżeli skasujesz linię:

```
Shared a
```

drugi wynik działania linii NPRINT będzie „zerowy”, bowiem zmienna zostanie potraktowana jako „lokalna”, czyli możliwa do odczytania wyłącznie w ramach procedury. Na tym właśnie polega różnica pomiędzy tymi dwoma rodzajami zmiennych.

ROZSZERZAMY PROGRAM

INFORMACJA OD UŻYTKOWNIKA

Wcześniej mówiliśmy o wyświetlaniu informacji, natomiast w wielu przypadkach będziesz chciał uzależnić działanie programu od informacji wpisanych przez użytkownika. Jest to możliwe przy użyciu funkcji EDIT() lub EDIT$().

Pierwszą z nich zastosuj, gdy chcesz uzyskać dane liczbowe, drugą – gdy wprowadzane informacje mają być tekstem. O funkcji EDIT$() piszemy również w punkcie pod tytułem „Obsługa plików".

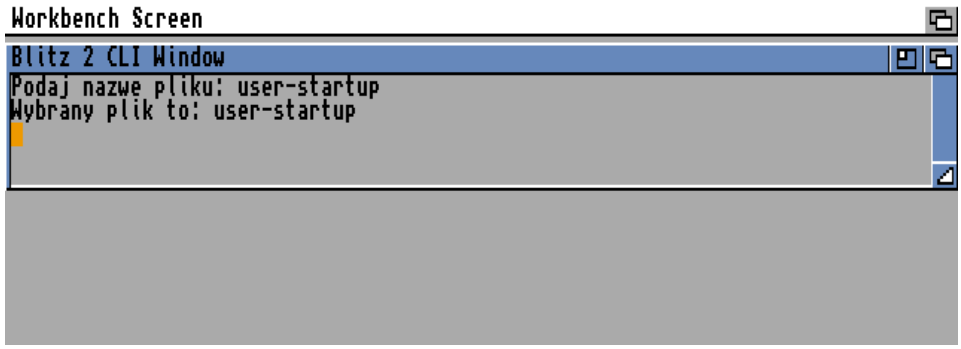
Zobacz przykład:

```
Print "Podaj nazwę pliku: "
a$=Edit$(30)
NPrint "Wybrany plik to: ",a$
MouseWait
```

Ten krótki program prezentuje sposób użycia wspomnianej funkcji. Należy ją przyporządkować do zmiennej, zgodnie z rodzajem wpisywanych informacji. W naszym wypadku jest to tekst, który następnie wyświetlane za pomocą zwykłego polecenia NPRINT. Na koniec program oczekuje za naciśnięcie lewego klawisza myszki i kończy swoje działanie.

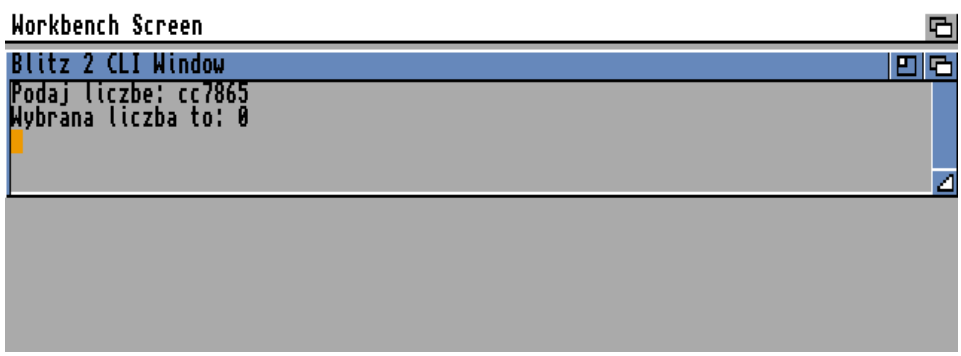
Jako argument obu funkcji trzeba podać liczbę oznaczającą maksymalna długość jaką będzie przyjmowała zmienna – u nas „a$". Pamiętaj, że nie oznacza to braku możliwości wprowadzenia większej

ilości znaków, ale wskazana ilość będzie zapamiętywana w ramach zmiennej. Zobacz rezultat działania powyższego programu:



Zwróć uwagę, że jeśli korzystasz z funkcji EDIT$() dla zapamiętania ciągu tekstowego, bez problemu umieścisz w nim również liczbę, choć będziesz musiał ją później zamienić na zmienną liczbową. Jak to zrobić piszemy w punkcie pod tytułem.

Natomiast użycie funkcji EDIT(), aby zapisać zmienną liczbową powoduje, że po wprowadzenia znaku innego niż cyfry program może przyjąć nieprawidłową wartość lub 0 (zero). Przykładowo:



Gdy chcesz użyć ułamka dziesiętnego jako separator wprowadź znak kropki, a nie przecinka, czyli przykładowo

```
22.537
```

Wiele innych dialektów języka Basic podobną operację realizuje za pomocą polecenia (na przykład INPUT), któremu podajemy zmienną bez używania funkcji. W Blitz Basicu konstrukcja programu musi być inna, ale działanie jest bardzo podobne.

EKRANY

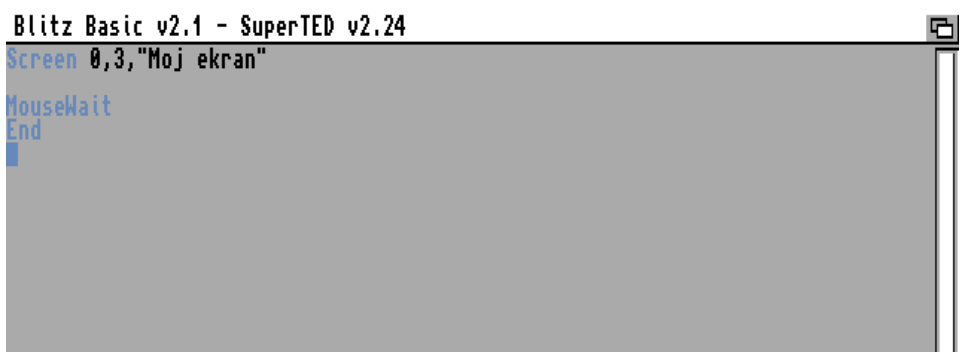
Blitz Basic umożliwia bezpośrednią obsługę elementów interfejsu graficznego systemu operacyjnego. Możesz więc otworzyć nowy ekran, tak samo jak robi to prawie każdy większy program dla Amigi. Masz do dyspozycji szereg poleceń, które w większości posiadają w nazwie słowo „Screen", co oznacza właśnie „ekran".

- Otwieranie ekranu

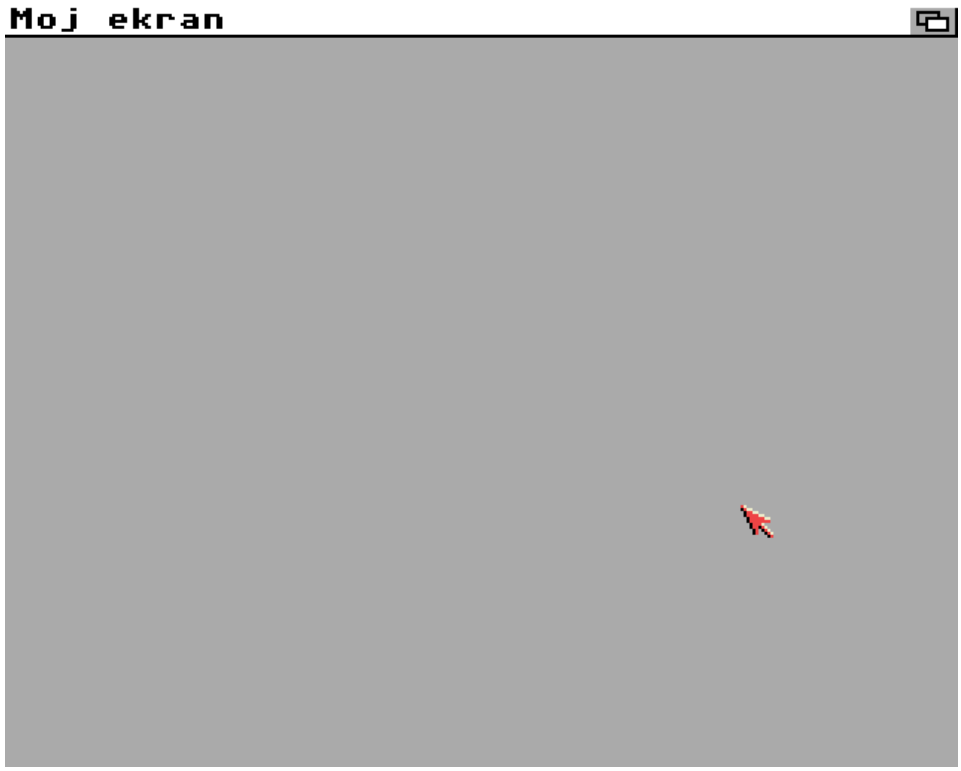
Jeśli chcesz otworzyć oddzielny ekran dla swojego programu, skorzystaj z krótkiego polecenia SCREEN. Wystarczy podać tylko trzy podstawowe parametry:

- numer ekranu,
- tryb wyświetlania,
- tytuł ekranu.

Może to wyglądać na przykład tak:



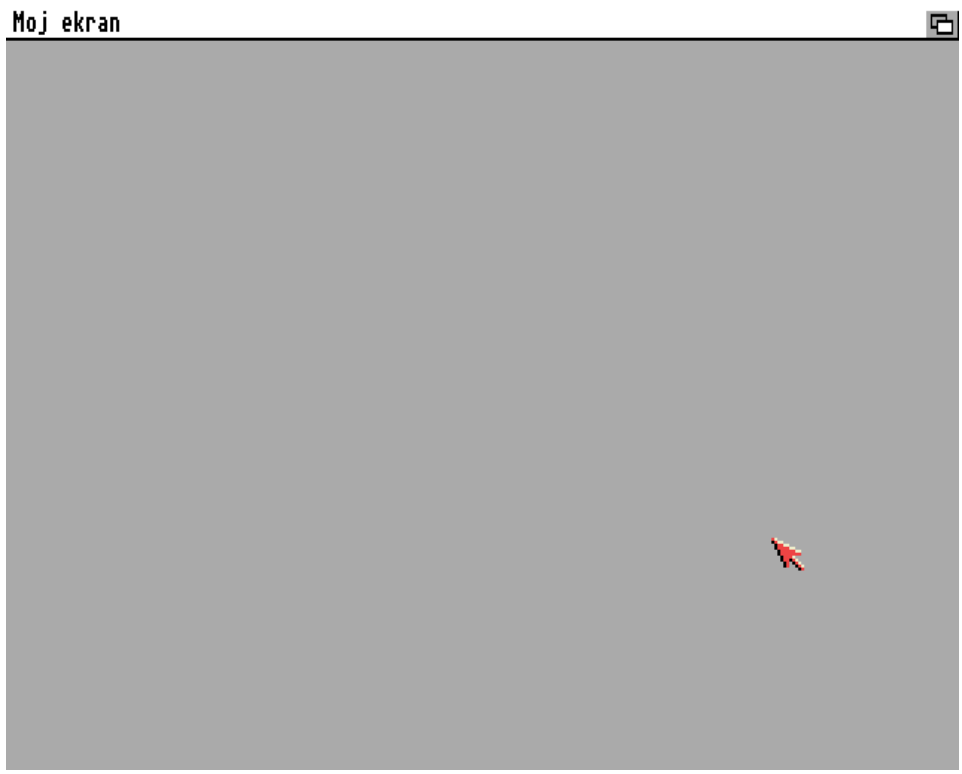
A oto rezultat działania tego prostego programu:



Ostatnia linia to oczywiście oczekiwanie na naciśnięcie lewego klawisza myszki. Piszemy o tym także w punkcie pod tytułem.

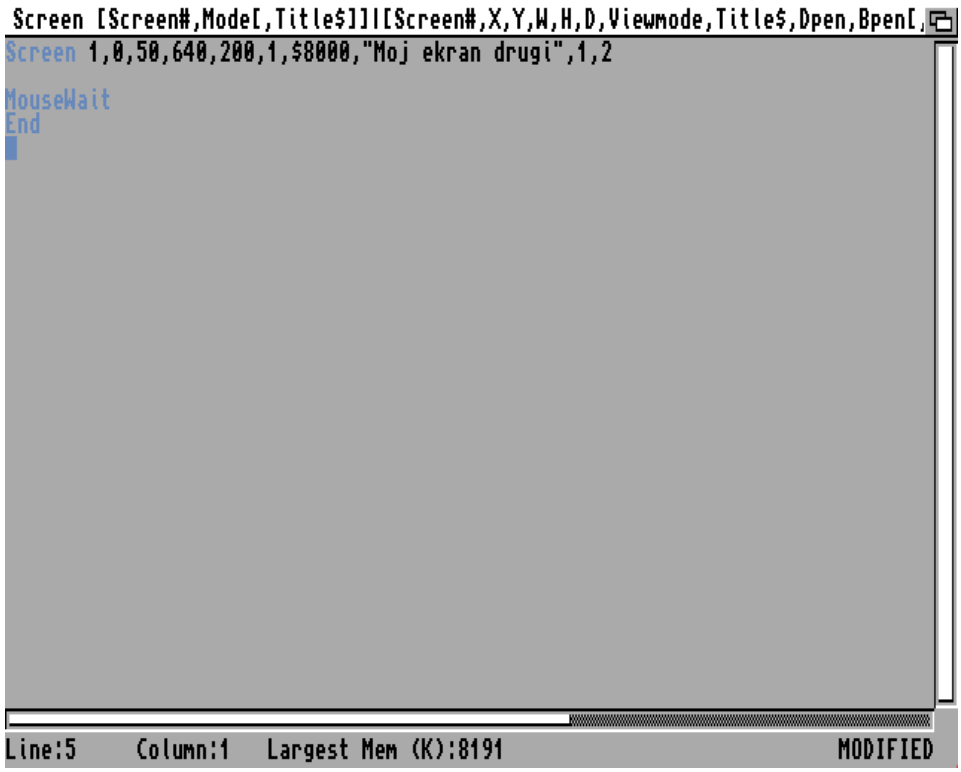
Parametr określający tryb wyświetlania jest drugi i wpisujemy go w formie liczby. Określa on tak zwaną „głębokość" ekranu, czyli ilość bitplanów jakie będą dostępne. Oczywiście przekłada się to na dostępne kolory, na przykład 4 bitplany to 16 barw. Aby obliczyć ilość należy liczbę 2 podnieść do potęgi oznaczającej ilość bitplanów.

Liczby od 1 do 6 to tryb Lowres, czyli rozdzielczość 320x256 punktów. Jeżeli potrzebujesz skorzystać z ekranu Hires (640x256 pikseli), dodaj liczbę 8 do numeru trybu. Rozdzielczość zostanie zmieniona, co widać szczególnie po tytule na listwie ekranu. Dla porównania zobacz ilustrację:



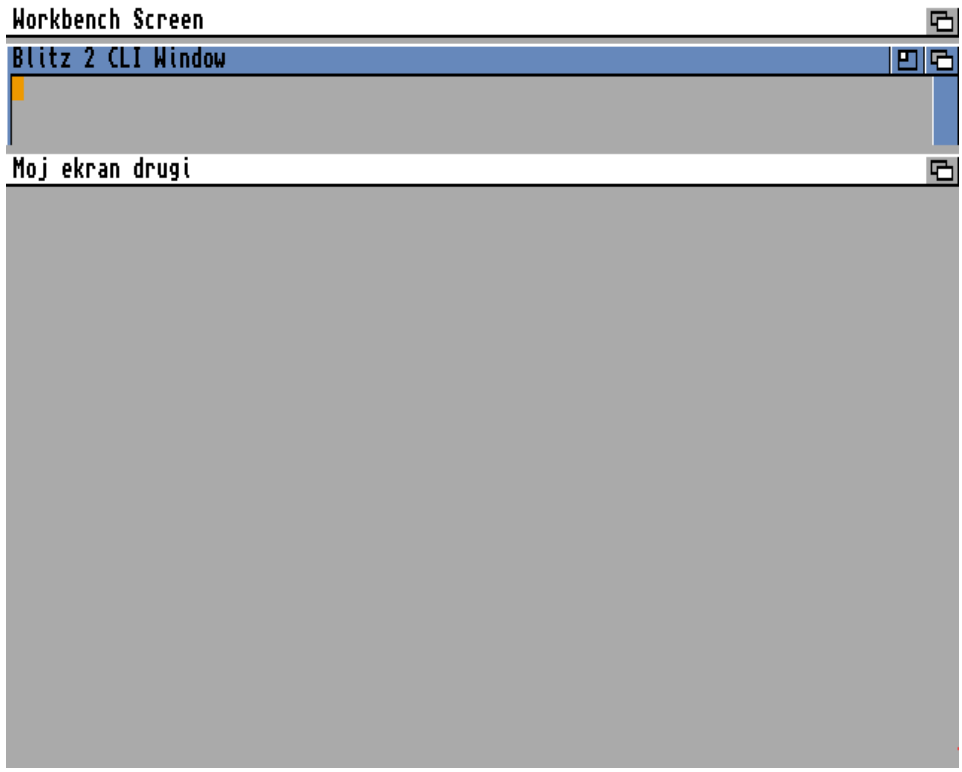
Możliwe jest również uruchomienie programu na ekranie w trybie Interlace, czyli z zachowaniem przeplotu. W tym wypadku należy dodać liczbę 16.

Z polecenia SCREEN możesz korzystać również w bardziej wyrafinowany sposób. Należy jednak podać aż 10 argumentów, które będą zachowywać inną kolejność. Spójrz na przykład:



Dzięki widocznej linii otworzysz ekran o rozdzielczości 640x256 z minimalną ilością 2 dostępnych kolorów (czyli 1 bitplan). Tak się dzieje, bowiem użyliśmy argumentu zapisanego jako liczba $8000.

Zwróć uwagę, że mamy nie tylko większą ilość danych w linii, ale także mają one różne rodzaje. Oto efekt działania programu:



Jak widzisz ekran jest automatycznie przesunięty w pionie, tak więc górna część Workbencha wraz z oknem „CLI” jest cały czas widoczna. To wszystko jest możliwe, bowiem polecenie SCREEN wymaga wpisania argumentów w istotny sposób wpływających na cechy ekranu. Są to poniższe elementy, w kolejności:

- numer ekranu,
- pozycja pozioma ekranu,
- pozycja pionowa ekranu,
- rozdzielczość pozioma,

- rozdzielczość pionowa,
- „głębokość” ekranu,
- tryb wideo,
- tytuł ekranu.

Die ostatnie liczby oznaczają kolory, które będą używane – powiemy o nich szczegółowo później.

Pozostałe pozycje oprócz trybu wideo powinny być dla Ciebie zrozumiałe. Tutaj muszą znaleźć się konkretne wartości oznaczające:

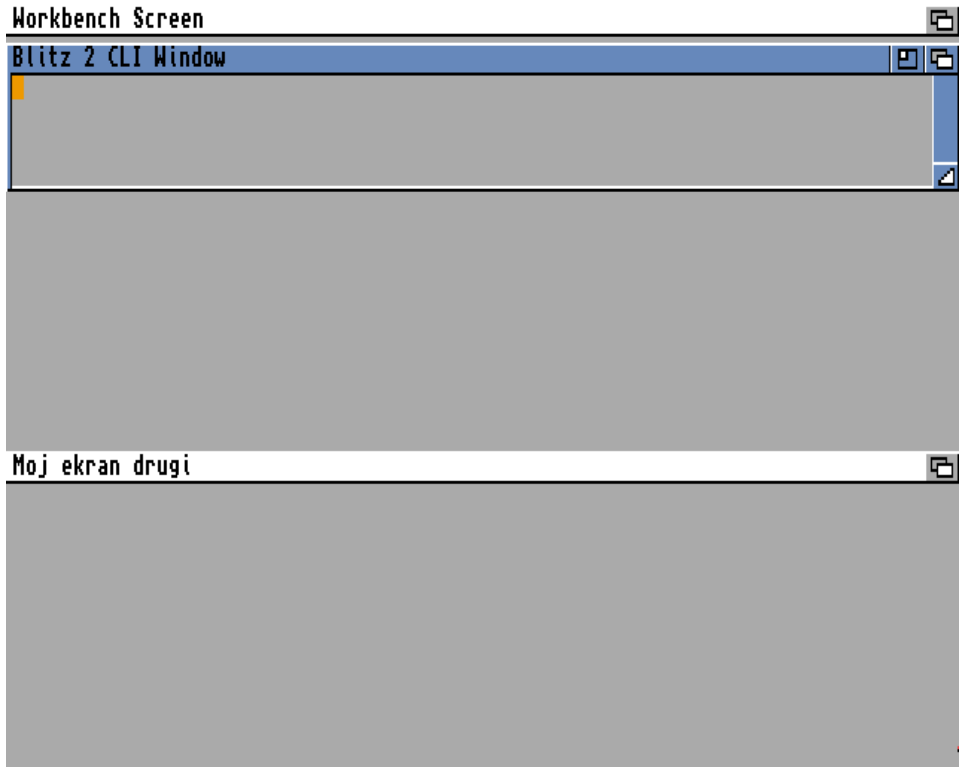
- 0 - tryb Lowres,
- $8000 - tryb Hires,
- 4 - tryb Interlace (512 punktów w pionie),

- Zmiana pozycji ekranu

Gdy już mamy gotowy ekran możemy wykonać na nim szereg dalszych czynności. Charakterystyczną cechą systemu Amigi jest funkcja przesuwania ekranów. Zwykle robimy to za pomocą myszki, gdy uruchomimy większą ilość programów. Jednak nic nie stoi na przeszkodzie, abyś wykorzystał to w swoim programie. Ekran może zmieniać pozycję według liczb podanych obok polecenia o nazwie MOVESCREEN. Musisz jedynie wpisać odległość od lewego górnego rogu, o którą ekran ma być przesunięty. Na przykład:

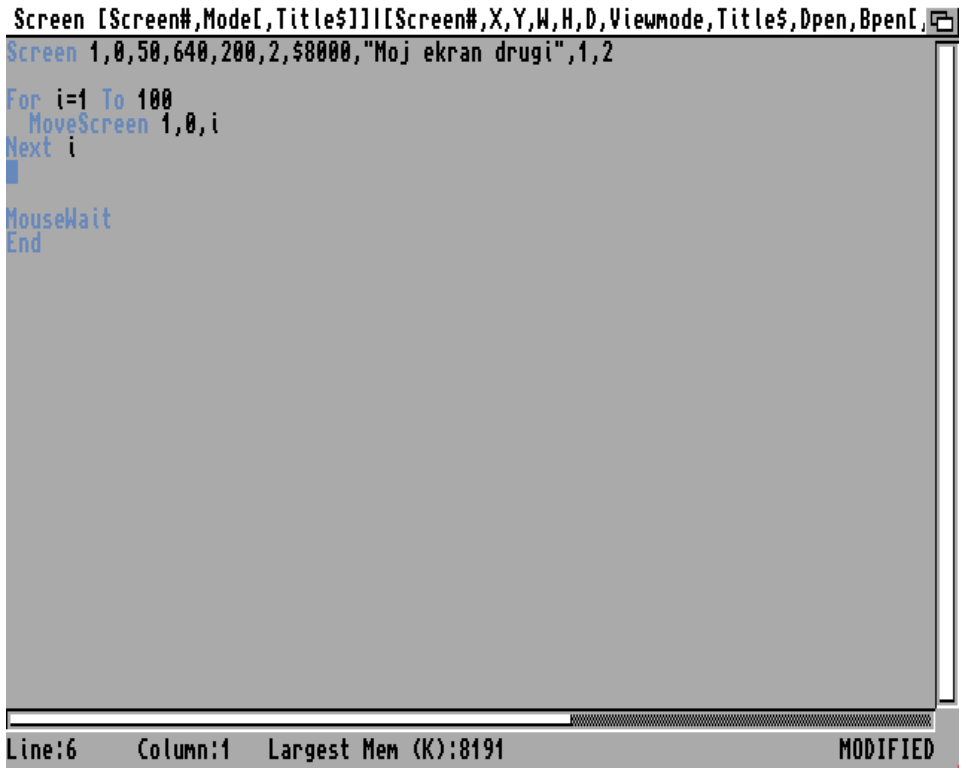
```
MoveScreen 0,0,100
```

Zobacz jaki będzie efekt działania programu, który najpierw otwiera nowy ekran, a następnie przesuwa go o wartości podane powyżej:



Oczywiście użytkownik nie obserwuje momentu zmiany pozycji ekranu, wszystko dzieje się na tyle szybko, że po prostu zobaczysz ekran „zsunięty” w dół.

Dzięki tej funkcji możesz spowodować płynne przemieszczanie się ekranu, wystarczy utworzyć pętlę modyfikującą liczby obok słowa MOVESCREEN. Oto kolejny przykład, tym razem fragment programu w edytorze „Ted”:



Tym razem efekt będzie widoczny. W przypadku programu użytkowego trudno znaleźć zastosowanie takiego działania, ale możesz to śmiało wykorzystać przy pisaniu gry.

Zwróć uwagę, że możesz otworzyć kilka ekranów i zmieniać ich pozycje względem siebie. Robi tak wiele gier dla Amigi. Aby skutecznie zarządzać wieloma ekranami powinieneś jednak poznać polecenia umożliwiające zmianę ich kolejności, czasowe ukrywanie oraz wyświetlanie odpowiednich fragmentów.

Najprostsza operacja to schowanie ekranu o określonym numerze. W tym celu wprowadź linię podobną do poniższej:

```
HideScreen 1
```

Obok polecenia podajemy tylko numer ekranu, który został wcześniej otwarty. Tak samo możliwe jest pokazanie na nowo ekranu, musimy jednak użyć słowa SHOWSCREEN, czyli razem:

```
ShowScreen 1
```

Pamiętaj, że w przypadku utworzenia wielu różnych ekranów, ich kolejność może być różna. Otwarty ekran niekoniecznie musi być tym, który jest „na wierzchu". Polecenie wykonuje dodatkowo wykonuje operację ustawienia ekranu o podanym numerze jako pierwszego.

W rezultacie po użyciu słów HIDESCREEN oraz SHOWSCREEN zmienisz kolejność obszarów roboczych. Jeżeli piszesz program operujący na kilku ekranach, a na każdym z nim umieszczasz ważne elementy sterujące, może mieć to kolosalne znaczenie dla czytelności tak utworzonego interfejsu użytkownika.

Może Ci się również przydać opcja wyszukania ekranu o określonym numerze. Służy do tego polecenie FINDSCREEN, któremu można dodatkowo podać tytuł ekranu. Funkcję można wykorzystać także w taki sposób:

```
FindScreen 0
```

co będzie oznaczało, że

Zwróć uwagę, że po wykonaniu tej operacji funkcji, podany ekran stanie się automatycznie aktualnie używanym. Oznacza to, że możesz na nim wykonywać różne operacje bez potrzeby pokazywania użytkownikowi Twojego programu. W wielu przypadkach będziesz miał potrzebę „cichej" zmiany zawartości ekranu, dlatego warto pamiętać o tej możliwości.

Operacje możesz również wykonywać bezpośrednio na ekranie Workbencha. Możesz to zrobić podając numer zerowy obok polecenia FINSCREEN, więc po prostu:

```
FindScreen 0
```

Możesz mu także przypisać własny numer, a następnie wykorzystywać go tak jak inne otwarte ekrany. Aby było to możliwe musisz użyć słowa WBTOSCREEN, zamiast FINDSCREEN.

Po przypisaniu numeru do ekranu Workbencha możliwe jest na przykład otwieranie na nim własnych okien. Innymi słowy Twój program może działać na pulpicie, ale nie wyklucza to równocześnie wykonywania operacji na innych ekranach. Pamiętaj jednak, że Amiga posiada ograniczoną pamięć graficzną (typu Chip). Im więcej ekranów w wysokiej rozdzielczości oraz kolorów, tym bardziej zużywasz dostępny obszar. Dlatego polecenie WBTOSCREEN może być przydatne także wtedy, gdy chcesz zmniejszyć zapotrzebowanie na pamięć.

Ekran możesz zamknąć w razie potrzeby i nie wymaga to specjalnych umiejętności. Trzeba tylko wpisać polecenie CLOSESCREEN i podać numer ekranu. Zanim to jednak zrobisz warto poznać inne możliwości modyfikacji ekranu, na którym pracować ma Twój program.

PALETA KOLORÓW

Każdy ekran może mieć swoją własną paletę kolorów. Barwy w systemie Amigi są reprezentowane jako wartości trzech podstawowych składowych:

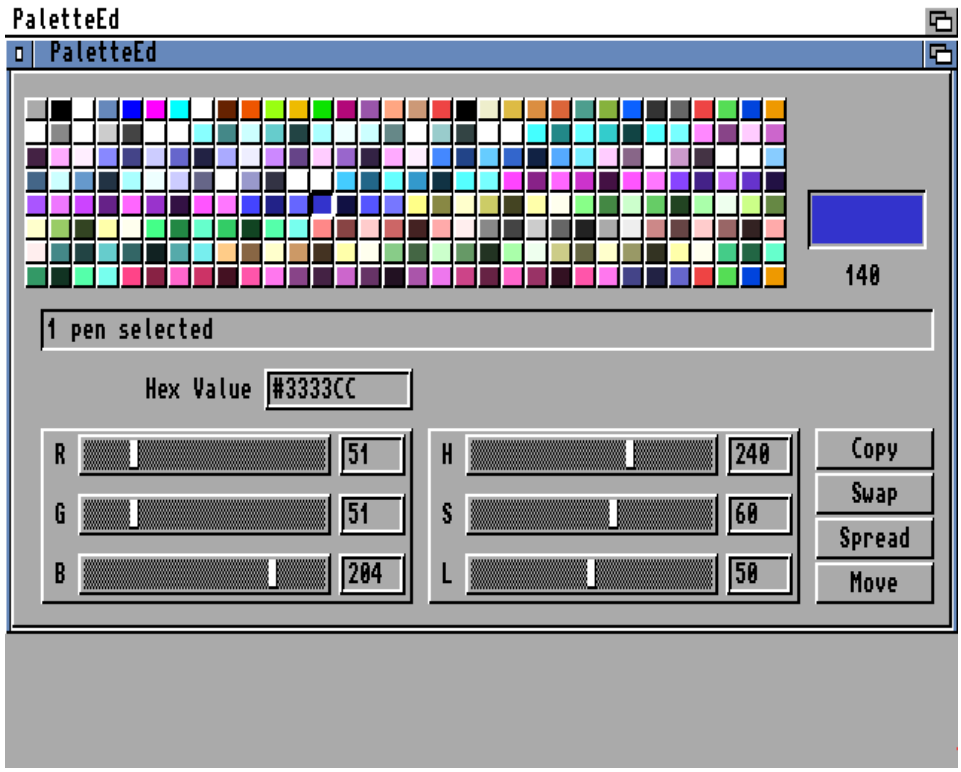
- czerwonej (ang. Red),
- zielonej (ang. Green),
- niebieskiej (ang. Blue).

Te wartości są zapisywane w formie tak zwanego modelu RGB. Jak łatwo można się domyślić, jego nazwa powstała ze złożenia pierwszych liter angielskich słów określających nazwy poszczególnych składowych. Dzięki różnym kombinacjom możemy otrzymać bardzo szeroki zakres kolorów. Cały czas musisz pamiętać, że ekran ma ustawioną określoną ilość dostępnych kolorów, ale ich wartości możesz zmieniać.

Zestawy kolorów nazywamy paletami. W różnych programach dla Amigi możesz modyfikować paletę oraz zapisywać ją w oddzielnym pliku. Przykładem może być program „PaletteEd" znajdujący się w zasobach serwisu Aminet, czyli na stronie:

aminet.net

Zobacz jak wygląda edycja palety kolorów:



Pokazujemy to nie bez powodu, bowiem w Blitz Basicu możesz wiele operacji na paletach kolorów. Informacje o palecie można załadować z pliku w formacie IFF lub zdefiniować za pomocą kilku poleceń. Najprostszym jest słowo RGB, które umożliwia zmodyfikować jeden z kolorów palety używanego ekranu. Wystarczy wpisać linię podobną do poniższej:

```
RGB 1,0,15,15
```

Obok podajemy numer koloru oraz wartości składowych zawsze według tego samego schematu – czerwona, zielona, niebieska. Każda

liczba musi zawierać się w przedziale od 0 do 15. Pozwala to dostosować kolory na ekranie do swoich potrzeb. Zwróć uwagę, ze w tym przypadku zmiany wykonywane są natychmiast.

Zwróć uwagę, że różne obiekty na ekranie mogą korzystać z osobnych palet. Dlatego, oprócz określania kolorów dla aktualnego ekranu, możesz również zmienić zestaw barw o określonym numerze. Osiągniemy to przy użyciu polecenia PALRGB, które należy zastosować bardzo podobnie do poprzedniego.

Główną różnicą jest fakt, iż na początku trzeba podać numer palety. Oto przykład:

```
PalRGB 0,1,0,15,15
```

Powyższe dwie funkcje działają w ramach możliwości układów graficznych Amigi, ale nie uwzględniają chipsetu AGA. Jak wiemy, pozwala on uzyskać większą ilość kolorów, a więc powinieneś móc ustawiać barwy w sposób bardziej dokładny. I tak jak jest rzeczywiście, bowiem istnieją odpowiedniki obu poleceń przeznaczone dla użytkowników Amigi 1200 i 4000. Mają one następujące nazwy:

- AGARGB,
- AGAPALRGB.

Ich sposób obsługi jest identyczny, ale teraz każda składowa może przyjmować wartość od 0 do aż 255. Pamiętaj, że ta różnica ma związek z budową układów specjalizowanych Amigi, dlatego nie uzyskasz podobnego efektu na Amidze 600 wyposażonej w układy ECS.

Dzięki powyższym funkcjom zmienisz określone kolory w palecie, a mówiąc bardziej precyzyjnie, zdefiniujesz ich nowe wartości. Jednak barwy nie są samoczynnie modyfikowane. Zmianę musisz wywołać ręcznie korzystać z polecenia SHOWPALETTE. Obok podajemy tylko numer, czyli przykładowo:

```
ShowPalette 2
```

Spowoduje to zmianę kolorów w palecie numer 2, która może być przyporządkowana do różnych elementów ekranu.

Ręczne ustawianie wszystkich kolorów nie jest zbyt wygodne, dlatego Blitz Basic daje możliwość wykonywania operacji zapisywania i odczytywania gotowej palety z dysku. Wymaga to użycia polecenia SAVEPALETTE lub LOADPALETTE. Aby to łatwiej zrozumieć zaczniemy od opcji zapisywania. Wpisując linię jak na poniższym przykładzie:

```
SavePalette 0,"Worek:plik3"
```

spowodujesz utworzenie pliku w formacie IFF na dysku „Worek". Jest to ważne, bowiem pliki tego typu są bardzo często kojarzone tylko z grafiką, którą można załadować do programów takich jak „Deluxe Paint" czy „Personal Paint".

Tymczasem Interchange File Format (w skrócie IFF) został opracowany przez firmę Electronic Arts w połowie lat '80-tych do używania w dużo szerszym zakresie. Może on przechowywać różne rodzaje danych, w tym oczywiście grafikę i dźwięk.

System Amigi jest bardzo mocno związany w formatem IFF i tak samo zachowuje się nasz język programowania. Dlatego też paleta kolorów będzie zapisana w formacie IFF CMAP. Z praktycznego punktu widzenia musisz tylko wiedzieć, że taki plik nie zawiera informacji o grafice, lecz tylko definicję zestawu kolorów.

Wczytywanie palety nie różni się zasadniczo od operacji zapisywania. Możesz użyć takiej linii:

```
LoadPalette 1,"Worek:plik5"
```

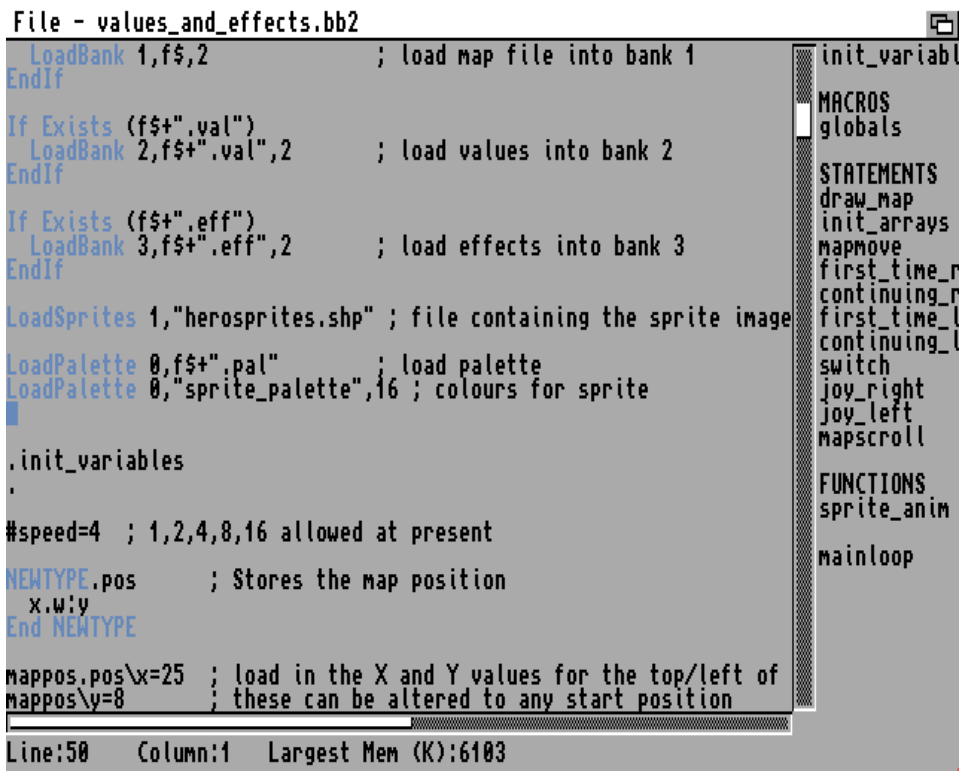
aby załadować paletę z zapisanego wcześniej pliku. Możliwe jest również „wydobycie" palety z pliku typu IFF ILBM, czyli grafiki jaką można oczywiście wyświetlić na ekranie. W tym wypadku program jedynie odczyta informacje o zestawie barw i nie spowoduje żadnych innych zmian na dysku. Dzięki temu możesz korzystać z różnych plików w formacie IFF bez obawy o uszkodzenie danych.

Osobne palety kolorów możesz również przygotować w innych programach. Przykładem może być wspomniany już program graficzny „Personal Paint".

Zarządzanie paletami możesz wykonywać w wielu programach, ale musisz pamiętać, aby pliki zawsze zapisywać w formacie IFF. W przeciwnym razie nie będziesz mógł ich później wykorzystać pisząc program w Blitz Basicu, chyba że zastosujesz dużo bardziej skomplikowane operacje. Dlatego lepiej nie zmieniać domyślnego formatu, przynajmniej na początku.

Wczytanie palety za pomocą polecenia LOADPALETTE – podobnie jak wcześniej - nie zmienia automatycznie wyglądu ekranu. Aby zmiany zostały wykonane należy użyć polecenia USEPALETTE podając obok numer palety.

Dla ułatwienia zobacz jak może to wyglądać w bardziej skomplikowanym listingu:



Paletę możesz też skopiować, aby użyć w inny sposób, na przykład dla innego ekranu. Aby to osiągnąć użyj polecenia DUPLICATEPALETTE w następujący sposób:

```
DuplicatePalette 1,2
```

Jako argumenty podajemy numer palety źródłowe, a następnie docelowej. Słowo to nie ma dodatkowych możliwości, ale może być przydatne na przykład, gdy chcesz wyświetlić grafikę stworzoną w tym samym programie graficznym albo potraktujesz jedną z palet jako „roboczą".

Oprócz manipulowania paletą, przydatną opcją jest także odczytywanie informacji o konkretnych kolorach. Nie zawsze będziesz chciał je zmieniać, czasem dla osiągnięcia odpowiedniego rezultatu wystarczy skopiować barwę i użyć ją w stosunku do innego ekranu lub innego elementu.

Dlatego Blitz Basic udostępnia polecenia, za pomocą których odczytasz informacje o używanych barwach. Mają one nazwy zbieżne ze składowymi – w języku angielskim, czyli:

- RED,
- GREEN,
- BLUE.

Wystarczy podać im numer koloru jaki chcesz sprawdzić, na przykład tak:

```
a=Red(1)
```

W odpowiedzi uzyskasz wartość składowej barwy, która będzie odpowiadała ustawieniom możliwym do wykonania za pomocą omówione już polecenia RGB oraz PALRGB. Będą to więc liczby od 0 do 15. Wcześniej powiedzieliśmy, że istnieją osobne funkcje przeznaczone dla użytkowników układów AGA pozwalające uzyskać większą dokładność – od o do 255 dla każdej składowej. Podobnie możesz użyć poleceń rozpoczynających się od symbolu „AGA", które mają analogiczną funkcję. Sa to słowa:

- AGARED,
- AGAGREEN,
- AGABLUE.

MENU GÓRNE

Oprócz używania typowych ekranów, Twój program może mieć własne menu górne. Nie jest to nic specjalnego, bo prawie każdy większy program je zawiera. Zanim przedstawimy polecenia pozwalające utworzyć menu warto dodać, w jaki sposób należy zaplanować jego konstrukcję.

W teorii menu może zawierać dowolne elementy, ale zwróć uwagę, że większość programistów trzyma się schematu znanego z systemu operacyjnego. Ujednolica to obsługę standardowych funkcji, a dla użytkownika oznacza to łatwiejsze korzystanie z Twojego programu. Przyjrzyjmy się więc podstawowym założeniom.

Menu górne to kolejny element charakterystyczny dla systemu Amigi. Jest wyświetlane w górnej części ekranu, tam gdzie znajduje się jego listwa. Przeciwieństwie do innych systemów operacyjnych, menu nie jest widoczne cały czas podczas pracy. Należy je odpowiednio wywołać. W tym celu należy najechać wskaźnikiem na listwę ekranu, a następnie nacisnąć i przytrzymać prawy klawisz myszki. Na listwie ekranu widoczne są nazwy poszczególnych grup menu. Aby wyświetlić opcje każdej grupy najeżdżamy wskaźnikiem na nazwę, cały czas trzymając prawy klawisz myszki.

Wszystkie opcje wyświetlane są w formie listy, stosownie do ruchów wskaźnika. Jeżeli najedziemy na nazwę opcji, zostanie ona podświetlona, lecz jeszcze nie uruchomiona. Dopiero po zwolnieniu prawego klawisza myszki, opcja zostanie wywołana, a całe menu automatycznie zniknie

Menu górne związane jest zwykle z ekranem, dlatego niezależnie od tego, jaką część programu obsługujemy, uzyskamy zawsze te same opcje. Może być jednak związane także z konkretnym oknem. Wtedy, w zależności od tego, które okno będzie aktywne, na listwie ekranu wyświetlone zostaną inne funkcje menu górnego. Sytuacja taka występuje szczególnie często, gdy uruchamiamy na tym samym ekranie wiele programów, a więc otworzymy kilka oddzielnych okien.

Jeśli korzystamy z programu, który otwiera własny ekran, od razu uzyskujemy dostęp do jego menu. Natomiast w przypadku, gdy program otwiera swoje okno, musimy je uaktywnić, w przeciwnym razie jego menu górne będzie niedostępne. Rzadko się zdarza, że nie będziemy mieli żadnych opcji do wyboru, najczęściej uzyskamy dostęp do jednego z menu, niekoniecznie tego, które nas interesuje. Dlatego przy pracy z wieloma programami na raz, należy zwracać szczególną uwagę na to, na jakim ekranie operujemy oraz które okno jest aktywne.

Zwróćmy także uwagę, że nie wszystkie opcje menu górnego są aktywne. Niektóre nazwy mogą mieć ciemniejszą barwę i nie będzie można ich wybrać. Oznacza to, że w danej chwili są niedostępne i zostaną uaktywnione samoczynnie, gdy będzie to możliwe, np. w momencie uruchomienia odpowiedniej sekcji w programie. Część opcji menu może również posiadać tzw. pod-menu (ang. sub-menu). W takiej sytuacji najechanie wskaźnikiem myszki na nazwę spowoduje wyświetlenie listy z dodatkowymi opcjami umieszczonej obok. Dzieje się tak, gdy dana funkcja udostępnia bardziej szczegółowe możliwości i jest ich duża ilość. Umieszczanie wszystkich w głównym menu jest wtedy niewskazane lub wręcz niemożliwe ze względu na brak miejsca.

Obsługa pod-menu jest analogiczna do zwykłego menu. Wybranie konkretnej opcji powoduje zwykle efekt w postaci uruchomienia programu, wyświetlenia komunikatu lub otwarcia okna ana ekranie. Istnieją także opcje, których wybranie powoduje jedynie zmianę statusu funkcji, tj. jej „włączenie" lub „wyłączenie". Można to łatwo poznać, gdyż po wybraniu zostanie ona zaznaczona charakterystycznym znakiem z lewej strony. Włączenie opcji może skutkować dodatkowymi czynnościami programu, lecz nie jest to regułą.

- Typowa budowa menu

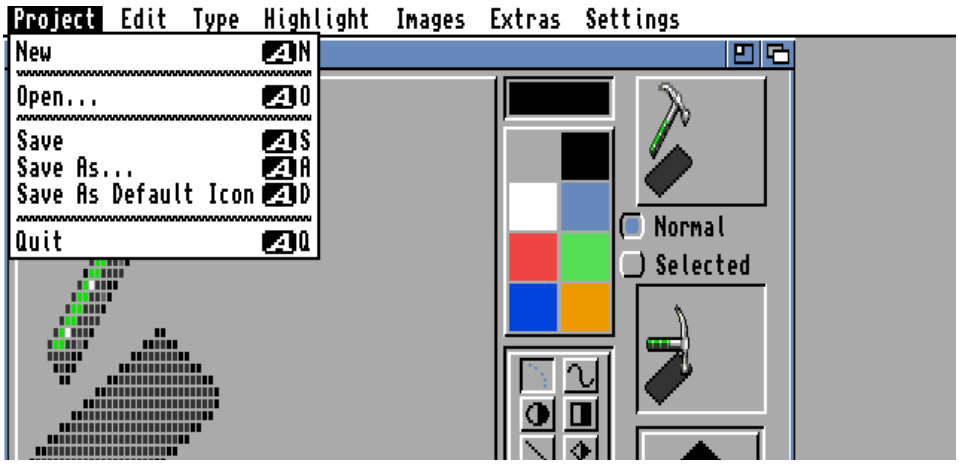
W większości programów widzimy szereg opcji, które znajdują się również w podstawowych programach systemowych. Zwróć uwagę na typowy układ menu. Znajdują się w nim takie pozycje jak:

- „Projekt" (ang. „Project"),
- „Edycja" (ang. „Edit")
- „Ustawienia" (ang. „Settings" lub „Preferences").

Menu „Projekt" jest zawsze związane z produktem działania programu, czyli zapisywanym plikiem. Znaleźć tu można funkcje operacji na plikach:

- „Wczytaj" lub „Załaduj" (ang. „Open" lub „Load"),
- „Zapisz" i „Zapisz jako" (ang. „Save" i „Save As"),

tworzenie nowego projektu - „Nowy" (ang. „New"), a także podstawowe informacje o programie i opcję wyjścia z programu - „Zakończ" lub „Skończ" (ang. „Quit").



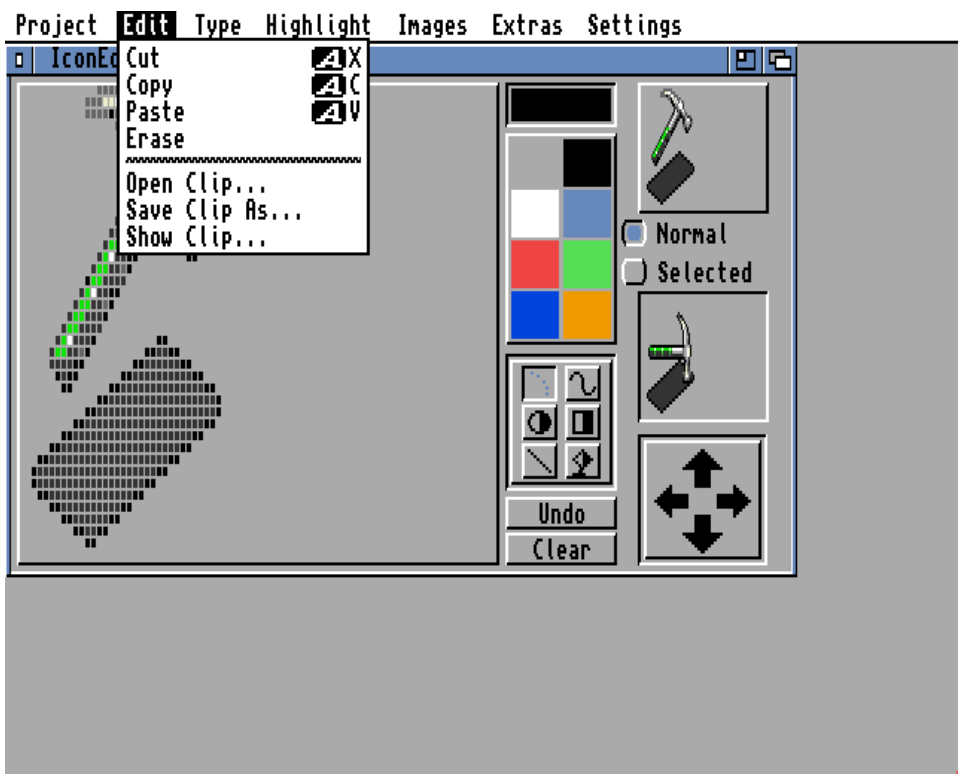
Dzięki temu najbardziej typowe funkcje pozostają identyczne i łatwiej można nauczyć się obsługi. Ta standaryzacja postępowała wraz z rozwojem systemu operacyjnego, dlatego im nowszy program, tym większa szansa, że znajdziemy w nim typowe elementy. Pamiętaj, że pisząc swój własny program nie powinieneś tworzyć własnych rozwiązań, a raczej korzystać z wymienionych elementów, chyba że ich zmiana jest absolutnie konieczna.

Warto też podkreślić, czym różni się opcja „Zapisz” od „Zapisz jako”. Jeśli użytkownik zapisuje projekt pierwszy raz, obie działają tak samo. Po ich wybraniu na ekranie pojawia się okno wyboru, w którym wskazujemy nazwę pliku, pod jakim zostanie zapisany nasz projekt. Jeśli jednak projekt był już wcześniej zapisywany, funkcja „Zapisz” automatycznie zapisuje go pod tą samą nazwą, tracąc jednocześnie wcześniejszy zapis. Gdy chcemy zapisać plik pod inną nazwą należy skorzystać z opcji „Zapisz jako”. Wyświetla ona zawsze okno wyboru, w którym decydujemy o nazwie pliku.

Drugie typowe menu to „Edycja” (ang. „Edit”). Zawiera opcje związane z operacjami na poszczególnych elementach projektu. Najczęściej występują tu funkcje takie jak:

- „Wytnij” (ang. „Cut”),
- „Skopiuj” lub „Kopiuj” (ang, „Copy”)
- „Wstaw” (ang. „Paste”).

Opcje te korzystają z tak systemowego Schowka (ang. Clipboard). Jego szerszym użyciem w programie Blitz Basica zajmiemy się później.



Opcja „Wytnij” powinna umożliwiać – zgodnie ze swoją nazwą - wycięcie zaznaczonej wcześniej część projektu i zapamiętanie jej tymczasowo w pamięci komputera. Sposób zaznaczania zależy od konkretnego programu, zwykle polega na podświetleniu fragmentu zawartości przy użyciu myszki.

Wycięty element zostaje zapamiętany i można go później wstawić w inne miejsce za pomocą drugiej wymienionej opcji „Wstaw”. Powinieneś też przewidzieć możliwość zapamiętania zaznaczonego fragmentu bez jego wycinania, dzięki czemu można duplikować zawartość projektu. Służy do tego opcja „Skopiuj” lub „Kopiuj”.

Ostatnim standardowym menu jest pozycja o nazwie „Ustawienia” lub „Preferencje”. Prawie zawsze znajduje się w prawej części ekranu, jako ostatnie lub przedostatnie menu. Posiada opcje umożliwiające wczytanie lub zapisanie ustawień programu – odpowiednio za pomocą funkcji:

- „Wczytaj ustawienia” (ang. „Load Settings”),
- „Zapisz ustawienia” (ang. „Save Settings”).

Można spotkać również opcję „Zapisz ustawienia jako” (ang, „Save Settings As”), która ma taką samą funkcję jak „Zapisz jako”, lecz dotyczy oczywiście pliku ustawień programu, a nie projektu

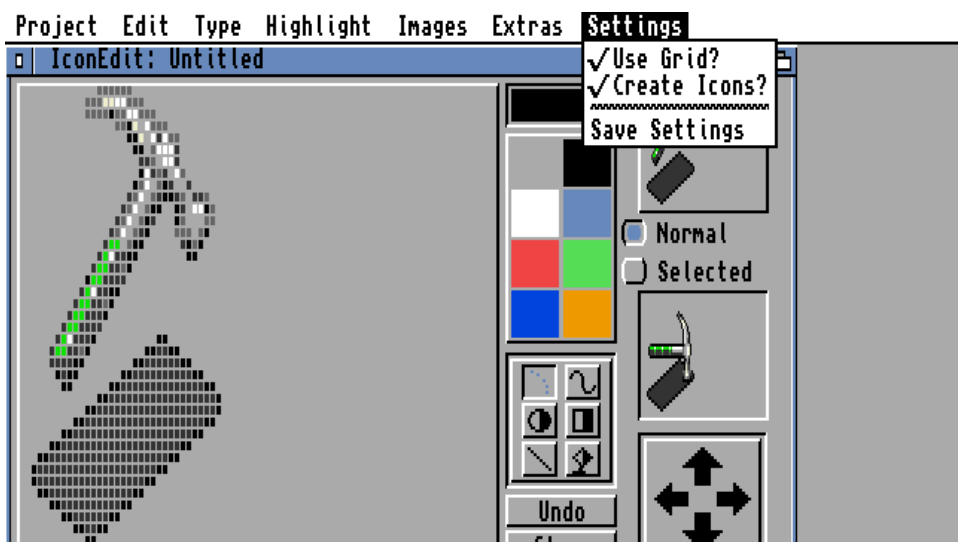
Bardzo często powtarza się także funkcja „Zapisuj ikony” lub umieszczona w formie pytania „Zapisywać ikony?” (ang. „Save Icons?”). Jeśli zostanie wybrana, wraz z plikami zawierającymi ustawienia będą zapisywane pliki ikon. Dzięki temu możliwe jest wczytanie ustawień

przez wskazanie ikony za pomocą myszki, choć nie w każdym przypadku będzie to możliwe. Jeżeli Twój program pozwala zapisywać własne pliki projektów, powinieneś także przewidzieć automatycznie zapisywanie ikon. Pamiętaj, że użytkownik powinien otrzymać wsparcie, aby mógł obsługiwać program w jak najprostszy sposób.

Niektóre programy posiadają ponadto opcje o nazwach:

- „Ustawienia standardowe” (ang. „Reset To Defaults” lub „Default Settings”),
- „Ustawienia ostatnio zapisane” (ang. „Last Saved”).

Służą one do automatycznego wczytywania plików ustawień, ale według ściśle określonych reguł. Pierwsza funkcja przywołuje domyślne wartości programu, ustalone przez jego autora. Druga powoduje załadowanie zapisanego przez użytkownika pliku ustawień.



Nie powiedzieliśmy jeszcze o czysto informacyjnej opcji menu jaką jest funkcja „ O programie” (ang. „About”). Powinna ona wyświetlać małe okno z podstawowymi informacjami o programie, jego wersji, autorze i dacie wydania. Możesz tez umieścić tu instrukcje dotyczące kontaktu z Tobą lub uzyskania szczegółowej dokumentacji programu.

Jeżeli przewidujesz różne edycje tego samego programu, na przykład rejestrację zwiększającą ilość dostępnych funkcji, w oknie „O programie” warto też podać symbol edycji, nazwisko osoby rejestrującej albo numer licencji. Niektórzy zapisują też tu prośby o dotacje, choć nie jest to najlepsze miejsce do takich wiadomości.

Te wszystkie uwagi powinieneś wziąć pod uwagę podczas tworzenia menu górnego w programie. Oczywiście nie musisz używać wszystkich elementów, bowiem wszystko zależy od przeznaczenia Twojego gotowego produktu.

Najlepiej działać w myśl zasady mówiącej, iż należy dodawać funkcje do menu na bieżąco, podczas pisania kolejnych modułów programu, a nie tworzyć nazwy wszystkich możliwości, które nie działają. Niemniej uważamy, że podstawowe elementy jak menu „Projekt” czy „Edycja” warto zaplanować z góry, a później jedynie rozbudowywać menu górne.

- Tworzenie menu

Gdy już wiesz technicznie, jak powinno wyglądać menu, czas na zastosowanie tych informacji w praktyce. Aby utworzyć menu górne należy użyć grupy poleceń o nazwach rozpoczynających się od słowa MENU.

Pierwszym elementem jest MENUTITLE, za pomocą którego dodajemy tytuł menu, który później będzie zawierał opcje. Obok trzeba podać:

- numer przypisany do całego menu górnego jakie tworzymy,
- numer tytułu menu,
- treść tytułu.

Może to wyglądać na przykład tak:

```
MenuTitle 0,0,"Projekt"
```

Dzięki powyższej linii stworzysz pierwsze menu o nazwie „Projekt”. Zwróć uwagę, że liczba porządkowa jest liczona od zera. Do „tytułu” trzeba teraz dodać funkcje, na przykład „Otwórz” czy „Zapisz”. To z kolei wykonujemy za pomocą polecenia MENUITEM. Pozwala ono dodać element do pozycji utworzonej poprzez MENUTITLE. Poszczególne opcje będą się oczywiście pojawiać pionowo, poniżej tytułu.

W tym wypadku jako argumenty należy wpisać:

- numer menu ,
- tak zwane „flagi”,
- numer tytułu menu,
- numer dodawanej opcji,
- nazwę dodawanej opcji,
- skrót klawiaturowy wywołujący opcję.

Przykładowe wpisy to:

```
MenuItem 0,0,0,0,"Otwórz... ","O"
MenuItem 0,0,0,1,"Zapisz ","Z"
MenuItem 0,0,0,2,"O Programie... ","I"
```

Numer menu musi być ten sam, który podaliśmy jako pierwszy w linii MENUTITLE. Inaczej linia będzie odnosiła się do innego menu rozwijanego, a przecież ekran powinien posiadać tylko jedną grupę funkcji wywoływaną prawym klawiszem myszki. Numer tytułu menu również powinien być identycznie jak wcześniej, aby dodawana pozycja znalazła się w ramach jednej rozwijanej linii.

Następne w kolejności są: numer i nazwa opcji. Kolejne funkcje powinny mieć następujące po sobie numery. Nazwa może być dowolna, lecz powinna uwzględniać uwagi dotyczące struktury menu oraz wskazywać jako na wywoływaną funkcję. Skrót klawiaturowy oznacza w tym przypadku tylko jeden znak, który będzie trzeba użyć wraz z klawiszem AMIGA, aby alternatywnie uruchomić opcję przypisaną do danej pozycji.

Tu także powinieneś skorzystać z doświadczeń innych programistów, a więc sprawdzić jakie klawisze zwykle wywołują różne operacje. Jest to jednak mniej ważne, bowiem Twój program może mieć własne skróty także do innych funkcji.

Poza tym każdy będzie mógł na bieżąco sprawdzać klawisze „wywołujące", bowiem obok nazwy opcji pojawi się symbol, podobnie jak w innych programach.

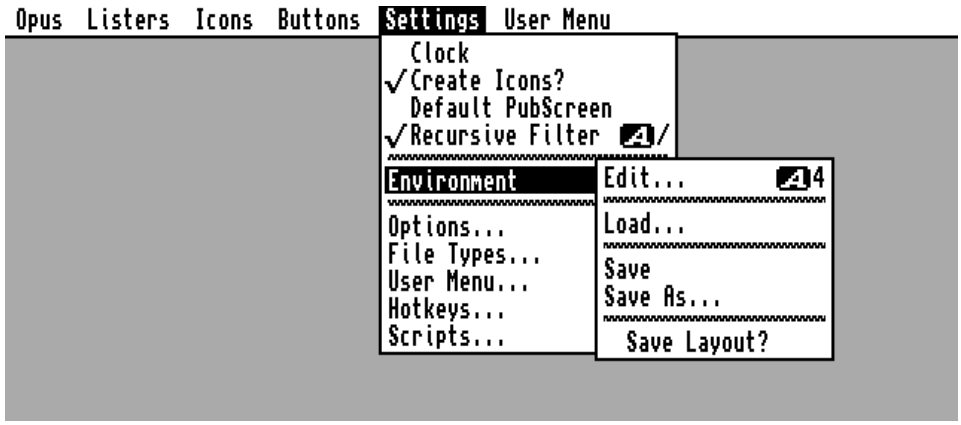
- Dodawanie opcji

Nie powiedzieliśmy jeszcze nic o bardzo ważnej kwestii jaką są „flagi" zapisywane jako drugie w linii z poleceniem MENUITEM. Przypomnijmy jak to wygląda:

```
MenuItem 0,0,0,1,"Zapisz ","Z"
```

Flaga to nic innego jak określona liczba oznaczająca rodzaj dodawanej opcji. W powyższej linii mamy zero, a więc będzie to zwykła pozycja, którą użytkownik wybiera, aby uruchomić daną funkcję. Oczywiście „Zapisz" powinna od razu wywołać okno wyboru, dlatego ten rodzaj pasuje najlepiej.

Z pewnością wiesz, że opcje mogą przyjmować inne formy. Dla ułatwienia spójrz na ilustrację:



Jeżeli chcesz uzyskać podobny efekt, musisz zmienić wartość „flagi". I tak, liczba 1 powoduje, że opcja będzie możliwa do „włączenia" i „wyłączenia", podobnie jak okno na ekranie Workbencha. Wartość 2 tworzy natomiast specjalną pozycję menu, która nie należy do standardowych elementów.

Wszystkie pozycje, które pojawią się w tym samym menu będą wzajemnie wykluczające. Oznacza to, że tylko jeden z nich może być „aktywny" na raz. Jeśli tak się stanie, wszystkie pozostałe pozycje menu zostaną automatycznie „wyłączone".

Ta ostatnia cecha może być bardzo przydatna wszędzie tam, gdzie chcesz dać użytkownikowi zawężony wybór opcji. Co prawda takie menu jest rzadko spotykane, ale nie łamie to standardów systemu operacyjnego, dlatego możesz śmiało z niego korzystać. Należy tylko przemyśleć funkcje tak, aby rzeczywiście zawsze tylko jedna opcja mogła być jednocześnie aktywna.

Cała struktura menu powinna być przygotowana według powyższego schematu. Ilość tytułów i pozycji jest dowolna, ale opcje powinny być grupowane logicznie. Ponadto pojedyncza rozwijana lista nie może być zbyt długa, bo nie zmieści się na ekranie.

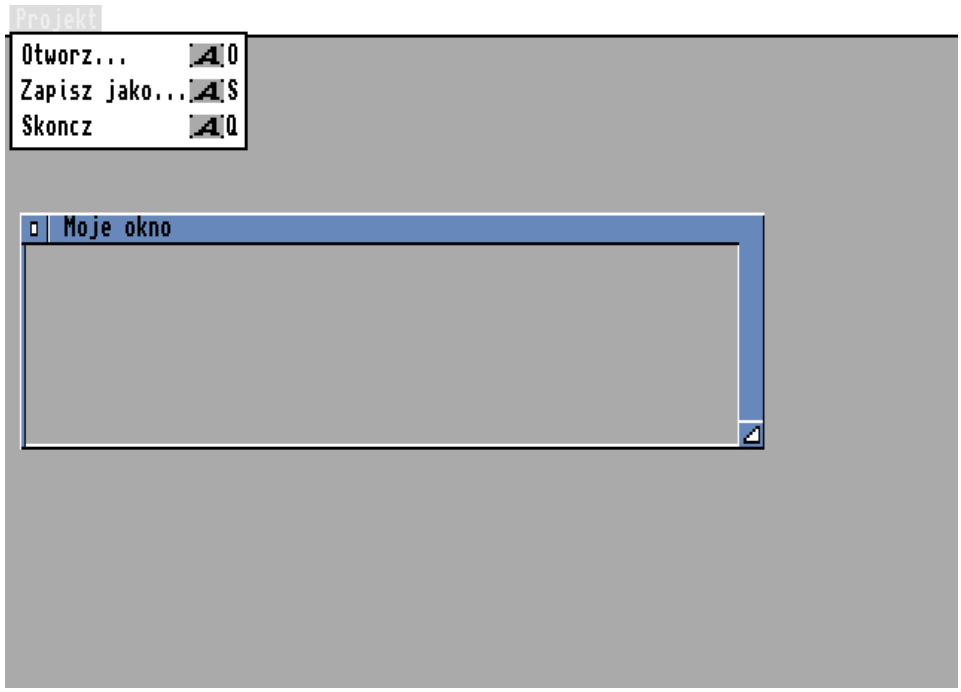
Najlepiej przyjmij, że użytkownik korzysta ze standardowej rozdzielczości 640x256 punktów oraz czcionki „Topaz” w rozmiarze 8 pikseli. Parametry te potraktuj ją jako punkt wyjścia. Jeśli musisz koniecznie użyć trybu Interlace lub innych krojów czcionek, nie wahaj się to zrobić, ale nie powinno być to także jedyne możliwe ustawienie programu. Zawsze lepiej rozbić opcje na kilka tytułów menu, niż zapisywać wszystko pod jedną rozwijaną listą.

Zobacz gotowy przykład sposobu tworzenia menu górnego – najpierw listing:

File - test.bb2

```
Screen 1,13,"Moj program"
Window 1,10,70,500,80,$9,"Moje okno",1,2

Activate 1

MenuTitle 0,0,"Projekt"

MenuItem 0,0,0,0,"Otworz...    ","O"
MenuItem 0,0,0,1,"Zapisz jako...    ","S"
MenuItem 0,0,0,2,"Skoncz    ","Q"

SetMenu 0

MouseWait
```

a teraz po uruchomieniu programu:



Jak widzisz oprócz linii tworzących menu mamy kilka innych elementów, o których będziemy mówić za chwilę. Program otwiera ekran, aktywne okno, a następnie przypisuje mu menu górne o nazwie „Projekt”.

Menu możesz przypisać do aktywnego okna za pomocą polecenia SETMENU. Podajemy mu tylko numer przypisany do całego menu górnego jakie tworzymy, czyli tę samą pierwszą pozycję, co w przypadku poprzednich funkcji. Oczywiście jedno okno może mieć przyporządkowany tylko jeden zestaw menu.

Jeśli chcesz uzyskać bardziej nietypowy wygląd Twojego programu lub wymaga tego specyficzna paleta kolorów, możesz spowodować, że tekst menu będzie wyświetlany za pomocą określonego koloru. Wpisz tylko linię podobną do poniższej:

```
MenuColor 2
```

Jak widać, trzeba tu tylko wpisać numer koloru. Zwróć uwagę, że powyższa linia powinna być wywołana przed poleceniami tworzącymi menu.

- Aktywacja menu

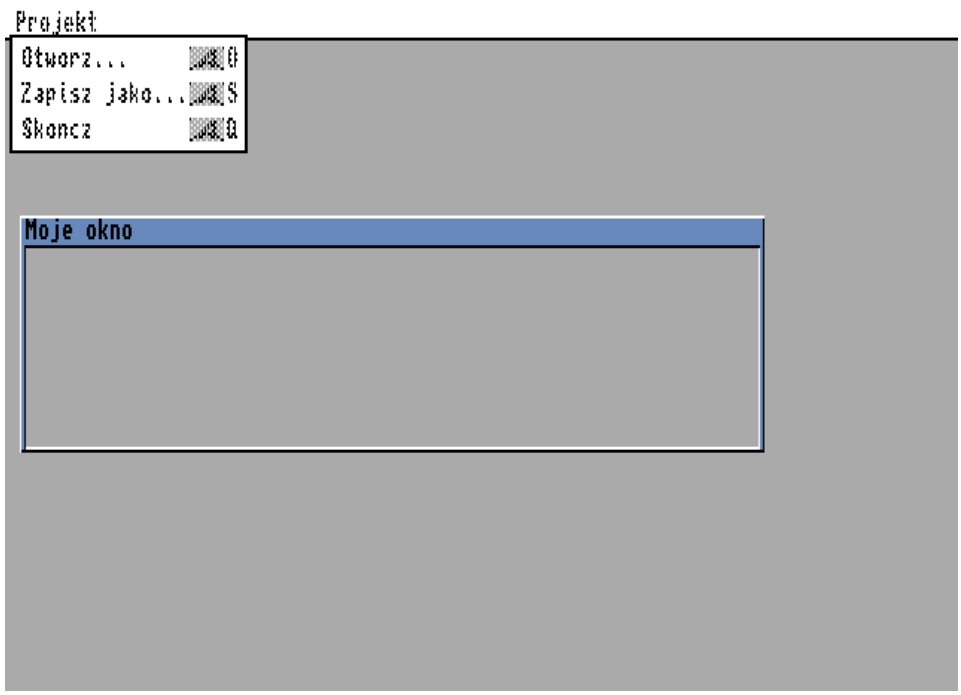
Menu górne w razie potrzeby możesz włączać i wyłączać. Służy do tego polecenie MENUSTATE. Możesz przy tym wybierać, czy chcesz dezaktywować całe menu, czy tylko jedno część. Aby było to możliwe, obok nazwy polecenia musisz podać:

- numer przypisany do menu.
- numer tytułu menu,
- numer opcji,
- słowo ON lub OFF.

Dwie pierwsze pozycje pokrywają się z wcześniejszymi poleceniami i nie wymagają komentarza. Numer opcji również nie powinien być niejasny, choć w tym wypadku trzeba dodać ważne zastrzeżenie. Jeśli opcja zawiera „pod-menu", w ten sposób włączysz lub wyłączyć całą sekcję.

Dodatkowo przed ostatnim argumentem możesz podać numer „pod-pozycji”, jeżeli chcesz wyłączyć tylko jedną z nich. Pamiętaj, że menu wraz z opcjami będzie cały czas widoczne na ekranie, ale niektóre funkcje będą nieaktywne. Takie zachowanie możesz zaobserwować w każdym programie zawierającym menu górne, jak również na Workbenchu.

Zobacz jak będzie wyglądało menu z poprzedniej ilustracji, gdy wyłączymy je w całości:



Taki efekt uzyskasz po wpisaniu skróconej wersji polecenia MENUSTATE, w której wskażesz tylko numer menu:

```
MenuState 0,Off
```

Jest to najłatwiejszy sposób zablokowania wszystkich funkcji, gdy nie chcesz, aby użytkownik mógł spowodować błąd programu, na przykład podczas wykonywania operacji dyskowych lub innych pracochłonnych zadań.

- Sprawdzanie stanu opcji

Jeśli gotowe menu górne zawiera pozycje, które można aktywować, będziesz musiał sprawdzać stan tych funkcji. Użytkownik w każdej chwili może je zmienić i program musi odpowiednio reagować na modyfikacje. Rzecz jasna, konkretne działanie zależy od Twoich pomysłów, ale musisz wiedzieć jak sprawdzić stan opcji.

Blitz Basic udostępnia w tym zakresie funkcja o nazwie MENUCHECKED(). Trzeba jej podać informacje te same, co podczas „włączania" i „wyłączania" pozycji, czyli numer menu, numer tytułu menu, numer opcji oraz ewentualnie numer „pod-pozycji".

Całość może wyglądać tak:

```
a=MenuChecked(1,0,0)
Nprint a
```

Funkcja w odpowiedzi przekaże typowe wartości , bo 0 (zero) lub -1. Pierwsza oznacza, że sprawdzana opcja jest nieaktywna, druga wskazuje na „włączenie" funkcji.

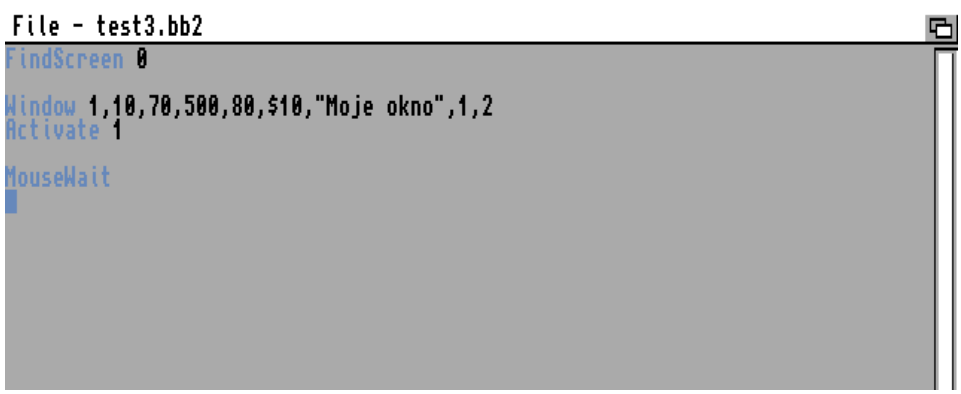
Oczywiście są to wartości logiczne oznaczające FAŁSZ lub PRAWDĘ. Więcej na ten temat przeczytasz na przykład w punkcie zatytułowanym „Operatory”.

OKNA

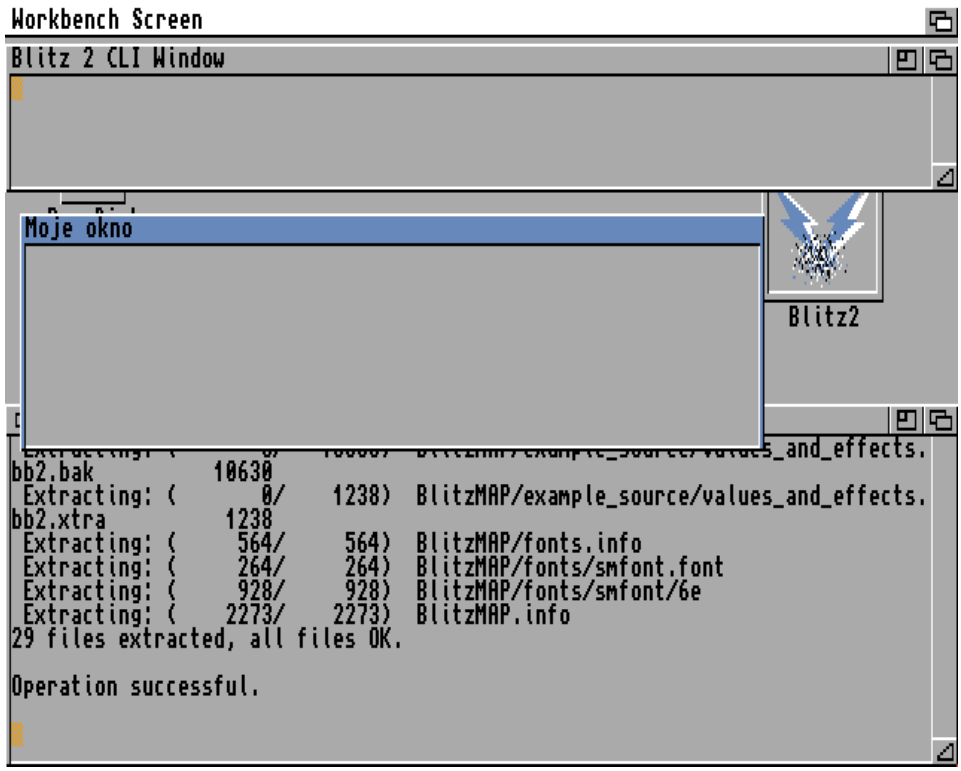
Okna są podstawowym elementem interfejsu graficznego systemu Amigi. Korzysta z nich prawie każdy program, a nawet jeśli uruchamiamy tylko polecenie AmigaDOS, wpisujemy je w oknie „Shell”. Jeśli chcesz stworzyć nową grę być może nie będziesz chciał używać standardowych okien, natomiast w przypadku program użytkowego powinieneś zachowywać wszystkie możliwe elementy systemu operacyjnego.

Blitz Basic pozwala na korzystanie z okien, podobnie jak ekranów. Do dyspozycji masz szereg poleceń, za pomocą których możesz otwierać, zamykać okna oraz wywoływać wiele operacji potrzebnych użytkownikowi podczas obsługi Twojego programu.

Na początek spójrz na przykładowy program napisany w Blitz Basicu. Jest to zmodyfikowana wersja listingu z poprzedniego punktu, która otwiera okno na ekranie Workbencha:

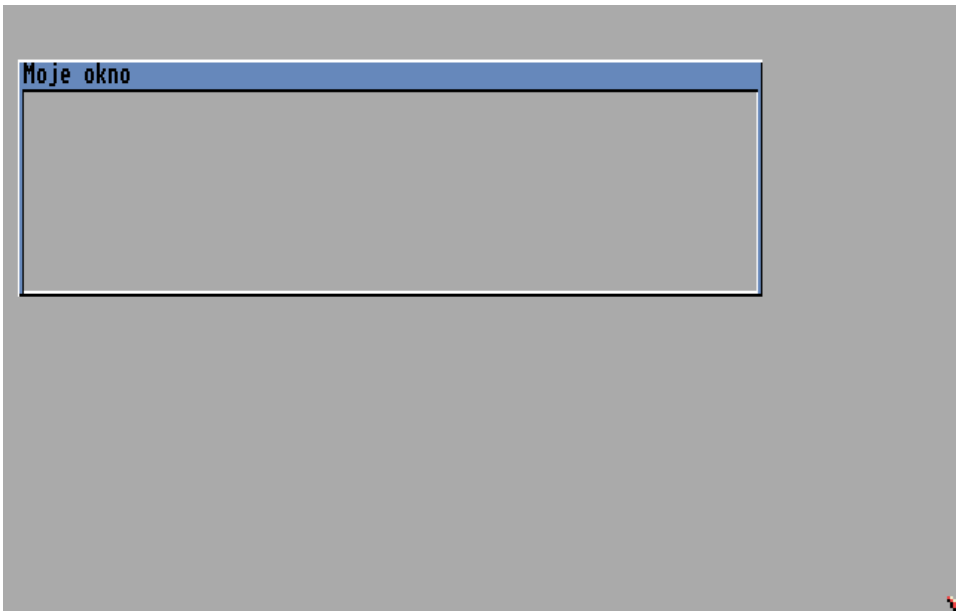


A oto efekt jego działania:



Takie okno możesz stworzyć w bardzo łatwy sposób. Wystarczy wykorzystać polecenie WINDOW. Pamiętaj, że okno będzie wyświetlane na aktywnym ekranie, a więc musisz otworzyć własny – tak jak omawialiśmy to wcześniej – lub użyć ekranu Workbencha.

Standardowe okno posiada przyciski na ramce, ale z pewnością widziałeś tez okna pozbawione niektórych elementów. Zobacz jak wygląda to samo okno otwarte na oddzielnym, pustym ekranie:



Dlatego słowo WINDOW również ma kilka argumentów, za pomocą których ustawiamy parametry okna. Aby otworzyć okno w najprostszy sposób, obok nazwy polecenia musisz podać następujące argumenty:

- numer okna,
- współrzędną poziomą lewego górnego rogu,
- współrzędną pionową lewego górnego rogu,
- szerokość okna,
- wysokość okna,
- tak zwane „flagi” przypisane do okna,
- tytuł okna.

Oznacza to użycie linii według poniższego schematu:

```
Window 1,100,100,300,50,$9,"Moje okno",1,2
```

Większość pozycji nie wymaga komentarza, ale czym są tajemnicze „flagi"? Są to wartości określające parametry okna, na przykład czy ma ono zawierać przycisk zamknięcia, czy możliwe ma być zmiana rozmiaru i inne. Wartość „flag" należy podawać rozpoczynając znakiem dolara ($).

Oto ich lista wraz z krótkim objaśnieniem:

- **$0001** - dodaje przycisk zmiany rozmiaru - w prawym dolnym rogu okna,

- **$0002** - pozwala na zmianę pozycji - okna za pomocą myszki,

- **$0004** - okno będzie mogło być - umieszczanie „z przodu" lub - „z tyłu" innych okien,

- **$0008** - dodaje przycisk zamknięcia - okna w lewym górnym rogu,

- **$1000** - aktywuje otwarte okno,

- **$0800** - powoduje, że okno nie będzie - posiadało ramek,

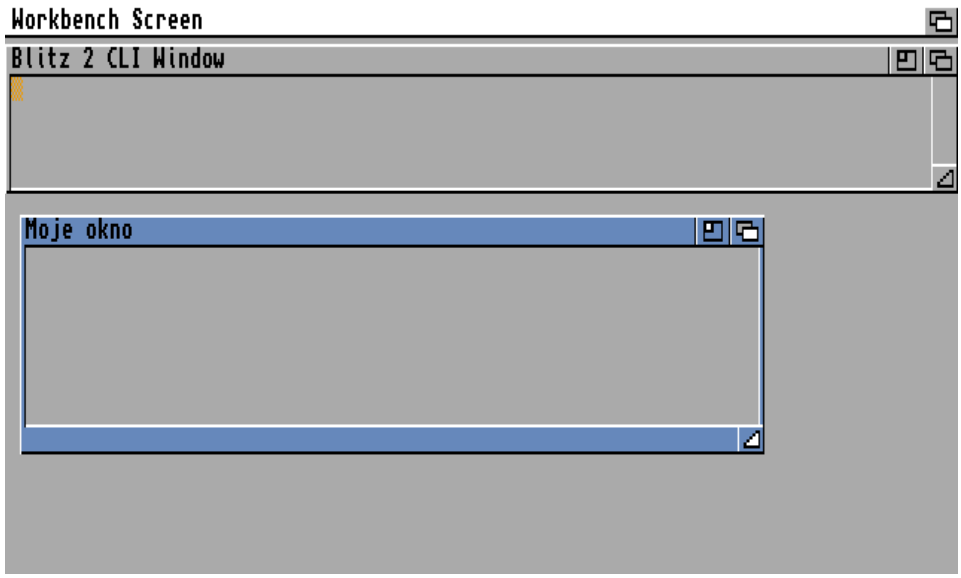
- **$0100** - okno będzie otwarte „na spodzie”
  - wszystkich innym aktualnie
  - dostępnych okien.

- **$0400** - powoduje, że elementy
  - umieszczane wewnątrz okna
  - nie będą mogły zasłonić ramki;
  - jest to wygodne, ale zajmuje
  - większą ilość pamięci.

Po ustaleniu, z jakich pozycji chcesz skorzystać, wartości należy zsumować, a uzyskaną wartość wpisać jako jeden argument. Dla ułatwienia zobacz kolejny przykład:

```
Window 1,0,0,200,35,$25,”Moje drugie okno”,1,2
```

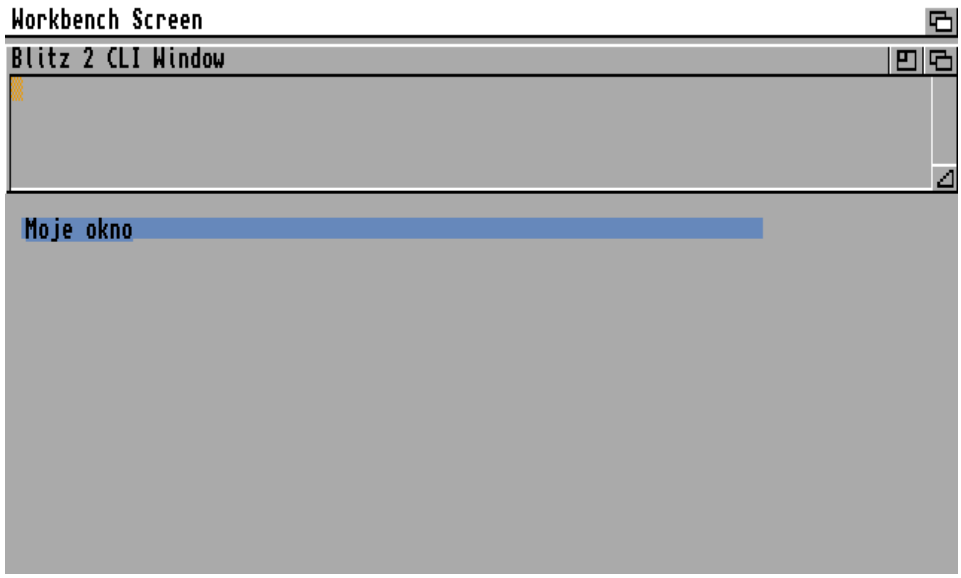
W powyższej linii mamy wartość „$25”, którą otrzymaliśmy z sumowania pierwszych pięciu pozycji z listy „flag”. Okno będzie teraz więc miało przycisk zmiany rozmiaru oraz standardowe pola w prawym górnym rogu. Wygląda następująco:



Ta sama zasada dotyczy innych wartości z listy. Zwróć uwagę, że na liście masz również opcje pozwalające określić sposób traktowania ramki okna oraz wyświetlania okna na aktywnym ekranie.

Na końcu linii z poleceniem WINDOW dodajemy dwa dodatkowe argumenty, które ustawiają kolorystykę okna. Pierwszy argument określa kolor, za pomocą którego wyświetlany będzie tytuł okna. Druga liczba to numer koloru używanego do rysowania zewnętrznej ramki okna. Dzięki temu możemy dostosować wygląd okna do palety barw na używanym ekranie, a także specyficznych funkcji naszego programu.

Zobacz jak bardzo nietypowo może wyglądać oto samo kno, gdy włączymy „flagę” o nazwie BORDERLESS :



Teraz nawet trudno zorientować się, jakie są jego rozmiary i dokładna pozycja. Daje to dodatkowe możliwości projektowania interfejsu Twojego progrramu.

Jeśli otworzysz kilka okien, przed wykonaniem operacji powinieneś określić, z którego chcesz skorzystać. W przeciwnym razie możesz wywołać nieprzewidziane rezultaty lub program zostanie zatrzymany z powodu wystąpienia błędu. Pamiętaj, że program może być obsługiwany przez użytkownika w różny sposób i musisz przewidzieć tę swobodę pisząc program.

Aby ustawić okno o określonym numerze jako aktualne wystarczy użyć polecenia USE WINDOW, podając obok numer, czyli przykładowo tak:

```
Use Window 1
```

Zwróć uwagę, że niektóre okna powinny mieć przypisane swoje własne minimalne i maksymalne rozmiary. Ma to związek przede wszystkim w ich zawartością lub ułożeniem, gdy zastosujesz własny ekran o ściśle określonych parametrach. Aby ograniczyć możliwość zmiany wielkości należy skorzystać z kolejnego kolecenia o nazwie SIZELIMITS. Oto jego sposób użycia:

```
SizeLimits 50,30,200,50
```

Najpierw podajemy minimalny rozmiar w poziomie i pionie, a następnie maksymalny – według tej samej kolejności. Linia ta powinna może znaleźć się przed poleceniem powodującym otwarcie nowego okna. Dopóki nie wprowadzisz kolejnej linii modyfikującej limity, będą one obowiązywać.

Możesz również umieścić słowo SIZELIMITS po otwarciu okna, ale wtedy każde nowe okno będzie zachowywało te same parametry. Oczywiście w każdej chwili można to zmienić i wszystko zależy od sposobu działania Twojego programu, umieszczonych w nim pętli, instrukcji warunkowych i innych elementów.

Aby wykonać operację na konkretnym oknie, powinieneś najpierw je aktywować. W przeciwnym razie bieżącym obiektem może być inne okno i Twój program nie zadziała poprawnie. Na szczęście nie jest to skomplikowana operacja, bowiem wystarczy wpisać linię podobną do poniższej:

```
Activate 1
```

co oznacza aktywację okna o podanym numerze. Polecenie to nie ma żadnych dodatkowych argumentów.

Pamiętaj, że jeśli udostępnisz użytkownikowi te możliwości, okna nie powinny zmieniać swoich pozycji bez wyraźnej przyczyny. Z drugiej strony, gdy obiekty będą miały na stałe przypisane rozmiary, z pewnością natrafisz na konieczność modyfikacji parametrów. Teraz już wiesz jak to robić.

Okna możesz przesuwać po ekranie, a także zmieniać ich rozmiar. Służą do tego polecenia o poniższych nazwach:

- WMOVE (od ang. Window Move)
- WSIZE (od ang. Window Size).

Ich użycie omawiamy razem, bo oba wymagają podania tylko dwóch argumentów. W pierwszym przypadku będą one oznaczały pozycję na ekranie, w jakiej ma znaleźć się okno, a w drugim – żądaną szerokość i wysokość okna. W obu przypadkach liczby oznaczają punkty na ekranie.

W związku z tym właściwe linie z wymienionymi poleceniami będą wyglądały tak:

```
WMove 150,50
```

lub

```
WSize 200,100
```

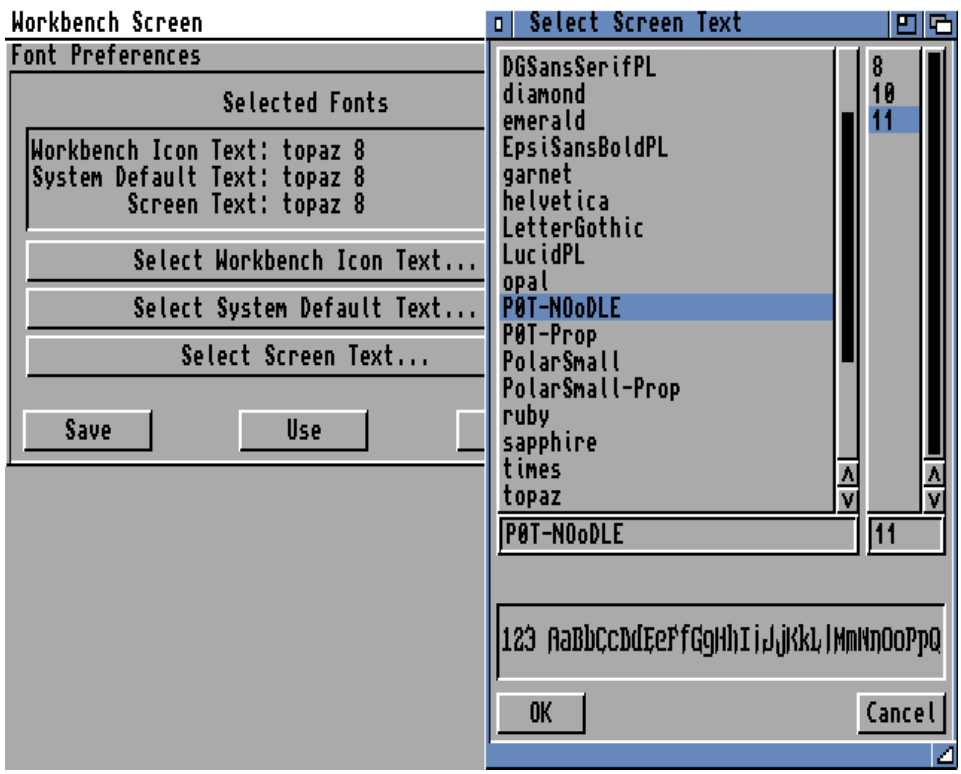
Oczywiście każde okno trzeba kiedyś zamknąć. Do tego z kolei służy polecenie FREE WINDOW, po którym podaj numer okna, przykładowo:

```
Free Window 1
```

Zwróć uwagę, że możliwość zamknięcia okna przez użytkownika powinna być przewidziana w programie w szczególny sposób, bowiem zmienia sposób obsługi wykonywanych czynności. Będziemy o tym mówić później.

TEKST I CZCIONKI

W oknie możesz wyświetlać tekst, aby przekazywać użytkownikowi różne komunikaty. Możliwe jest przy tym określenie czcionki jaka będzie używana. Gdy korzystasz z gotowych programów, z pewnością jesteś przyzwyczajony, że udostępniają one czcionki zapisane w systemowym katalogu „Fonts". Jeśli otworzysz okno wyboru czcionek, widać w nim poszczególne nazwy w porządku alfabetycznym. Na przykład tak jak w programie preferencyjnym „Font" znajdującym się w katalogu „Prefs" na dysku systemowym:



Pisząc własny program należy najpierw „wczytać” do niego czcionkę. Oczywiście odpowiednie pliku muszą znajdować się w katalogu „Fonts” i właśnie stamtąd będziesz mógł załadować dane. Innymi słowy, użytkownik musi wiedzieć, że ma zainstalować określoną czcionkę w systemie przed uruchomieniem Twojego programu, albo też musisz napisać skrypt instalacyjny (najlepiej przy użyciu programu „Installer”), który zrobi to automatycznie.

Wczytanie czcionki w Blitz Basicu realizujemy za pomocą polecenia LOADFONT. Obok trzeba podać:

- numer czcionki,
- nazwę kroju jaki chcesz użyć,
- rozmiar w pionie.

Cała linia może wyglądać następująco:

```
LoadFont 1,”opal.font”,12
```

Dodatkowo możliwe jest automatycznie ustalenie stylu czcionki za pomocą parametry na końcu linii. Należy wprowadzić liczbę oznaczającą:

- **2** - czcionkę pogrubioną,
- **4** - czcionkę pochyłą (kursywa),
- **1** - czcionkę podkreśloną.

W naszym przykładzie ładuje krój o nazwie „opal” w rozmiarze 12 pikseli. Czcionka ta należy do standardowego wyposażenia Workbencha, a więc możesz z niej korzystać bez potrzeby instalowania dodatkowego oprogramowania.

Po załadowaniu czcionki nic nie stoi na przeszkodzie, aby jej użyć w oknie. Do tego celu służy polecenie WINDOWFONT, któremu trzeba podać numer czcionki przypisany do kroju, na przykład:

```
WindowFont 1
```

Numer jest oczywiście zbieżny z liczbą wpisaną w linii z poprzednim słowem LOADFONT, tak więc możesz precyzyjnie określić zestaw znaków jakim będzie posługiwać się interfejs programu.

Teraz możesz już wypisywać tekst w oknie mając pewność, że nie zostanie użyta zbyt mała lub zbyt duża czcionka. Na początek ustaw kursor w określonej pozycji - zwykle jest to lewy górny róg okna, choć wszystko zależy od potrzeb. Aby to zrobić musisz wpisać linię z poleceniem WLOCATE, któremu podajemy współrzędne – poziomą i pionową.

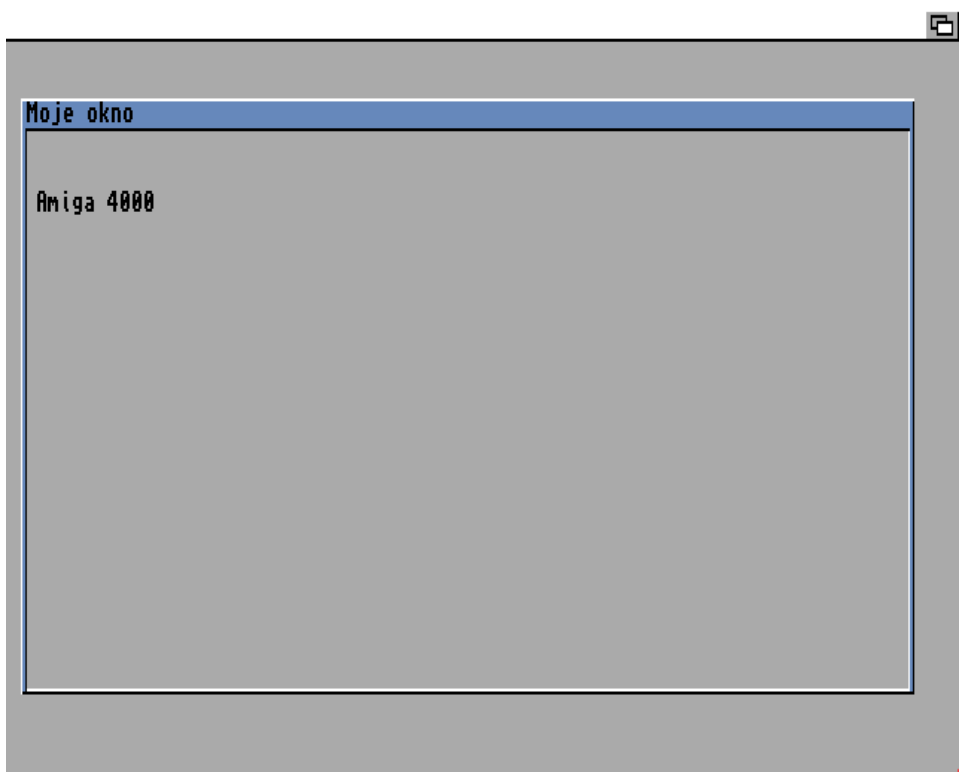
Pamiętaj, że pozycja cały czas jest określana przy pomocy punktów na ekranie, a nie linii tekstu. Dodatkowo każde okno ma własną pozycję kursora tekstu, dlatego zmiana w jednym oknie nie będzie miało wpływu na położenie kursora w innym okna.

Oto krótki program prezentujący te możliwości:

File - test4.bb2

```
Screen 0,13

Window 0,10,30,600,200,$10,"Moje okno",1,2
Activate 0

WLocate 6,20

a$="Amiga 4000"
Print a$

MouseWait
```

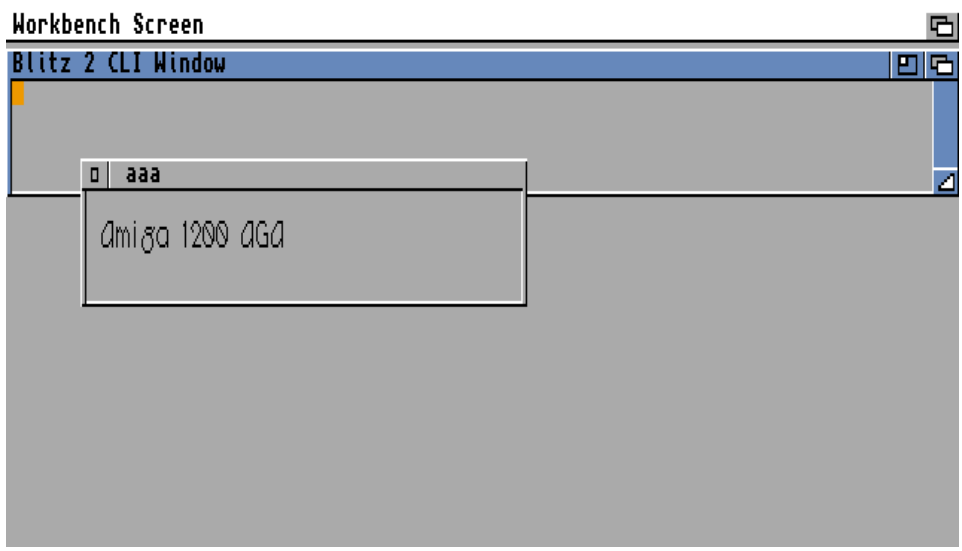
oraz jego efekt działania:



Kolejny przykład zmienia czcionkę i otwiera mniejsze okno posiadające przycisk zamknięcia. Spójrz na listing:

```
LoadFont IntuiFont#,Fontname.font$,Y size [,style]
FindScreen 0

Window 1,50,50,300,50,$8,"aaa",1,1
Use Window 1

LoadFont 1,"opal.font",12
WindowFont 1

WLocate 10,10
NPrint "Amiga 1200 AGA"

MouseWait
```

i gotowe okno, tym razem – nieaktywne, ale na ekranie Workbencha:



Zwróć uwagę jak wiele możliwości daje tylko kilka linii zawierających bardzo proste polecenia. Napisanie programu wyświetlającego tekst na ekranie Workbencha okazuje się pracą na kilka minut. Jest to oczywiście tylko podstawowy sposób wykorzystania systemowego interfejsu graficznego, ale warto podkreślić, że nie posługujemy się żadnymi nietypowymi funkcjami, a więc nasz program działa w zgodzie z systemem operacyjnym.

SPRAJTY

Następnym ważnym elementem, szczególnie podczas tworzenia gier, są tak zwane sprajty (ang. sprite). To obiekty graficzne, które można wykorzystać na przykład do animowania bohatera w grze. Są tworzone i poruszane na ekranie oddzielnie od pozostałym części oraz niezależnie od siebie. Jest to możliwe, bowiem obiekty te są obsługiwane bezpośrednio przez chipset Amigi. Oznacza to, że nie muszą być ręcznie usuwane, gdy chcemy wykonać ruch bohatera, ponadto sprajty nie niszczą, ani nie zakłócają grafiki bitmapowej. Za ich pomocą możemy tworzyć złożone efekty na ekranie.

Blitz Basic pozwala tworzyć sprajty w 2 lub 4 bitplanach, to znaczy mogą one zawierać 3 lub 15 kolorów. Podstawowa szerokość obiektu to 16 pikseli oraz dowolna ilość linii w pionie. Sprajty wymagają także ustawienia oddzielnej palety kolorów. Pamiętaj, że Amiga posiada sprzętową obsługę 8 sprajtów.

- Tworzenie sprajtów

Aby wyświetlić sprajta, najpierw należy wczytać grafikę za pomocą polecenia LOADSHAPE. Podany obok plik będzie traktowany jako „kształt" (ang. Shape) o określonym numerze. Na przykład poniższa linia:

```
LoadShape 1,"Worek/date/grafika.iff"
```

spowoduje wczytanie pliku „grafika.iff" i utworzenie z niego „kształtu" numer 1. Plik musi być zapisany w formacie IFF ILBM.

Następną czynnością jest przyporządkowanie załadowanego pliku do sprajta. Służy do tego polecenie GETASPRITE, które wymaga podania dwóch numerów, w kolejności:

- docelowego sprajta,
- źródłowego „kształtu".

Informacje te podkreślamy, bo można je łatwo pomylić. Jeśli więc z naszego „kształtu" o numerze 1 chcemy utworzyć sprajta o numerze „zerowym" należy wpisać poniższą linię:

```
GetaSprite 0,1
```

Od tego momentu sprajt jest utworzony w pamięci. Jest od niezależny od „kształtu", który można usunąć bez utraty informacji o sprajcie. Dzięki temu oszczędzimy pamięć. W tym celu wystarczy użyć polecenia FREE SHAPE podając obok numer „kształtu", czyli przykładowo:

```
Free Shape 1
```

- Wyświetlanie sprajtów

Gdy mamy gotowy sprajt możemy go wyświetlić na ekranie. Uzyskamy to a pomocą kolejnego słowa SHOWSPRITE, które ma bardziej rozbudowaną składnię.

Spójrz na przykład:

```
ShowSprite 0,50,30,1
```

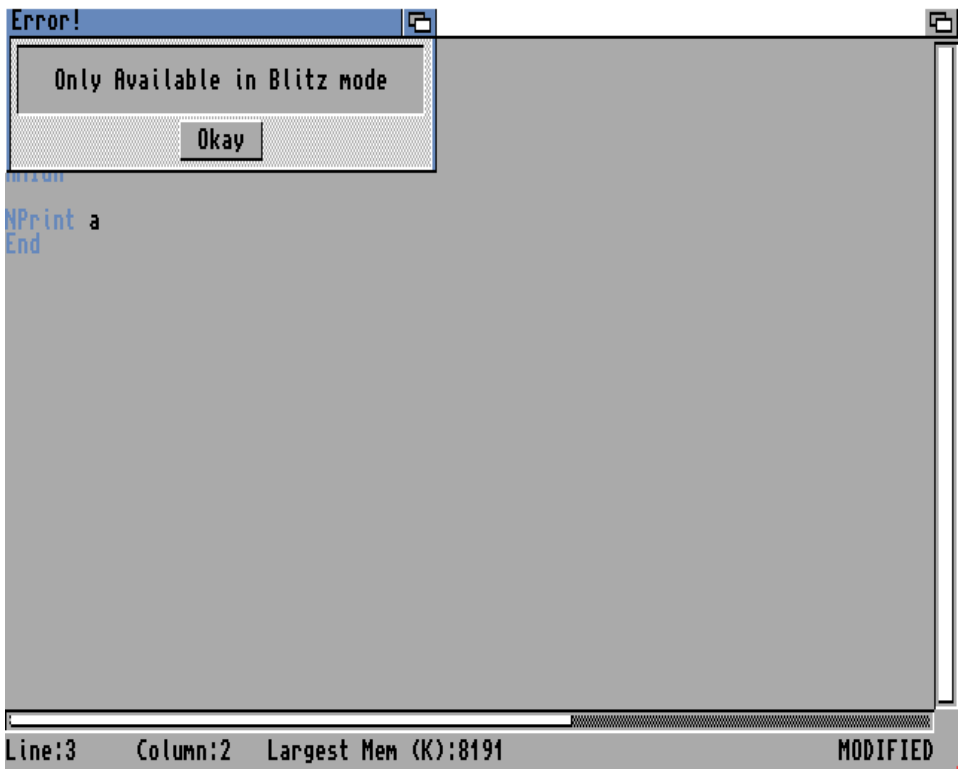
Pierwsza liczba to numer sprajta, który został utworzony. Następne dwie oznaczają współrzędne – poziomą i pionową, gdzie chcesz wyświetlić obiekt. Natomiast ostatnia wartość to tak zwany „kanał", do którego sprajt będzie przypisany. Kanałów jest 8, bowiem – jak już wspomnieliśmy - tyle Amiga obsługuje sprzętowo sprajtów. Wprowadzone współrzędne będą się odnosić do trybu wyświetlania Lowres.

Zwróć uwagę, że omawiane funkcje są możliwe do uruchomienia tylko w tak zwanym trybie „Blitz", kiedy wyłączamy system Amigi i mamy bezpośredni dostęp do układów specjalizowanych. Aby go uruchomić wystarczy wpisać nazwę:

```
BLITZ
```

w osobnej linii. Powrót do „normalnego" stanu uzyskujemy poprzez polecenie „AMIGA". Dla ułatwienia spójrz na ilustrację:

Więcej na temat obsługi „bezpośredniego” trybu pracy będziemy pisać później. W tej chwili musisz nauczyć się podstawowych poleceń i ich znaczenia. Pamiętaj jednak, że jeśli na ekranie pojawi się okno jak poniżej:



oznacza to, że musisz włączyć tryb „Blitz”.

- Wielkość sprajtów

Pisaliśmy wcześniej, że standardowa szerokość sprajtów to 16 pikseli. Jeśli chcesz używać w programie większych obiektów musisz zmienić tryb wyświetlania za pomocą polecenia SPRITEMODE. Obok trzeba podać następujące liczby:

- szerokość 16 pikseli	0
- szerokość 32 pikseli	1
- szerokość 64 pikseli	2

Gotowy obiekt możesz zapisać na dysku w oddzielnym pliku. DO tego celu użyj słowa SAVESPRITES wpisując zakres numerów sprajtów oraz nazwę pliku. Dla ułatwienia zobacz jak może to wyglądać:

```
SaveSprites 0,2,"Worek:date/moje_sprajty"
```

Dwie wartości na początku określają początkowy i końcowy numer sprajtów, które mają zostać zapisane w pliku „moje_sprajty". Oczywiście możesz też zapisać jeden obiekt, ale przechowywanie informacji o wielu sprajtach w jednym pliku oszczędza miejsca. Jest to szczególnie ważne, gdy chcesz uruchamiać swój program z dyskietki na mało rozbudowanych Amigach.

Zapisany plik wczytujemy do pamięci za pomocą analogicznego polecenia LOADSPRITES. Jego sposób użycia jest taki sam, również należy określić zakres numerów oraz nazwę pliku. Dla naszego pliku będzie to miało formę:

```
LoadSprites 0,2,"Worek:date/moje_sprajty"
```

Zwróć uwagę, że nie musisz wczytywać od razu wszystkich zapisanych pozycji, co pozwala z kolei za oszczędzić miejsce w pamięci komputera. Dowolny obiekt możesz również usunąć z pamięci za pomocą słowa FREESPRITE. Obok wpisz tylko numer sprajta.

- Zmiana kolejności wyświetlania

Ważną cechą sprajtów jest możliwość wyświetlania „przed" lub „za" grafiką bitmapową widoczną na ekranie. Umożliwia to polecenie INFRONT, któremu trzeba podać jeden z parzystych numerów kanału, do którego przypisany jest obiekt, na przykład:

```
InFront 2
```

Po wykonaniu powyższej linii sprajty umieszczone na kanałach o numerach równych lub wyższych od „drugiego" będą wyświetlane „za" grafiką. Pozostałe obiekty – na kanałach poniżej numeru 2 – będą natomiast pokazywane „przed" grafiką. W naszym wypadku będzie to więc wyglądało tak:

- kanały 0,1,2 - sprajty wyświetlane „na wierzchu"
- kanały 3, 4, 5, 6, 7 - sprajty wyświetlane „na tle"

Obsługa kolejności wyświetlania obiektów wygląda nieco inaczej w trybie o nazwie Dual Playfield. Jest to kolejny szeroki temat, dlatego będziemy o nim mówić później.

ANIMACJA

Blitz Basic umożliwia obsługę nie tylko plików graficznych IFF, ale także animacji, czyli popularnie zwanego formatu „ANIM". Trzeba tu dodać, że posiada on wiele odmian, natomiast w programie bezpośrednio możesz użyć tak zwanego ANIM-2, czyli „Long Delta Mode". Jest to format, z którego korzystają popularne programy graficzne, na przykład słynny „Deluxe Paint". Jeżeli chcesz więc skorzystać z animacji w Blitz Basicu, musisz sprawdzić w jakim formacie zostanie utworzony plik i w razie potrzeby wykonać konwersję lub zapisać materiał ponownie przy użyciu innych parametrów.

- Wczytywanie plików

Aby wczytać animację trzeba użyć polecenia LOADANIM, które wymaga podania numeru oraz nazwy pliku. Może to więc wyglądać następująco:

```
LoadAnim 1,"Praca:pliki/animacja.anim"
```

Odtwarzanie jest równie problem, bowiem rozpocznie się po wykonaniu linii podobnej do poniższej:

```
InitAnim 1
```

Obok należy podać ten sam numer, wcześniej musimy również otworzyć ekran w odpowiedniej rozdzielczości. Jak to zrobić piszemy w punkcie zatytułowanym „Ekrany".

Powyższe polecenie spowoduje jednak odtworzenie tylko dwóch pierwszych klatek pliku. Jeżeli chcesz kontynuować odtwarzanie trzeba skorzystać z kolejnego polecenia o nazwie NEXTFRAME. W tym przypadku również trzeba wpisać wyłącznie numer przypisany do animacji.

Zwróć uwagę, że dzięki temu zobaczysz kolejną klatkę, więc w celu płynnego odtwarzania musisz stworzyć dłuższą pętlę. Nie ma większego znaczenia w jakiej formie zostanie ona zapisana, wszystko zależy od pozostałych części Twojego listingu.

- Odczytywanie parametrów pliku

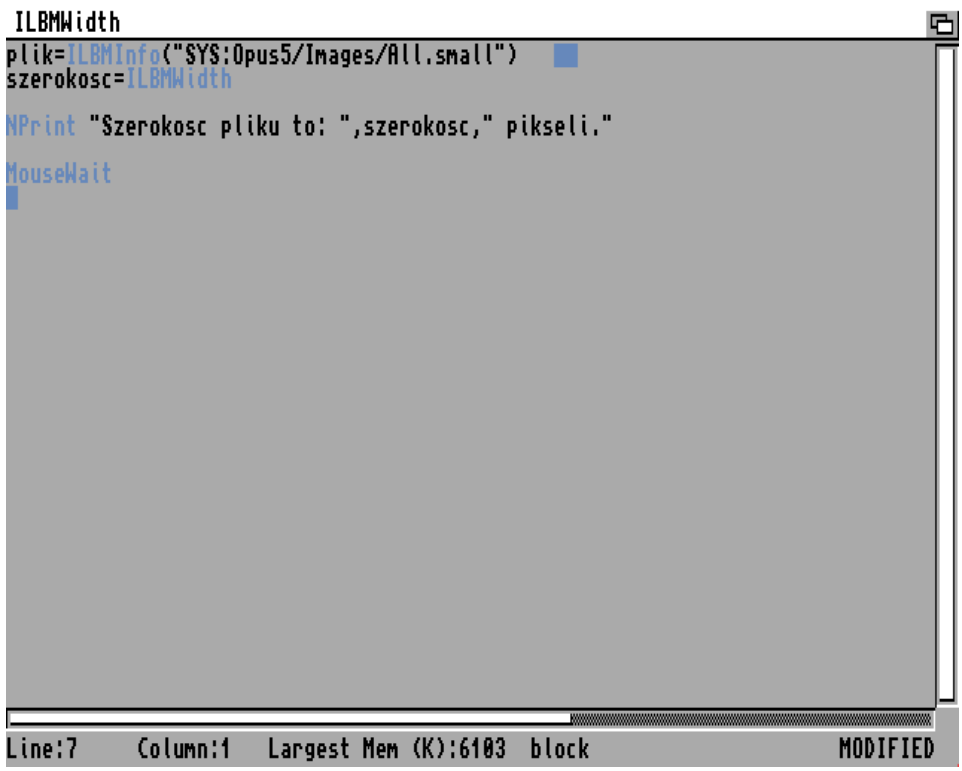
Gdy program natrafi na ostatnią klatkę pliku, automatycznie „przeskoczy" na początek, czyli ponownie do dwóch pierwszych klatek. Nie zawsze będziesz chciał uzyskać taki efekt, ale niekoniecznie musi się to wiązać z określaniem konkretnej długości animacji. Możesz odczytać informacje o zawartości pliku w formacie ANIM na kilka różnych sposobów.

Pierwszym jest użycie funkcji FRAMES(), dzięki czemu sprawdzisz ilość wszystkich klatek w pliku. W tym celu utwórz nową funkcję, na przykład tak:

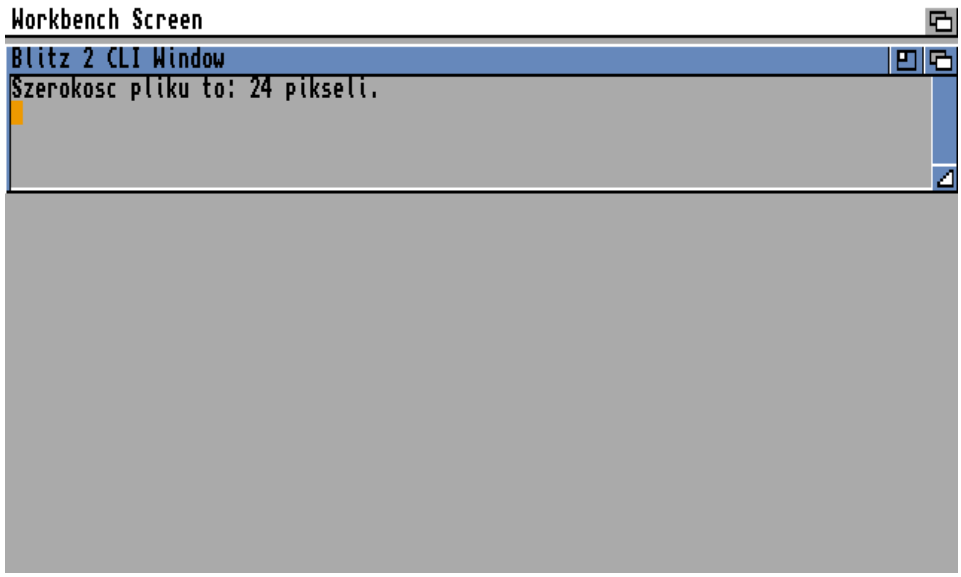
```
a=Frames(1)
NPrint a
```

Na tej podstawie bardzo łatwo możesz napisać program sprawdzający ilość klatek zapisanych w pliku i odtwarzający konkretny zakres, wprowadzony przez użytkownika.

Kolejna możliwość to użycie grupy funkcji o nazwach rozpoczynających się od symbolu „ILBM". Dzięki funkcji ILBMINFO() możesz odczytać szczegółowe informacje o pliku. Oto przykładowy listing:



i efekt działania:



Jak widać, pierwsza funkcja odczytuje informacje o pliku, a druga podaje konkretne parametry bez potrzeby wpisywania żadnego parametru. Inne cechy pliku możliwe są do odczytania osobno, należy jednak skorzystać z jednej z poniższych funkcji:

- ILBMWIDTH(),
- ILBMHEIGHT(),
- ILBMDEPTH().

Sposób użycia jest analogiczny. Dzięki nim sprawdzisz szerokość i wysokość grafiki składającej się na animację oraz „głębokość”, czyli ilość zastosowanych bitplanów.

- Tryb wyświetlania

Ostatnią opcją jest odczytanie trybu graficznego animacji, co może być szczególnie ważne, jeśli chcesz odtwarzać plik nie ma osobnym oknie Workbencha, lecz własnym ekranie. Będziesz musiał go otworzyć w określonym trybie wyświetlania oraz różnych ilościach kolorów.

Aby było to możliwe automatycznie użyj funkcji ILBMVIEWMODE() przed uruchomieniem odtwarzania. Pomiędzy tymi częściami wstaw instrukcje warunkowe otwierające odpowiedni ekran, w zależności od uzyskanych danych. Funkcja może przekazać następujące wartości:

- tryb Lores	0
- tryb Hires	32768
- tryb Interlace	4
- tryb Extra Half-Brite (EHB)	128
- tryb Hold-And-Modify (HAM)	2048

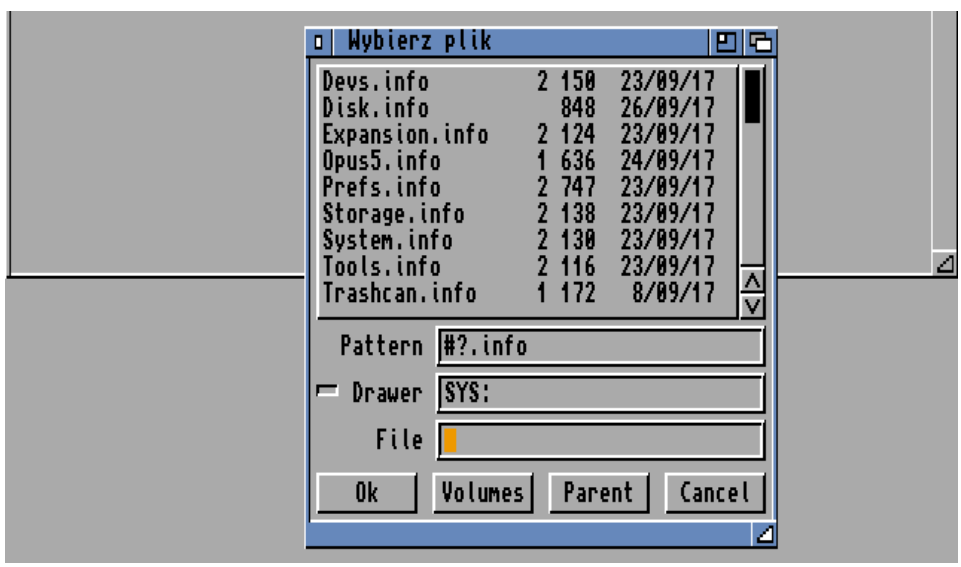
Według tych liczb - lub ich kombinacji, na przykład Hires oraz Interlace, należy otworzyć ekran, aby animacja była odtwarzana poprawnie.

OKNA WYBORU

Z omówionym wcześniej menu górnym związane są okna wyboru plików. W prawie każdym programie istnieje możliwość odczytywania i zapisywania plików, a więc Twój program też nie może być ich pozbawiony. Najprostszym sposobem utworzenia takiego okna jest skorzystanie z funkcji AmigaDOS, a konkretnie polecenia REQUESTFILE. Aby wyświetlić okno wyboru plików trzeba ustawić podstawowe opcje. Spójrz na przykład:

```
requestfile DRAWER="SYS:" TITLE=„Wybierz plik"
PATTERN="#?.info"
```

W rezultacie działania tej linii zobaczysz następujące okno wyboru:



Tytuł został podany w cudzysłowie zaraz po nazwie argumentu o nazwie TITLE. Argument PATTERN pozwala ustalić jakie pozycje będą wyświetlane w oknie. W naszym przypadku chcemy, aby widoczne były pliki z rozszerzeniem „.iff" zapisane na dysku systemowym, którego symbol („SYS:") wpisaliśmy jako argument DRAWER.

W ten sposób możemy spowodować, że użytkownik będzie mógł wpływać na pracę naszego programu. Na bazie wymienionych elementów można stworzyć wiele użytecznych elementów.

Drugim sposobem utworzenia okna wyboru jest napisanie krótkiego programu, który będzie niezależny od pozostałych elementów systemu operacyjnego. Nie wymaga to wielu wysiłków, bowiem wystarczy otworzyć okno, stworzyć listę plików w określonym katalogu, a następnie odczytać informacje o wyborze dokonanym przez użytkownika.

Taki sposób nie jest zalecany, bowiem Twój program będzie w tym momencie operował nietypowym elementem, zamiast wyświetlić zwykłe systemowe okno. Z drugiej jednak strony możesz stworzyć własny moduł do obsługi operacji dyskowych bez ograniczeń nakładanych z zewnątrz.

Naszym zdaniem powinieneś jednak trzymać się wytycznych dotyczących pisania programów, a więc używać standardowym okien jakie można znaleźć w innych programach. Będziemy więc korzystać z bibliotek systemowych.

W związku z tym musisz poznać funkcję FILEREQUEST$() oraz powiązana z nią grupę o nazwach rozpoczynających się symbolem „ASL”. Pochodzi on oczywiście on nazwy biblioteki „asl.library”, która jest elementem Workbencha.

Funkcja FILEREQUEST$() pozwala otworzyć standardowe okno wyboru plików typu na aktualnie używanym ekranie. Wykonywanie programu zostanie zatrzymane do czasu, aż użytkownik wskaże plik lub wybierze przycisk „Poniechaj” (ang. „Cancel”). Jeśli plik będzie wybrany, funkcja przekaże jego pełną nazwę w formie ciągu znaków. Gdy operacja zostanie anulowana, otrzymasz pusty ciąg.

Linia korzystająca z tej funkcji może wyglądać tak:

```
a$=FileRequest$("Wybierz plik",b$,c$)
```

Jak widać, należy ją przypisać do zmiennej tekstowej. Omawialiśmy do w rozdziale „Pierwszy program” i nie stanowi to żadnej nowości. W nawiasie musisz podać trzy parametry w następującej kolejności:

- tytuł okna (tutaj - „Wybierz plik”),
- ścieżka dostępu (tutaj - zmienna „b$”),
- nazwa pliku (tutaj – zmienna „c$”).

Tytułem może być dowolny ciąg znaków, ale powinien on wskazywać użytkownikowi, jaka opcja została wywołana. Może się bowiem zdarzyć, że wybrana będzie funkcja zapisywania pliku, zamiast wczytywania.

Dlatego okno powinno mieć precyzyjnie określony tytuł. Oczywiście możesz skorzystać ze zmiennej, zamiast wprowadzać konkretną treść.

Nazwę ścieżki dostępu wprowadzamy po to, aby mogła być ona automatycznie odczytana po wyświetleniu okna. Dzięki temu od razu widać listę plików z określonego katalogu. Użytkownik może ją później zmienić. Nie musisz jej podawać, ale często jest to wygodne, bowiem możliwe jest również naprowadzenie odbiorcy na katalog, w którym został zapisany Twój program.

Zwróć uwagę, że musi być to zmienna tekstowa, a jej maksymalna długość wynosi 160 znaków. Powinno być to wystarczające do większości zastosowań, choć maksymalna długość pliku w systemie Amigi wynosi tylko 107 znaków. Z tego względu Twój program nie powinien samodzielnie tworzyć długich dodatków do nazw czy rozszerzeń plików.

Nazwa pliku – podobnie jak w przypadku ścieżki – także jest ciągiem tekstowym, ale maksymalna długość w tym wypadku wynosi 64 znaki. Blitz Basic ma pod tym względem ograniczenie, lecz i tak przekracza ono dwukrotnie możliwości standardowego systemu plikowego „Fast File System” (31 znaków).

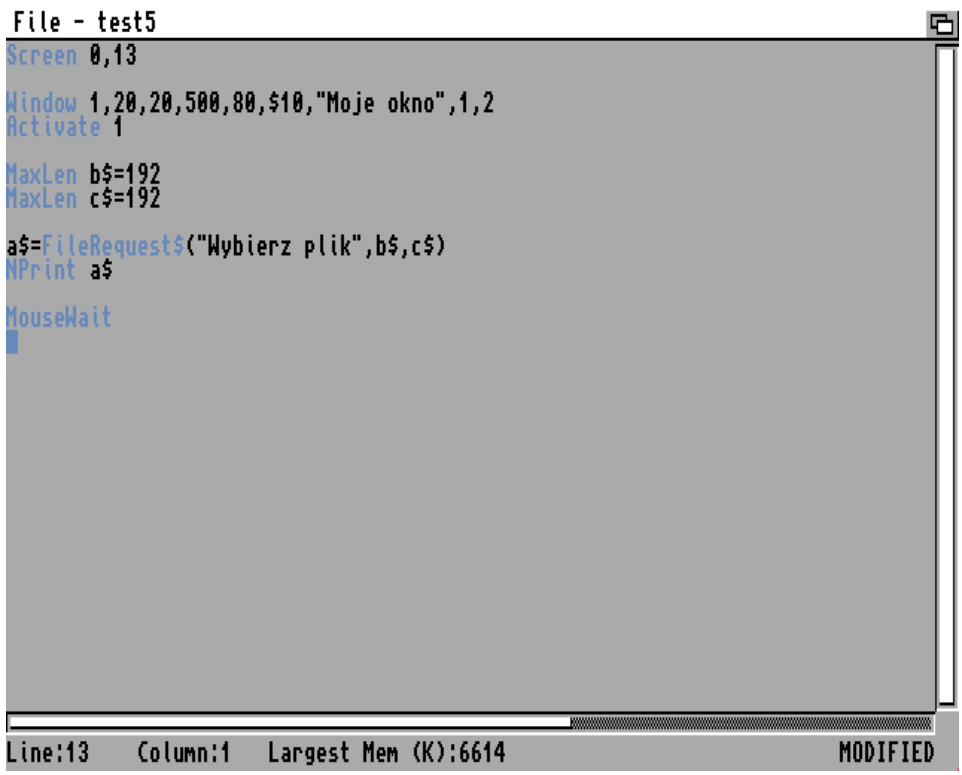
Przed wywołaniem funkcji FILEREQUEST$() musisz jeszcze określić maksymalny rozmiar zmiennej oznaczającej nazwę ścieżki dostępu, jak i nazwy pliku. U nas będą to pozycje „b$” i „c$”. Aby ustalić rozmiar skorzystaj z polecenia MAXLEN w taki sposób:

```
MaxLen b$=192
```

oraz

```
MaxLen c$=192
```

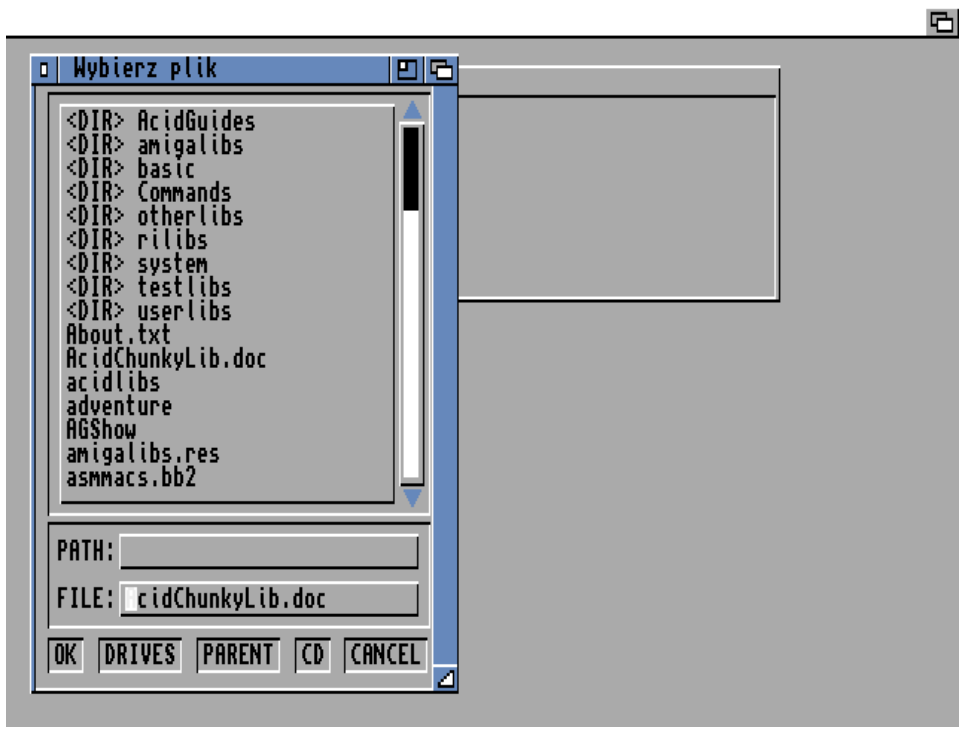
Rzecz jasna wartości mogą być inne, ale nie powinny być zbyt małe, w przeciwnym razie program nie będzie możliwy do uruchomienia. Polecenie MAXLEN służy do przydzielenia określonej ilości pamięci dla zmiennych, dlatego pamiętaj, aby użyć go przed ich deklaracją. W większym programie może to wyglądać na przykład tak:



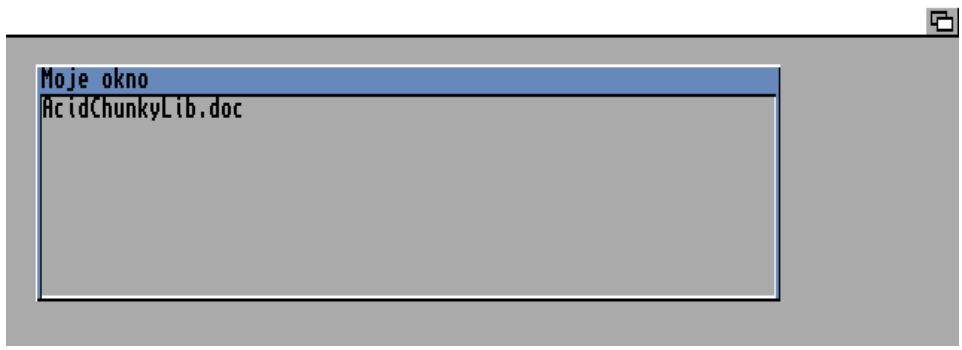
Jest to pierwsza części programu. Samo utworzenie funkcji nie wywoła okna wyboru, musisz jeszcze wprowadzić linię z poleceniem NPRINT podając obok zmienną przypisaną do funkcji. Przykładowo, tak jak na naszej ilustracji, czyli:

```
NPrint a$
```

Powyższą linię możesz wpisać bezpośrednio po deklaracji zmiennej „a$”. Zobacz jak wygląda rezultat działania takiego programu:



Na ilustracji wskazaliśmy już jeden z plików widocznych w oknie wyboru. Gdy wybierzesz przycisk „OK” nazwa zostanie wypisana w oknie widocznym „pod spodem”:



JOYSTICK

Pisząc grę nie sposób pominąć obsługę joysticka. Z kolei w programach użytkowych korzystających z własnych rozwiązań interfejsu użytkownika lub tworzących różne strefy obsługi przyda się umiejętność sterowania wskaźnikiem myszki.

Za pomocą funkcji JOYX() oraz JOYY() możemy odczytywać stan wychylenia joysticka. Obie należy używać w ten sam sposób, czyli – jak zwykle w przypadku funkcji – przypisać je do zmiennej liczbowej, na przykład:

```
a=JoyX(1)
```

Jako parametr podajemy port, do którego podłączony jest joystick (0 lub 1). W Amidze zwykle joystick jest używany w porcie numer 1, ale nie ma problemu, aby skorzystać z „zerowego" lub obu jednocześnie – dla dwóch manipulatorów.

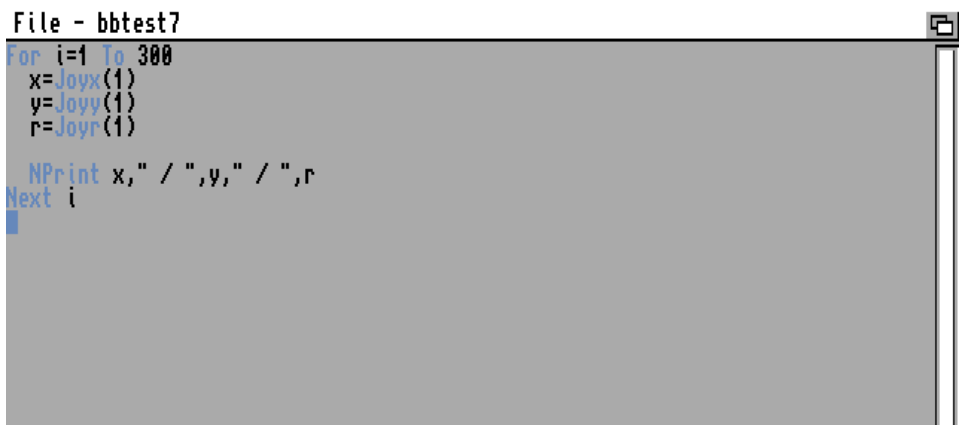
W rezultacie tej funkcji uzyskamy poniższe wartości:

- brak wychylenia	0	(JOYX, JOYY),
- kierunek w lewo	-1	(JOYX),
- kierunek w prawo	1	(JOYX),
- kierunek w górę	-1	(JOYY),
- kierunek w dół	1	(JOYY).

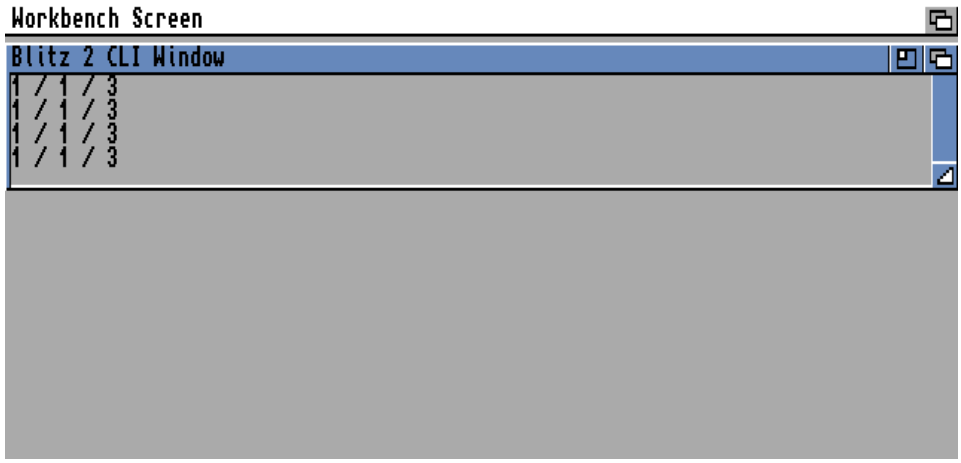
Stan joysticka możemy również odczytać w ramach jednej funkcji o nazwie JOYR(). Sposób użycia jest identyczny, ale tym razem otrzymamy wartości od 0 do 8, które oznaczają:

- kierunek w górę	0
- kierunek w górę i prawo	1
- kierunek w prawo	2
- kierunek w dół i prawo	3
- kierunek w dół	4
- kierunek w dół i lewo	5
- kierunek w lewo	6
- kierunek w górę i lewo	7
- brak wychylenia	8

W ten sposób możesz sprawdzać, czy użytkownik Twojego programu lub gry skorzystał z joysticka. Pamiętaj, że testowanie stanu musi znaleźć się w pętli, tak samo jak inne ważne elementy. Dla ułatwienia spójrz na kolejny prosty przykład w edytorze:



oraz efekt jego działania:



Oprócz kierunków ważne jest również użycie przycisku Fire. Za to odpowiada funkcja JOYB() - nazwa jest skrótem od ang. Joystick Button. Jako argument znowu podajemy numer portu, czyli:

```
b=JoyB(1)
```

ale tym razem w odpowiedzi należy się spodziewać innych wartości:

- brak naciśnięcia przycisku Fire 0
- naciśnięcie przycisku Fire 1

W Blitz Basicu możesz także programować kontrolę joypada dołączonego do konsoli CD32. Posiada on dodatkowe klawisze, jak na poniższym zdjęciu:

Obsługa dodatkowych przycisków wymaga zastosowania kolejnej funkcji o nazwie GAMEB (od ang. Game Button). Jej użycie jest wciąż takie samo, otrzymasz natomiast wartości jak poniżej:

- Play/Pause		1
- Reverse		2
- Forward		4
- Green	(przycisk zielony)	8
- Yellow	(przycisk żółty)	16
- Red	(przycisk czerwony)	32
- Blue	(przycisk niebieski)	64

Pamiętaj, że wychylenia kierunków oznaczane za pomocą poprzednich funkcji, a GAMEB służy tylko do sprawdzania przycisków niedostępnych na zwykłym joysticku. Do Amigi CD32 można podłączać różne manipulatory, dlatego jeżeli Twój program będzie korzystał ze wszystkich możliwych funkcji, może to zawęzić grono odbiorców. Gracz będzie musiał korzystać z oryginalnego joypada lub zamiennika posiadającego dodatkowe klawisze. Dlatego sugerujemy, abyś umożliwił wywoływanie tych samych opcji także za pomocą klawiatury lub myszki – wtedy program będzie działał uniwersalnie na każdej Amidze.

MYSZ

Obsługa myszki jest bardziej rozbudowana, bowiem siłą rzeczy dotyczy pozycji wskaźnika widocznego na ekranie. Z pewnością widziałeś programy, w których zmienia on swój wygląd. Jest to także stosowane w grach, choć rzadziej, bo zwykle opierają się one na obsłudze za pomocą joysticka.

- Śledzenie ruchu

Pierwszą rzeczą jest określenie, czy program ma śledzić ruch myszki. Służy do tego polecenie MOUSE, które pozwala włączyć obsługę:

```
Mouse On
```

lub ją wyłączyć:

```
Mouse Off
```

Standardowo wskaźnik może być przemieszczany na całym ekranie, co dla trybu Lowres daje obszar o współrzędnych początkowych 0,0 i końcowych 320,00 punktów. W razie potrzeby możesz go zawęzić, wystarczy tylko użyć polecenia MOUSEAREA. Przyjmuje od cztery argumenty, w kolejności:

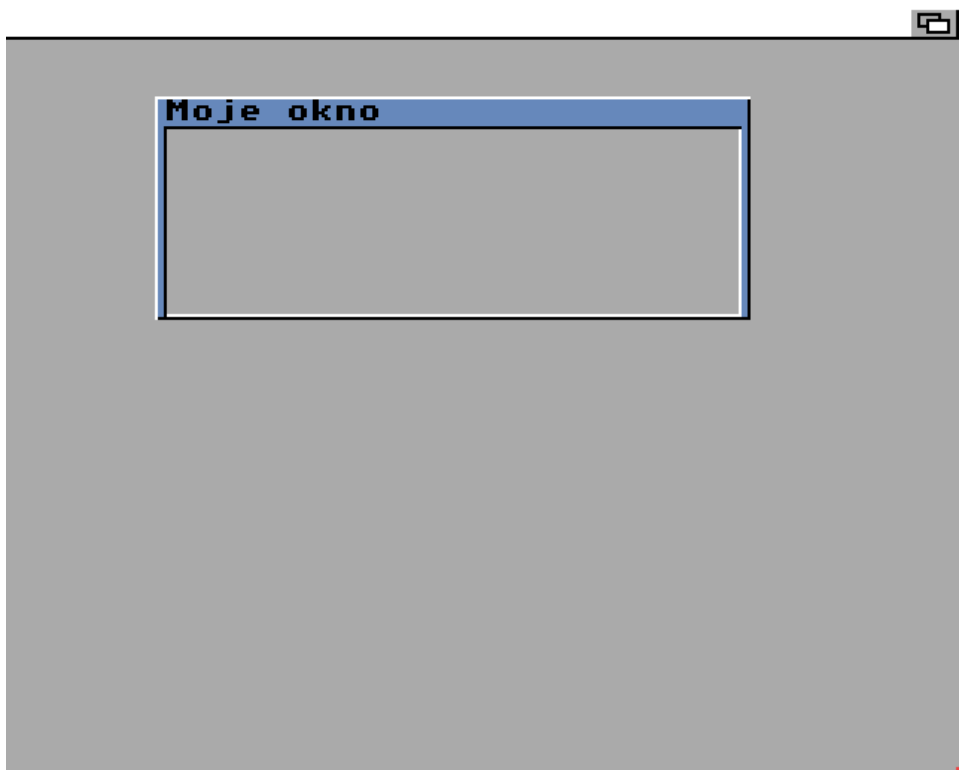
- początkowa współrzędna pozioma,
- początkowa współrzędna pionowa,

- końcowa współrzędna pozioma,
- końcowa współrzędna pionowa.

Przykładowa linia może więc wyglądać tak:

```
MouseArea 50,30,200,75
```

Teraz użytkownik będzie mógł przemieszczać wskaźnik w mniejszym zakresie, czyli w kwadracie o współrzędnych, które przedstawia poniższe okno:



- Pozycja wskaźnika

Działanie programu możesz uzależnić od pozycji wskaźnika myszki. Taką możliwość dają dwie funkcje – MOUSEX() oraz MOUSEY(). Odczytują one aktualne współrzędne i tak jak poprzednio, powinny być umieszczone w pętli, aby program mógł odpowiednio reagować na zmianę pozycji.

Powyższe funkcje są możliwe do uruchomienia tylko w tak zwanym trybie „Blitz", który charakteryzuje się wyłączeniem systemu operacyjnego. Masz wtedy bezpośredni dostęp do układów Amigi, można więc powiedzieć w skrócie, że jest „niedosowy" sposób pisania programów.

Aby uruchomić tryb „Blitz" wystarczy wpisać tę nazwę w osobnej linii. Na przykład tak jak w naszym prostym listingu:

Ekran zostanie wygaszony, trzeba więc bardzo ostrożnie pisać programy wykorzystujące tę funkcję. Powrót do „zwykłego" trybu działania jest realizowane poprzez wpisanie słowa „Amiga", bowiem tak właśnie nazywa się drugi tryb. I to widać na naszej ilustracji. O tym, jak obsługiwać działanie komputera w trybie „Blitz" będziemy pisać później, w tej chwili ważne jest, abyś był świadomy istnienia wyminionych dwóch trybów.

Zwróć uwagę, że stan wskaźnika jest aktualizowany 50 razy na sekundę, ale więc pomiary będą bardzo precyzyjne i będą często się zmieniał. To także należy przewidzieć piszą program.

Jeśli chcesz, aby zmiany były dokonywane przy przemieszczaniu określonych stref na ekranie, powinieneś użyć kilku instrukcji warunkowych.

Blitz Basic daje również możliwość uzyskania pozycji myszki w stosunku do różnych elementów na ekranie. Pierwsza opcja to odczytanie pozycji przy uwzględnieniu lewego górnego rogu bieżącego ekranu jako punkt wyjścia. Może się to przydać w sytuacji, gdy zmieniasz pozycje kilku otwartych ekranów. Do tego celu służą dwie funkcje o nazwach pochodzących od angielskich słów Screen Mouse:

- SMOUSEX(),
- SMOUSEY().

Należy ich używać analogicznie do MOUSEX() i MOUSEY(), pamiętaj jednak, ze w zależności od pozycji ekranu możesz uzyskać inne wartości.

Kolejne funkcje dotyczą pozycji wskaźnika w aktualnie aktywnym oknie (ang. Window). Dlatego też mają nazwy WMOUSEX() i WMOUSEY(). Odczytane wartości określają pozycję w odniesieniu do lewego górnego rogu okna. Dodatkowo mogą one uwzględniać ramkę okna lub jedynie samą zawartość. Zależy to od parametrów okna, a konkretnie ustawionej flagi GIMMEZEROZERO.

Gdy nie będzie aktywna, podana wartość będzie wskazywać na lewy górny róg ramki okna. W przeciwnym wypadku będzie oznaczać lewy górny róg zawartości okna.

Więcej na temat parametrów piszemy w punkcie pod tytułem „Okna".

- Testowanie „zdarzeń"

Pozycja wskaźnika może być też odczytywana w momencie, gdy w oknie zajdzie tak zwane „zdarzenie" (ang. Event). Znowu należy użyć funkcji – w analogiczny sposób – ale tym razem mają one inne nazwy:

- EMOUSEX(),
- EMOUSEY().

Zwróć uwagę, że zdarzenia nie zachodzą ciągle, jak wcześniej 50 razy na sekundę, dlatego pętla zawierająca powyższe słowa będzie działała szybciej i musi sprawdzać wykonywane czynności w inny sposób. Możliwości te będziemy omawiać później.

- Szybkość wskaźnika

Poza samą pozycją wskaźnika myszki, może on być poruszany z różną szybkością. Tę cechę również możesz obsłużyć, bowiem do dyspozycji masz dwie kolejne funkcje:

- MOUSEXSPEED(),
- MOUSEYSPEED().

Pierwsza odpowiada za sprawdzanie ruchu w poziomie, druga – w pionie. W tym przypadku możesz uzyskać liczby dodatnie lub ujemne, w zależności od kierunku ruchu wskaźnika. Będą one wskazywać na następujące zmiany:

- ruch w lewo	wartość ujemna MOUSEXSPEED(),
- ruch w prawo	wartość dodatnia MOUSEXSPEED(),
- ruch w górę	wartość ujemna MOUSEYSPEED().,
- ruch w dół	wartość dodatnia MOUSEYSPEED().

Im wyższa wartość, tym większa szybkość ruchu. Pamiętaj, że wskaźnik może mieć różny wygląd, może również nie być widoczny na ekranie. Jak go zmienić piszemy w punkcie pod tytułem „Zmiana wyglądu wskaźnika”.

Powyższe funkcje będą odczytywały stan niezależnie od tego, czy ruch pozostawia zmiany na ekranie, czy też użytkownik nie może śledzić wskaźnika. Warto również podkreślić, że przed użyciem funkcji musisz zastosować linię:

Mouse On

W przeciwnym razie nie będą one reagować na zmiany.

- Obsługa przycisków

Testowanie stanu przycisków myszki należy wykonywać przy użyciu tej samej funkcji, co dla joysticka, czyli JOYB(). Wcześniej pisaliśmy, że przyjmuje ona wartość 0 lub 1. W przypadku myszki będą to poniższe liczby:

- naciśnięcie lewego przycisku myszki 1
- naciśnięcie prawego przycisku myszki 2
- naciśnięcie obu przycisków jednocześnie 3

Jak widać możemy to robić w różnych konfiguracjach, w zależności od potrzeb. Znowu wszystko powinno znaleźć się w pętli, na przykład takiej:

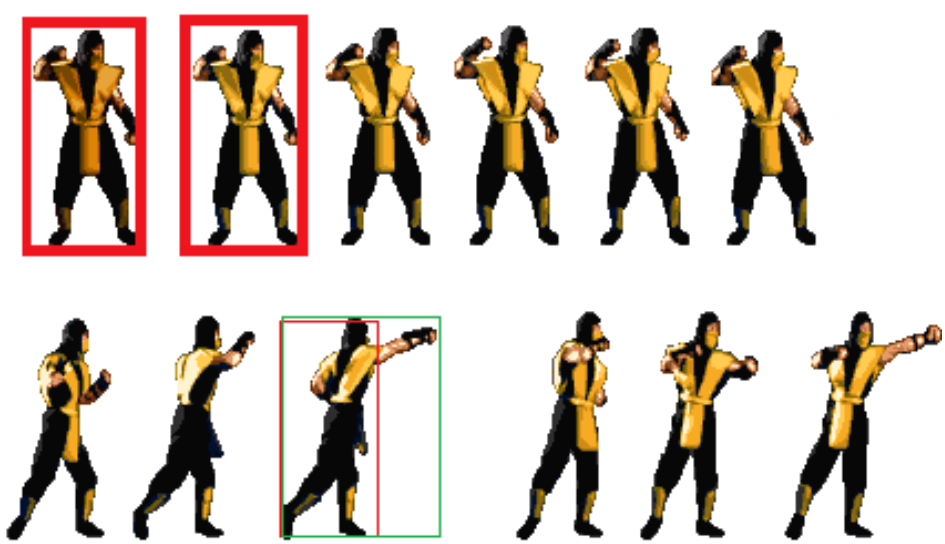
```
While Joyb(0)=0
    ...
Wend
```

Taki zapis sprawi,że program będzie oczekiwał na naciśnięcie dowolnego przycisku, aby zakończyć pętlę.

KOLIZJE

Gdy chcesz napisać własną grę, będziesz musiał przewidzieć interakcje pomiędzy obiektami. Twój bohater może napotkać wroga i należy pozbawić do części energii. Kontakt dwóch obiektów na ekranie nazywamy „kolizją”. Jest to sytuacja, w której figury użyte do ich tworzenia mają części wspólne.

Do wykrywania kolizji zwykle posługujemy się uproszczonymi wersjami obiektów w zależności od potrzeb. Im większe zastosujemy uproszczenie, tym szybsze będzie wykrycie kolizji, choć nieco mniej precyzyjne. Dla ułatwienia spójrz na przykład jak to może wyglądać na bazie bohatera popularnej gry:



Oczywiście gracz nie widzi technicznego aspektu wykrywania kolizji, a więc trzeba zadbać o właściwą reakcję programu. Jest to szczególnie ważne w momencie, gdy na ekranie mamy dużą ilość animowanych obiektów.

- Sprajty i grafika

W Blitz Basicu masz do dyspozycji różne sposoby wykrywania kolizji. Najprostszym jest wywołanie reakcji w momencie kontaktu sprajta z określonym kolorem grafiki bitmapowej na ekranie. Polega to na ustaleniu „niebezpiecznego" koloru, który wywoła reakcję jeśli sprajt osiągnie jego pozycję. Dzięki temu możliwe jest projektowanie stref, po których bohater może się poruszać, tak jak w typowej grze zręcznościowej.

Aby było to możliwe musisz użyć polecenia SETCOLL. Obok należy podać numer koloru oraz „głębokość" ekranu (ilość bitplanów) jaka będzie testowana podczas wykrycia kontaktu. Na przykład:

```
SetColl 2,3
```

spowoduje, że kolorem „zagrożenia" będzie drugi z cztech bitplanów, czyli 8 barw grafiki. Sposób ten pozwala dość dokładnie kontrolować kolizje, niezależnie od faktycznej ilości kolorów dostępnej na ekranie.

Podobnie możesz spowodować, aby niebezpiecznym kolorem był każdy nieparzysty w palecie. Innymi słowy będą to barwy o numerach 1,

3, 5,7 itd. W tym wypadku wystarczy użyć słowa SETCOLLODD bez żadnego argumentu.

W ramach operacji manipulowania kolorami dostępne jest jeszcze polecenie SETCOLLHI, dzięki któremu niebezpiecznymi kolorami będzie określony zakres barw – druga połowa palety. Tak się stanie po podaniu „głębokości", czyli na przykład:

```
SetCollHi 4
```

Powyższa linia oznacza, że bierzemy pod uwagę 4 bitplany, a więc 16 kolorów. Druga połowa palety w tym wypadku to barwy od numerów 8 do 15. Zwróć uwagę, że sposób podawania ilości kolorów jest podobny jak wcześniej, ale funkcja przyjmuje zakres barw, zamiast konkretnych pozycji.

Aby kolizja mogła być wykryta nie wystarczy ustalić sposobu działania programu. W pewnym momencie należy wydać polecenie, aby stan obiektów mógł być sprawdzony. Do tego celu służy słowo DOCOLL. Pamiętaj, że musisz je wprowadzić do programu dopiero po skorzystaniu z poprzednich poleceń, inaczej nie zadziała.

Samo polecenie DOCOLL powoduje jednak, że kolizja będzie mogła być przetestowana. Po nim należy określić, że chcemy sprawdzać kontakt pomiędzy sprajtem a grafiką. Musisz więc wpisać kolejną linię zawierająca funkcję PCOLL(), przykładowo:

```
k=PColl(1)
```

Liczba obok oznacza numer tak zwanego „kanału”, do którego przypisany może być konkretny obiekt. Dzięki temu wskażesz sprajta, którego chcesz wywołać. Więcej na temat kanałów piszemy w punkcie pod tytułem „Sprajty”.

Jeżeli kolizja zostanie wykryta, funkcja przyjmie wartość logiczną PRAWDA (czyli -1), w przeciwnym razie przekazany zostanie FAŁSZ (czyli 0 – zero). Pamiętaj, że oprócz nietypowego zastosowanie funkcja zachowuje się analogicznie do innych, o których pisaliśmy w oddzielnych częściach książki.

- Kolizje pomiędzy sprajtami

Oprócz testowania „bezpiecznych” i „niebezpiecznych” kolorów możesz sprawdzać kontakt pomiędzy różnymi sprajtami. W tym celu trzeba użyć funkcji SCOLL(), podobnej do PCOLL() omawianej w poprzednim punkcie. W tym przypadku jednak jako parametry podajemy dwa różne numery kanałów. Cel jest ten sam – określamy w ten sposób sprajty, które maja być brane pod uwagę podczas kolizji. Przykładowo może to wyglądać tak:

```
k=SColl(1,3)
```

Tak samo jak wcześniej, uzyskamy wartość logiczną PRAWDA lub FAŁSZ.

Możesz również sprawdzić, czy dwa obszary zajmowane przez sprajty nakładają się na siebie. W tym celu uzywamy polecenia SPRITESHIT wpisując obok:

- numer sprajta „pierwszego”,
- współrzędną poziomą,
- współrzędną pionową,
- numer sprajta „drugiego”,
- współrzędną poziomą,
- współrzędną pionową,

Jeśli sprajty nakładają się na siebie, funkcja przekaże wartość logiczną PRAWDA, w przeciwnym razie – FAŁSZ.

DŹWIĘK

Obsługa dźwięku to w wielu programach ważna kwestia. Domyślnie możesz korzystać z plików w formacie IFF, a mówiąc bardziej dokładnie – z odmiany o nazwie 8SVX. Jest to format służący do przechowywania 8-bitowych danych dźwiękowych, zarówno monofonicznych, jak i stereofonicznych.

Aby odtworzyć jakikolwiek dźwięk, najpierw musisz go wczytać do pamięci. Robimy to za pomocą polecenia LOADSOUND. Obok trzeba podać numer dźwięku oraz nazwę pliku, najlepiej wraz z pełną ścieżką dostępu. Na przykład tak:

```
LoadSound 1,"Pobrane:barrel"
```

Po tej operacji możesz wydać instrukcję, aby dźwięk został odtworzony. Wystarczy wpisać kolejne polecenie SOUND, ale teraz należy podać nie tylko numer, lecz także tak zwaną „maskę". Przyjmuje ona wartości od 1 do 15 i wskazuje na to, jakie kanały Amigi będą używane do odtwarzania. Przypomnijmy, że możemy używać 4 kanałów, po 2 dla każdego kanału stereo.

W związku z powyższym właściwa linia może wyglądać tak:

```
Sound 1,15
```

Polecenie to spowoduje odtwarzanie dźwięku numer 1 na wszystkich kanałach Amigi.

Możliwe jest stosowanie wszystkich kombinacji „włączenia” i „wyłączenia” kanałów, należy tylko użyć odpowiedniej liczby oznaczającej maskę. Oto lista możliwości:

Maska	Kanał 0	Kanał 1	Kanał 2	Kanał 3
1	włączony	wyłączony	wyłączony	wyłączony
2	wyłączony	włączony	wyłączony	wyłączony
3	włączony	włączony	wyłączony	wyłączony
4	wyłączony	wyłączony	włączony	wyłączony
5	włączony	wyłączony	włączony	wyłączony
6	wyłączony	włączony	włączony	wyłączony
7	włączony	włączony	włączony	wyłączony
8	wyłączony	wyłączony	wyłączony	włączony
9	włączony	wyłączony	wyłączony	włączony
10	wyłączony	włączony	wyłączony	włączony
11	włączony	włączony	wyłączony	włączony
12	wyłączony	wyłączony	włączony	włączony
13	włączony	wyłączony	włączony	włączony
14	wyłączony	włączony	włączony	włączony
15	włączony	włączony	włączony	włączony

Postępując według tabeli sprawisz, że dźwięk będzie odtwarzany na określonych kanałach. Dodatkowo możesz ustawić głośność każdego z nich. W tym celu na końcu linii dodaj wartości od 0 do 64. Musi być ich tyle, ile kanałów będzie używanych do odtwarzania. Na przykład, jeśli wprowadzisz linię:

```
Sound 0,10,32,16
```

używane będą dwa kanały – drugi i czwarty, dlatego dodaliśmy dwa argumenty – liczby 32 i 16. Nie jest to zbyt intuicyjny sposób obsługi, ale w praktyce wymaga jedynie sprawdzenia ilości kanałów i wprowadzenia określonej ilości dodatkowych argumentów.

Głośność możesz również zmieniać podczas odtwarzania. Dzięki temu możliwe jest stworzenie efektu wyciszania, podgłaśniania lub innych, w zależności od potrzeb. W tym celu wykorzystamy kolejne polecenie o nazwie VOLUME. Podobnie jak wcześniej musisz mu podać maskę oraz wartości oznaczające głośność.

Można więc powiedzieć, że wpisujemy to samo, co przy poleceniu SOUND, ale bez pierwszego argumentu. Zamiast:

```
Sound 0,10,32,16
```

użyj linii:

```
Volume 10,32,16
```

Zwróć uwagę, że tym razem nie zmieniasz pliku dźwiękowego, a jedynie modyfikujesz głośność kanałów, dlatego możesz to robić w różnych kombinacjach.

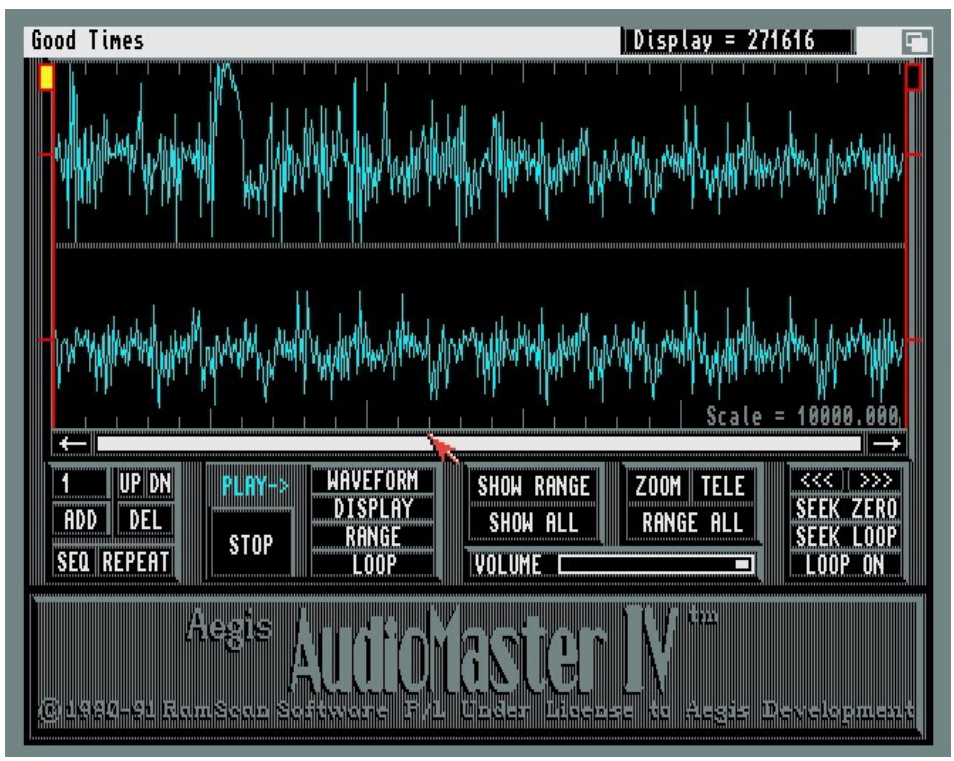
Dźwięk standardowo odtwarzany jest tylko raz. Możesz użyć pętli, aby go powtórzyć, ale bardziej wygodne jest wprowadzenie słowa LOOPSOUND. Jego argumenty są analogiczne do SOUND, tak więc jednocześnie z uruchomieniem odtwarzania „w pętli" możesz zmienić głośność. Oto przykład:

```
LoopSound 0,10,25,42
```

W pliku IFF 8SVX może być zapisany punkt startowy dla zapętlenia dźwięku. Można go utworzyć w edytorach próbek

dźwiękowych, przykładowo w słynnym „AudioMasterze”. Szczególnie polecamy jego czwartą odsłonę.

Pętle możemy tworzyć w sposób graficzny za pomocą znaczników, które standardowo umieszczone są na początku i końcu pliku. Wygląda to tak:



Nie zawsze będziesz miał taką możliwość, ponadto możesz chcieć zmienić sposób zapętlenia dźwięku. Jest to możliwe przy użyciu polecenia INITSOUND, którego składnia jest bardziej skomplikowana.

Oto jego argumenty, w kolejności wpisywania:

- numer dźwięku,
- długość dźwięku,
- wysokość dźwięku,
- miejsca zapętlenia dźwięku.

Tylko dwa pierwsze słowa są absolutnie konieczne do działania, dlatego możesz uprościć sobie pracę wpisując linię podobną do poniższej:

```
InitSound 0,32
```

Jeśli jednak chcesz wykorzystać wszystkie możliwości, warto wypróbować działanie pełnej wersji polecenia. Dodajmy, że wartość oznaczająca miejsce pętli musi mieścić się w przedziale od -128 do 127, tak więc będzie ona mocno uzależniona od wielkości pliku oraz szybkości odtwarzania.

Mówiliśmy, że podczas odtwarzania możliwa jest zmiana głośności. Podobnie możesz modyfikować wysokość dźwięku. Do tego służy następne polecenie o nazwie SETPERIOD. Obok niego należy podać numer dźwięku oraz wysokość, na przykład

```
SetPeriod 1,2
```

Próbki dźwiękowe mogą być zapisane w różny sposób, dlatego wartość drugiego argumentu należy określać w oparciu o wykonane testy na konkretnych plikach.

Użycie powyższych poleceń wiąże się z wczytaniem całej zawartości pliku do pamięci. Nie zawsze będzie to możliwe ze względu na ograniczenia maksymalnej ilości, ponadto długie pliki zapisane na dysku mogłyby być odtwarzane w częściach. Polecenie LOADSOUND pozwala wczytywać pliki o maksymalnej objętości 128 kilobajtów, co w dzisiejszych czasach nie jest wartością imponującą.

Aby posłuchać dłuższej ścieżki dźwiękowej możesz jednak wykorzystać większe pliki wczytywane z twardego dysku lub płyty CD. Trzeba jednak ładować je w częściach, dzięki czemu odtwarzanie jest możliwe także na komputerach wyposażonych w średnią ilość dostępnej pamięci.

Taka operacja jest możliwa za pomocą polecenia DISKPLAY. Używamy go podobnie jak SOUND, ale należy zamienić pierwszy argument na nazwę pliku. Właściwa kolejność jest więc następująca:

- nazwa pliku, najlepiej ze ścieżką dostępu,
- maska kanałów
- głośność kanałów.

Maskę i głośność należy ustawiać identycznie jak wcześniej, dlatego ilość argumentów będzie się zmieniać w zależności od ilości używanych kanałów audio.

Wyjaśnijmy sprawę nazwy pliku, która powinna być zapisana razem ze ścieżką, ale nie jest to zupełnie konieczne. Program będzie działał według zasad AmigaDOS, tak więc po wpisaniu samej nazwy pliku, będzie on poszukiwany w katalogu bieżącym – tym, w którym

zapisany jest Twój program. Na początku możesz dodać nazwę katalogu, czyli na przykład zapis:

```
DiskPlay „data/audio3.iff”,3,64,64
```

będzie oznaczał, że plik jest umieszczony w dodatkowym katalogu „data” oraz będzie odtwarzany na dwóch pierwszych kanałach przy zachowaniu pełnej głośności. Zasady wprowadzania ścieżek dostępu są typowe dla całego systemu Amigi. Jeśli chcesz uniezależnić program od katalogu, w którym zapiszesz gotowy program, stosuj pełne ścieżki dostępu, czyli rozpoczynając od nazwy dysku, przykładowo:

```
DiskPlay „Praca:pliki/data/audio3.iff”,3,64,64
```

Odtwarzanie muzyki z dysku nie jest zupełnie niezależne od ilości pamięci w komputerze. Zawartość jest odczytywana w częściach o określonych wielkościach. Standardową wartością jest 1024 bajty (nie kilobajty!), a więc minimalna porcja dla każdego modelu Amigi. Możesz ją zmienić za pomocą polecenia DISKBUFFER. Tym razem wystarczy podać nową wielkość „bufora”, czyli na przykład:

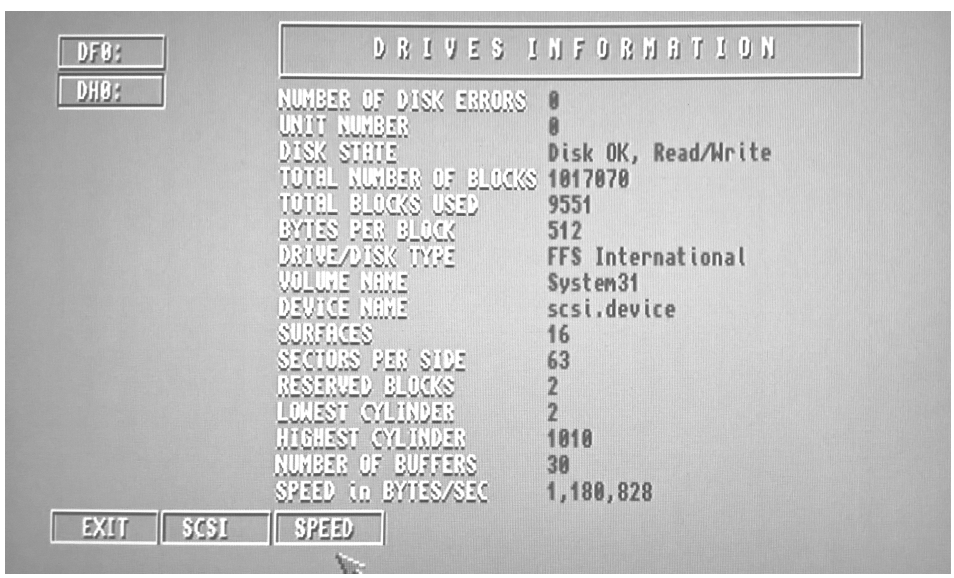
```
DiskBuffer 8192
```

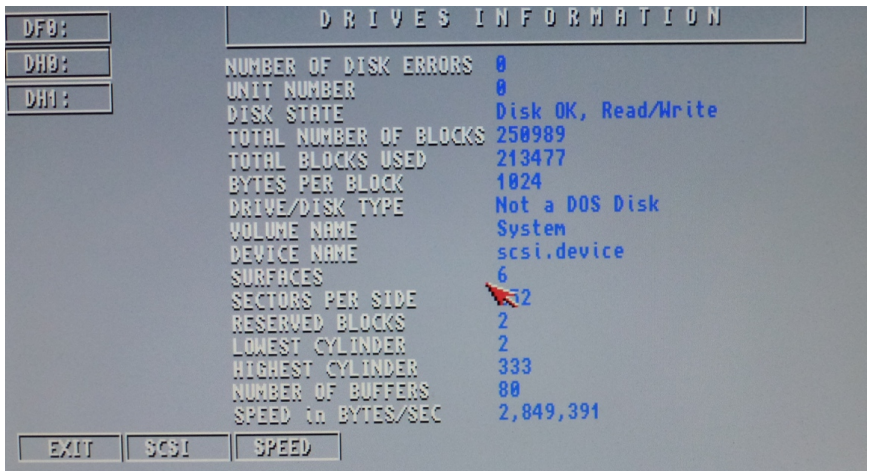
Teraz będzie on wynosił 8192 bajty, czyli 8 kilobajtów. Pamiętaj, że jego rozmiar musi być określany biorąc pod uwagę nie tylko ilość pamięci i szybkość procesora w komputerze. Dodatkowo – a może przede wszystkim – trzeba przewidywać szybkość nośnika, z którego odczytywany będzie plik oraz jakość zapisanego dźwięku.

Długiego pliku raczej nie będziesz odczytywał z dyskietki, w grę będzie więc wchodzić płyta CD/DVD, twardy dysk lub karta pamięci. Zarówno same nośniki, jak i różne typy urządzeń umożliwiają różną szybkość odczytywania danych. Dodatkowo wchodzą w grę kwestie związane z korzystanie z różnych systemów plikowych.

Twój program powinien być jak najbardziej uniwersalny, a więc nie może działać „na granicy ryzyka". Musisz obliczyć średnią prędkość odczytywania informacji w stosunku do jakości pliku, czyli częstotliwości pliku audio.

Dla ułatwienia spójrz na przykładowy pomiar w popularnym programie „SysInfo":





Powyższe ekrany pochodzą z różnych modeli Amigi, tak więc możesz zorientować się, jak może to wyglądać. Jakie to ma praktyczne znaczenie? Jeśli użyjesz zbyt małego bufora, dźwięk nie będzie odtwarzany płynne. Z kolei, za duża wartość spowoduje większe obciążenie pamięci.

W konsekwencji program będzie wymagał większej ilości pamięci, która będzie zajmowana w dużej części dla obsługi dźwięku ze szkodą dla pozostałych elementów.

Nasza propozycja to sprawdzenie sytuacji na Twoim komputerze, czyli określenie minimalnej wartości pozwalającej odtwarzać płynnie dźwięk, a następnie powiększenie tej liczby o ok. 20-30%, czyli założenie bezpiecznego „marginesu błędu”. Najlepiej jednak przesłać gotowy program lub tylko jego część do kilku osób, które sprawdzą jak to wygląda na ich konfiguracjach Amigi. Dzięki temu określić najlepszą wartość.

Amiga ma również możliwość aktywacji filtra dźwięku, tak zwanego dolnoprzepustowego. Jego zadaniem jest wydzielenie pewnego fragmentu częstotliwości z sygnału, leżącego poniżej częstotliwości granicznej. Jest to wykorzystywane w wielu programach i grach. Cechą charakterystyczną „włączenia" lub „wyłączenia" tej funkcji jest przygasanie diody Power w komputerze.

Aby włączyć filtr zastosuj linię:

```
Filter On
```

natomiast w celu jego wyłączeina:

```
Filter Off
```

Poza odtwarzaniem plików IFF 8SVX możesz również korzystać z tak zwanych „modułów muzycznych", czyli plików zawierających muzykę odgrywaną za pomocą próbek dźwięku – instrumentów. Szerzej powiemy o tym w następnym punkcie.

MUZYKA

Blitz Basic pozwala na odtwarzanie muzyki nie tylko zapisanych jako próbki dźwięku, lecz również w formie modułów muzycznych. Mówiąc w największym skrócie różnica polega na tym, że moduł zawiera listę nut oraz dodatkowych parametrów określających, w jaki sposób mają być odtwarzane próbki dźwięku na poszczególnych kanałach. Te próbki są traktowane jako „instrumenty", a więc muzyka jest tworzona w czasie rzeczywistym – podczas odtwarzania. Natomiast plik IFF 8SVX to próbka dźwiękowa, czyli zapis dźwięku, który jest odtwarzany podobnie jak z płyty CD.

W Blitz Basicu możesz używać modułów w formacie „MOD" oraz „MED". Ten pierwszy oznacza ogólny format 4-kanałowej muzyki, który zaczął być używany w programie „SoundTracker" na Amidze w latach '80-tych. Zwróć uwagę, że pliki tego typu mogą być nazywane dwojako – ich nazwy mogą posiadać rozszerzenie lub przedrostek „mod". Innymi słowy może to wyglądać tak:

```
muzyka.mod
```

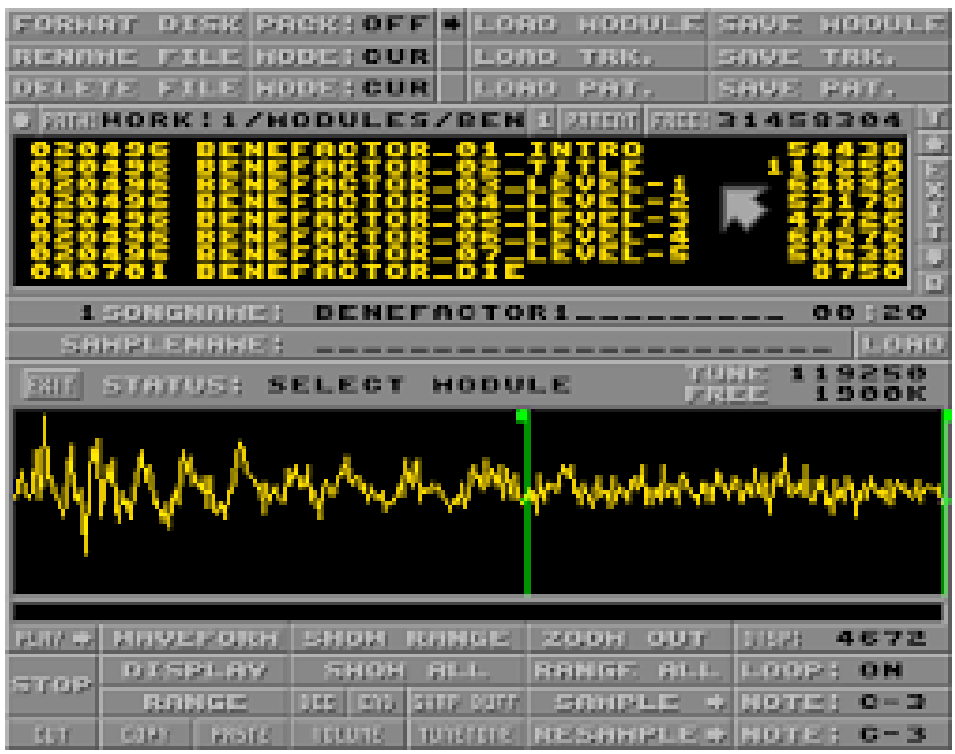
lub

```
mod.muzyka
```

Pierwszy sposób zapisu jest bardziej rozpowszechniony, bowiem komputery PC kiedyś nie potrafiły obsługiwać tak „dziwnej" nazwy, jak „mod.", zamiast „.mod". Stąd użytkownicy zmieniali nazwy, aby ujednolicić zapis i umożliwić bezproblemową obsługę na różnych

systemach operacyjnych. Jednak pierwotnie był stosowany przedrostek, a nie rozszerzenie i możesz spotkać się z różnymi nazwami plików. Sam program „ProTracker" pozwala wyświetlić pliki bez względu na nazwę.

Wygląda to podobnie do poniższej ilustracji:



Warto pamiętać o tych różnicach, bowiem korzystając ze starszych kolekcji oprogramowania możemy uruchomić programy nie pozwalające zmienić obsługi nazwy plików. Wtedy możemy być zmuszeni do ich zmiany, aby później użyć ich w Blitz Basicu.

Aby odtworzyć moduł typu MOD musimy skorzystać z polecenia LOADMODULE. W tym wypadku trzeba podać tylko numer modułu oraz nazwę pliku, przykładowo:

```
LoadModule 1,"muzyka.mod"
```

Oczywiście powyższa linia powoduje tylko wczytanie pliku do pamięci. Dalej należy uruchomić odtwarzanie za pomocą linii:

```
PlayModule 1
```

Liczba obok nazwy polecenia musi być oczywiście zbieżna z wpisana obok LOADMODULE. Zatrzymanie odtwarzania będzie realizowane po wykonaniu polecenia STOPMODULE, któremu tym razem nie podajemy żadnego numeru. Tylko jeden moduł może być odtwarzany jednocześnie i jest on w tym momencie przerywany. Dlatego cała linia wygląda tak:

```
StopModule
```

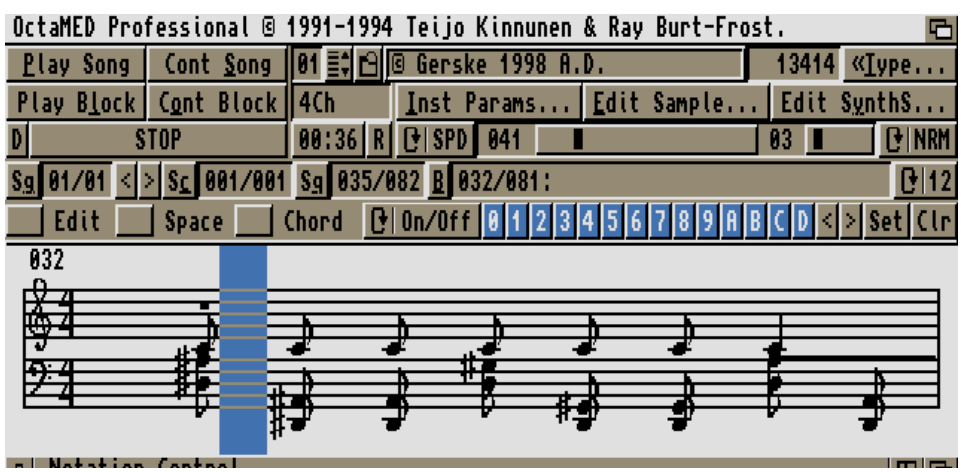
Pamiętaj, że wczytanie pliku zajmuje pamięć, niezależnie od tego, czy moduł jest odtwarzany, a możesz załadować więcej niż jeden plik naraz. Po włączeniu odtwarzania program może działać nieco wolniej, bowiem pewna część czasu procesora będzie używana na generowanie dźwięku. Jednak nie powinno być to znaczące obciążenie, więc nie musisz się tym bardzo przejmować.

Jednak w końcu będziesz potrzebował zwolnić pamięć dla innych operacji. Moduł możesz usunąć korzystając ze słowa FREE MODULE i tym razem musisz wpisać także numer modułu. Przykładowo:

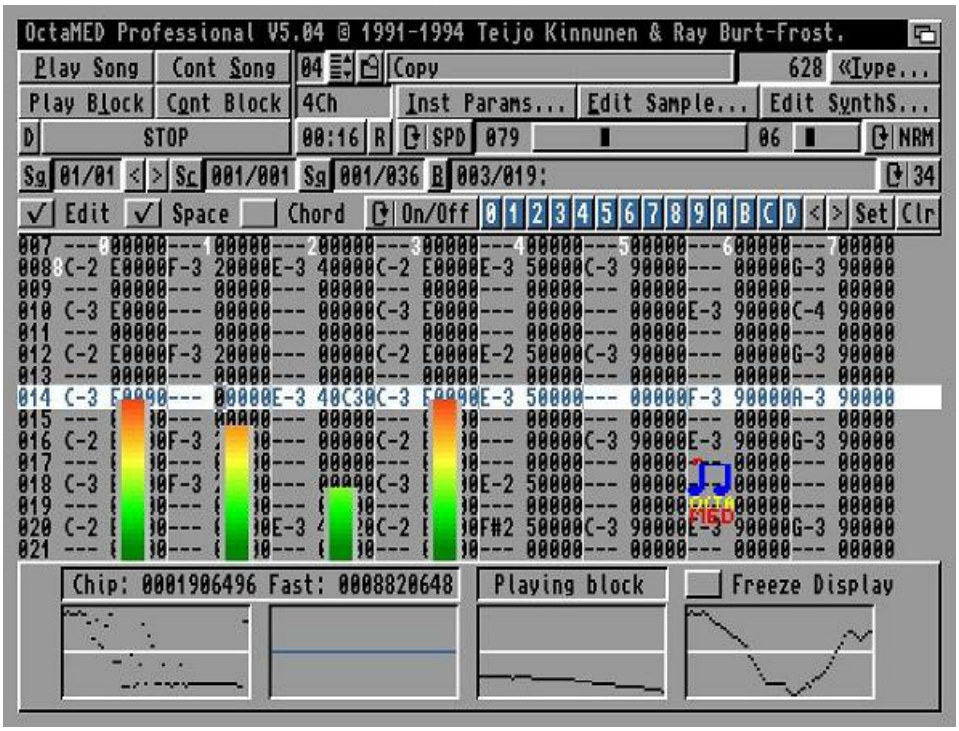
Free Module 1

Po czasie największej popularności modułów 4-kanałowych użytkownicy Amigi chcieli uzyskać większą ilość kanałów. Ponadto postulowano stworzenie programu, którzy korzysta z systemowego interfejsu użytkownika, bowiem stosowano już różne monitory i tryby wyświetlania. Niestety, najsłynniejszy „ProTracker", choć jest świetnym edytorem, nie pozwala ich zmieniać, co ogranicza użycie.

Z tych i innych względów, powstało wiele podobnych programów, operujących na tym samym formacie plików. Powstał także program „MED" oraz jego rozszerzona wersja o nazwie „OctaMED". Oba wykorzystują nowy format danych, który pozwala zapisać więcej informacji. Program umożliwia ponadto tworzenie modułów o większej ilości kanałów – na początku 8 (stąd nazwa rozpoczynająca się od „Octa"), a następnie aż do 64 ścieżek. Zobacz jak różni się „OctaMED" od swojego protoplasty:



lub



Moduły zapisane w formacie „MED” - określanego tak od nazwy programu - mają rozszerzenia „.med” i tym razem raczej nie spotkasz się z przedrostkiem. Możesz je obsługiwać w Blitz Basicu, ale musisz użyć innej grupy poleceń.

Podstawowy sposób działania jest podobny do wcześniejszego, bowiem należy użyć słowa LOADMEDMODULE, ale jego argumenty są analogiczne do LOADMODULE. Podajemy tu numer modułu oraz jego nazwę, na przykład:

```
LoadMedModule 1,"muzyka.med"
```

Pozwala to jednak uzyskać obsługę jedynie 4-kanałowych modułów w tym formacie. Jeśli chcesz uzyskać więcej będziesz zmuszony do skorzystania z funkcji zewnętrznych bibliotek systemowych. Szczegółowo będziemy o tym mówić później.

Odtworzenie pliku „.med" realizujemy poprzez wprowadzenie następującej linii:

```
PlayMed
```

Aby zatrzymać muzykę, wpisz po prostu:

```
StopMed
```

Jest to bardzo proste. Należy jednak pamiętać, że moduły MED mogą powodować większe obciążenie procesora, bowiem pozwalają uzyskać efekty niedostępne dla formatu MOD. Z tego względu pamiętaj, aby przetestować działanie na różnych konfiguracjach Amigi. Ponadto, jak już wspomnieliśmy, używanie modułów wykorzystujących 8 ścieżek lub więcej wymaga specjalnych środków.

W ramach plików z rozszerzeniem „.med" możesz spotkać się z różnymi rodzajami i nie każdy będzie możliwy do użycia w Twoim programie. Niektóre możesz spróbować zapisać w inny sposób w samym programie „OctaMED", ale konwersja nie zawsze będzie możliwa ze względu na używane funkcje.

Blitz Basic w zakresie obsługi formatu MED daje jednak większe możliwości niż tylko słuchanie muzyki. Przede wszystkim możesz rozpocząć odtwarzanie modułu na nowo, bez potrzeby ponownego wczytywania pliku. Wystarczy skorzystać ze polecenia STARTMEDMODULE, podając obok przypisany wcześniej numer. Oto kolejny przykład:

```
StartMedModule 1
```

W tym miejscu dotarliśmy do grupy poleceń, które nie mają swoich odpowiedników dla formatu MOD. Pierwszym jest JUMPMED, które pozwala odtworzyć muzykę od określonej części zwanej „patternem". Aby to lepiej zrozumieć musimy powiedzieć kilka słów o wewnętrznej budowie modułów muzycznych.

Każdy plik, niezależnie od tego, czy należy do formatu MOD czy MED, zawiera określoną ilość pozycji, w których zapisywane są nuty oraz odpowiadające im instrumenty i inne parametry. Jedna „strona" takiego zapisu jest nazywana właśnie „patternem". Może mieć ona różną długość, ale zawsze posiada podobną strukturę. Patterny są układane w określonej kolejności, mogą być również powtarzane i w ten sposób odtwarzany jest cały moduł.

W Blitz Basicu możesz odtworzyć moduł od określonego patternu za pomocą wspomnianego polecenia JUMPMED. Wystarczy podać obok numer, a więc cała lania wygląda tak:

```
JumpMed 5
```

Oczywiście zmiana dotyczy aktualnie odtwarzanego modułu, tak więc możesz wpływać na muzykę w trakcie słuchania. Może być to użyteczne w wielu przypadkach, na przykład podczas pisania dynamicznych gier, w których ścieżka dźwiękowa ma reagować na akcję widoczną na ekranie. Przykładem takiej produkcji może być seria bilardów, od „Pinball Dreams" począwszy do słynnego „Slam Tilta".

Zwróć uwagę, że tam poszczególne fragmenty muzyki bardzo często są przełączane lub powtarzane. Dzieje się to na tyle szybko, że nie byłoby to możliwe bez rozgrywki w twardego dysku, natomiast produkcje te działają także z dyskietek. Rozwiązaniem jest właśnie odpowiednia struktura modułu muzycznego, a następnie przełączanie „patternów". Jego odtwarzanie nie wymaga szybkiego procesora, nie zajmuje dużej ilości pamięci i umożliwia zapisanie różnych fragmentów muzyki. Pamiętaj, że różne patterny nie muszą być w żaden sposób ze sobą związane. Wszystko zależy od kolejności odtwarzania, dlatego jest to wręcz idealny format do tego typu eksperymentów.

Zwróć też uwagę, że obsługa plików „.med" wymaga obecności w systemie biblioteki o nazwie:

```
medplayer.library
```

Skąd ją pobrać? Najlepiej zdobyć instalacyjną wersję programu „MED" lub „OctaMED". Można też przejść do serwisu Aminet i pobrać plik o nazwie „medplayer_lib.lha". Oto bezpośredni odnośnik:

```
http://aminet.net/package/util/libs/medpla-
yer_lib.lha
```

Podczas odtwarzania modułu w formacie MOD, możesz łatwo zmienić jego głośność. Wystarczy skorzystać z polecenia SETMEDVOLUME podając obok tylko liczbę z zakresu od 0 do 64, przykładowo:

```
SetMedVolume 32
```

Powyższa linia ustawi głośność na 50% możliwej skali. W tym wypadku nie musisz wpisywać oddzielnych liczb dla poszczególnych kanałów audio, bowiem operacja jest wykonywana na wszystkich jednocześnie. Z jednej strony ogranicza to nieco użycie, z drugiej – dzięki temu możliwe jest stworzenie efektów typu wyciszanie czy podgłaśnianie, gdy chcesz płynnie rozpocząć lub zakończyć odtwarzanie muzyki.

Jeśli jednak chcesz modyfikować na bieżąco głośność różnych kanałów, możesz to zrobić za pomocą słowa SETMEDMASK. Należy mu podać maskę, czyli wartość oznaczającą określone kanały – tak jak omawialiśmy to wcześniej. Jeśli nie wiesz jak to robić przejdź do punktu zatytułowanego „Dźwięk”.

- Odczytywanie głośności

W niektórych sytuacjach możesz mieć potrzebę uzyskania informacji dotyczącej aktualnej głośności odtwarzanego modułu. W tym celu użyj polecenia GETMEDVOLUME. Pozwala ono odczytać głośność konkretnych kanałów muzycznych, a więc jego zastosowanie jest szersze niż tylko w stosunku do plików z rozszerzeniem „.mod" czy „.med".

Aby zorientować się jaka jest aktualna głośność danego kanału wpisz linię:

```
GetMedVolume 2
```

Liczba obok nazwy polecenia oznacza numer kanału. W odpowiedzi uzyskasz wartość odpowiadającą aktualnej głośności. Może Ci się to przydać w sytuacji, gdy chcesz dostosować siłę głosu kolejno odtwarzanych plików audio lub chcesz umożliwić kontrolę głośności przez użytkownika Twojego programu. Z pewnością można znaleźć wiele zastosowań tej funkcji.

- Inne formaty audio

Pamiętaj, że Amiga używa nie tylko formatów IFF 8SVX, MOD oraz MED. Powstało wiele programów pozwalających tworzyć muzykę, jednak ich wykorzystanie wymaga instalacji dodatkowego oprogramowania. Jeżeli chcesz odtwarzać inne rodzaje plików powinieneś skorzystać z zewnętrznych bibliotek, które można znaleźć między innymi w serwisie:

Aminet.net

Przykładem może być plik „xmplay-lib.lha” zawierający pliki pozwalające używać modułów programu „FastTracker”, czyli z rozszerzeniem „.xm”. Obsługę bibliotek omówimy później.

GENEROWANIE MOWY

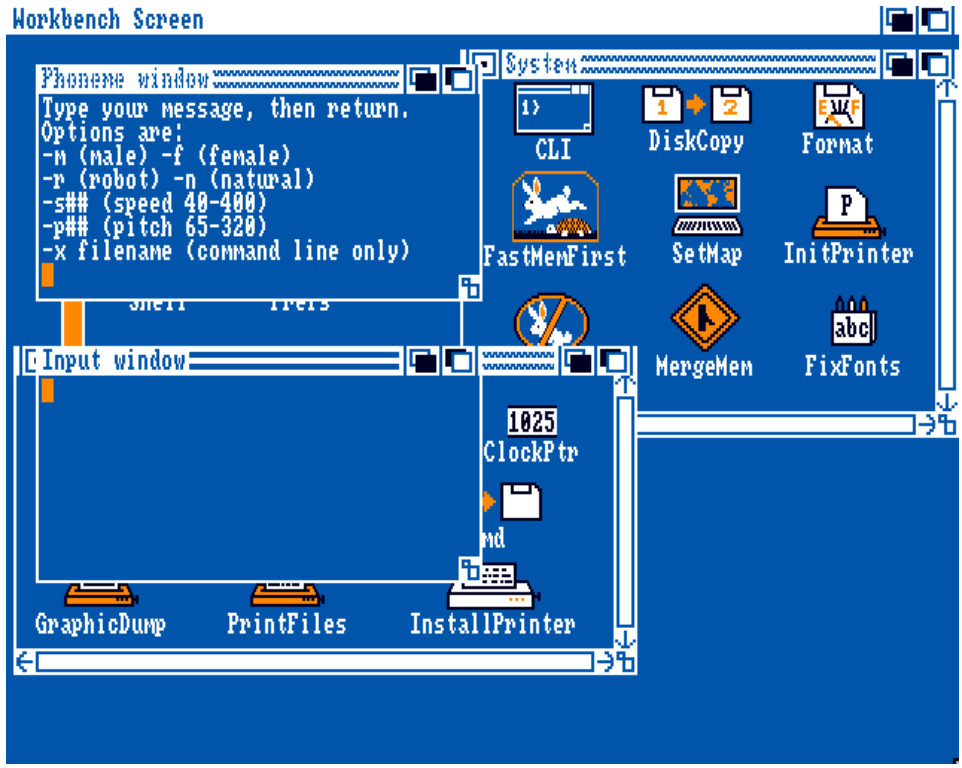
Pierwsza generacja Workbencha zawierała program generujący mowę. Można to przyrównać do nowoczesnych syntezatorów mowy, które można znaleźć w Internecie. W kolejnych edycjach systemu operacyjnego Amigi pliki odpowiadające za mowę zostały usunięte, wraz z prostym edytorem o nazwie „Say".

Przekształcanie tekstu na głos jest możliwe za pomocą urządzenia systemowego o nazwie „narrator" i wciąż jest jest to obsługiwane w Blitz Basicu.

Aby komputer „wypowiedział" tekst wystarczy użyć polecenia SPEAK, któremu podajemy zmienną tekstową. Przykładowo poniższe linie:

```
a$="Amiga i Atari"
Speak a$
```

spowodują wypowiedzenie słów zapisanych w ramach zmiennej „a$". Na tym nie kończą się możliwości, bowiem możemy zmieniać brzmienie „wirtualnego lektora". Zwróć uwagę, że możliwości takie posiada także program działający nawet na Workbenchu w wersji 1.3, czyli na systemie Amigi 500:



Podobne zmiany możemy uzyskać przy użyciu słowa SETVOICE. Obok trzeba jednak wpisać kilka nowych argumentów, w następującej kolejności:

- rate
- pitch
- expression
- płeć
- głośność
- częstotliwość.

Parametry te trudno zrozumieć bez przetestowania kilku różnych zestawów funkcji. Dlatego najlepiej zmieniać je metodą prób i błędów. Można też uruchomić Workbencha 1.3 oraz program „Say", aby zorientować się w możliwościach syntezatora mowy.

Pamiętaj, że nie jest to produkt, który może konkurować z nowoczesnymi odpowiednikami, ale „wypowiadane" słowa są na tyle wyraźne, że funkcja ta może mieć wiele zastosowań. Urządzenie „narrator" jest rzadko używane, ale z pewnością warto poznać jego możliwości. Warto podkreślić, że nie ma problemu ze stosowaniem różnych akcentów, a więc obsługiwanych jest wiele języków, w tym polski.

OPERACJE
ZAAWANSOWANE

OBSŁUGA PLIKÓW

W bardziej złożonych programach będziesz potrzebował uzyskać możliwość odczytywania i zapisywania danych nie tylko za pomocą opcji typu „Otwórz" i „Zapisz", ale również wewnętrznie. Może się to przydać na potrzeby zachowania danych tymczasowych lub innych, gdy chcesz skorzystać z informacji na dysku bez udziału i kontroli użytkownika.

Aby uzyskać dostęp do pliku musisz wskazać jego nazwę i ścieżkę dostępu. Ponadto – tak jak w wielu innych omawianych przypadkach – plik musi zostać przypisany do określonego numeru. To wszystko wykonujemy za pomocą jednej funkcji o nazwie OPENFILE(). Należy jej użyć w następujący sposób:

```
plik=OpenFile(1,"Worek:dane/dane3")
```

Powyższa linia „otworzy" plik o nazwie „dane3" i przyporządkuje mu numer 1. Mówiąc dokładniej, zostanie wykonana próba dostępu. Czy operacja się powiodła możesz sprawdzić odczytując wartość, jaka uzyskała zmienna „plik". Jeżeli wszystko jest w porządku przekazana zostanie logiczna PRAWDA (1), w przeciwnym razie FAŁSZ (0). Funkcja zachowuje się w sposób typowy i czytelny.

Należy tu zwrócić uwagę na kilka szczegółów. „Otwarty" plik nie ma z góry określonego przeznaczenia, a więc można będzie zarówno odczytywać z niego dane, jak i zapisywać w nim informacje. Jest to uniwersalny sposób, ale czasem może powodować problemy. Jeżeli bowiem podany plik nie istnieje na dysku. Zostanie automatycznie utworzony.

Co zrobić, gdy chcesz uzyskać dostęp do pliku na płycie CD lub dyskietce, ale bez tworzenia nowych danych? Musisz w podobny sposób skorzystać z innej funkcji – READFILE(). Tym razem nie będzie możliwe zapisywanie danych, ale za to nie będą też wykonywane inne operacje dyskowe. Jeśli plik nie istnieje funkcja przekaże wartość 0 (zero), czyli logiczny FAŁSZ i zakończy działanie.

Może być to bardzo przydatne nie tylko w sytuacji, gdy na dyskietce brakuje miejsca. Zwróć uwagę, że możesz mieć potrzebę odczytywania kilku plików po kolei, z różnych katalogów lub nośników. W momencie przekazania informacji o braku pliku możesz stworzyć procedurą nakazującą przełożyć dyskietkę, na przykład drugą dyskietkę z grą.

Ponadto w wielu przypadkach nośniki są tak zapisywane, aby były w jak największym stopniu zapełnione. Gdyby zawsze był tworzony nowy plik, mógłbyś wywołać systemowy komunikat o zapełnieniu nośnika. Poza tym użytkownik szybko przekładający dyskietki może to zrobić w trakcie wykonywania operacji i w konsekwencji doprowadzić do utraty struktury dyskietki, czyli tak zwanego braku „walidacji". Zazwyczaj skutkuje to informacją jak poniżej:

```
Disk not validated
```

To wszystko powoduje, że czasem lepiej użyć funkcji READFILE(), zamiast bardziej ogólnej OPENFILE().

- Odczytywanie danych

Informacje z pliku można odczytywać na kilka sposobów. Po pierwsze, możemy skorzystać z polecenia FILEINPUT. Obok należy podać numer „otwartego" pliku, czyli na przykład:

```
FileInput 0
```

Linia ta określi sposób użycia pliku, czyli w tym przypadku – odczyt danych. Nie powoduje jednak wykonania operacji, bowiem należy to zrobić za pomocą funkcji EDIT$(). Jako jej argument podajemy ilość znaków, które chcemy uzyskać. Oczywiście, tak jak w przypadku innych funkcji, należy ją przypisać do zmiennej, przykładowo tak:

```
dane[1]=Edit$(100)
```

Zastosowaliśmy tablicę, której element z indeksem 1 będzie zawierać 100 znaków. Więcej o tablicach przeczytasz w oddzielnym punkcie. Teraz zwróć uwagę, że sposób ten może się przydać w ramach różnych pętli. Spójrz na bardziej rozbudowany program:

```
If ReadFile(1,"Worek:dane/dane3")
     FileInput 1
          While NOT Eof(0)
                    For i=0 To 10
                         dane[i]=Edit$(100)

                    Next
          Wend
Endif
```

Jak widzisz, funkcję można bezpośrednio umieścić w linii ze słowem IF. Tym razem jednak, oprócz dwóch pętli oraz instrukcji warunkowej, użyliśmy wpisu:

```
EOF(0)
```

Jest to kolejna funkcja ułatwiająca pracę, bowiem pozwala określić koniec pliku (ang. End-Of-File). Wartość użyta w linii ze słowem WHILE wskazuje na to, iż nie osiągnięto końca pliku (zero, czyli logiczny FAŁSZ). Gdy plik zostanie zakończony funkcja przyjmie wartość logicznej PRAWDY (czyli 1) i działanie pętli zostanie również zakończone.

Polecenie FILEINPUT ma pewną wadę. Działa pod warunkiem, że w pliku nie został umieszczony znak końca linii (ang. End-Of-Line). Aby to sprawdzić możesz zastosować funkcję CHR$(), o której piszemy między innymi w punkcie pod tytułem „Operacje na zmiennych tekstowych".

Gdy znasz zawartość pliku lub chcesz odczytać jego treść linia po linii, jest to bardzo dobry wybór. W przeciwnym wypadku możesz mieć problemy z pobieraniem danych.

Dlatego Blitz Basic pozwala odczytywać pliki w inny sposób, za pomocą kolejnej funkcji o nazwie INKEY$. Sposób użycia jest nieco inny, bowiem jako parametr należy podać tylko ilość znaków, które chcemy uzyskać, na przykład:

```
dane[1]=Inkey$(100)
```

Operacja będzie wykonywana zawsze w stosunku do aktualnie „otwartego” pliku, dlatego trzeba uważać podczas jej stosowania w pętlach.

W tym wypadku możesz również nie podawać żadnego argumentu, wtedy domyślnie zostanie odczytany tylko jeden znak. To może mieć swoje zastosowanie nie tylko przy informacjach pobieranych z dysku. Piszemy o tym w punkcie pod tytułem „Obsługa plików”.

- Zapisywanie danych

Blitz Basic pozwala również „otworzyć” plik tak, aby możliwy był tylko jego zapis. Aby było to możliwe, zamiast słowa FILEINPUT należy wprowadzić polecenie FILEOUTPUT, analogicznie uzupełniając je numerem „otwartego” pliku. Na przykład:

```
a=OpenFile(1,”Worek:dane/dane3”)
```

a później:

```
FileOutput 1
```

Aby zapisywać dane w pliku należy jednak użyć zupełnie innej formuły. Korzystamy mianowicie ze zwykłych poleceń wypisujących informacje na ekranie, czyli:

```
PRINT
```

lub

NPRINT

Pisaliśmy o nich wcześniej. Teraz jednak ich „wyjście" będzie kierowało do pliku wskazanego w linii zawierającej funkcję OPENFILE().

Kolejnym sposobem zapisywania danych w pliku jest użycie funkcji WRITEFILE(), która działa analogicznie do omawianej wcześniej READFILE(). Tworzy automatycznie nowy plik i przyjmuje wartość 1, czyli logicznej PRAWDY, chyba że operacja zakończy się niepowodzeniem.

Zwróć uwagę, że w tym wypadku bardzo łatwo stracić zawartość pliku o podanej nazwie, bowiem funkcja nie sprawdza, czy taka sama pozycja była wcześniej zapisana na dysku.

Poniżej przedstawiamy przykład wykorzystania w ramach instrukcji warunkowej:

```
If WriteFile (1,"Worek:dane/dane3")
     FileOutput 1
          For i=0 To 10
               NPrint dane[i]
          Next
Endif
```

Pamiętaj, że różnice pomiędzy poleceniami PRINT i NPRINT nadal obowiązują. Innymi słowy, drugie słowo będzie powodowało tworzenie nowej linii przy każdym zapisywaniu danych. Jest to ważne, gdy tworzysz plik z wpisami wykonywanymi wielokrotnie, tak jak powyżej.

- Kończenie pracy z plikiem

Po zakończonej operacji odczytywania lub zapisywania informacji, plik powinien być „uwolniony", czyli musimy zamknąć kanał transmisyjny. Należy to zrobić za pomocą poniższego polecenia:

```
CloseFile 1
```

Obok niego podajemy po prostu numer pliku. Nie oznacza to ingerencji w treść czy też skasowania pozycji z dysku. Tę ostatnią funkcję omawiamy w punkcie pod tytułem „Usuwanie plików".

- Poruszanie się po zawartości pliku

Gdy mamy gotowy plik możemy z niego odczytywać dane w bardziej wyrafinowany sposób. Użycie omówionych poleceń wywołuje potrzebne funkcje, ale nie mamy kontroli nad aktualną pozycją w ramach zawartości. Jest to możliwe za pomocą dodatkowej polecenia o nazwie FILESEEK.

Jako argumenty należy tu podać numer pliku i pozycję, której na razie nie znamy. Na początek trzeba więc rozpoznać rozmiar pliku. Uzyskamy ją za pomocą funkcji LOF() (ang. Length-Of-File). Wystarczy, tak jak zwykle, przypisać ją do zmiennej i wprowadzić numer pliku, na przykład:

```
plik=Lof(1)
```

Teraz możemy stworzyć program sprawdzający pozycję w pliku. Jeśli chcemy „przenieść" się do określonego miejsca korzystamy ze słowa FILESEEK w następujący sposób:

```
FileSeek 1,1024
```

Powyższa linia spowoduje ustalenie pozycji na 1024 bajty, czyli pierwszy kilobajt pliku o numerze 1. Jeśli po wykonanej operacji chcesz sprawdzić, jaka jest nowa pozycja użyj z kolei funkcji LOC() (ang. Location-Of-File). Należy jej użyć analogicznie do LOF(), jednak przekazana wartość będzie oznaczać aktualną pozycję. Może być również zerowa, gdy plik został dopiero „otwarty" i nie wykonaliśmy na nim żadnej czynności.

Zwróć uwagę, że przy uwzględnieniu powyższych funkcji i poleceń możesz łatwo napisać program dopisujący dane do utworzonego już pliku. Trzeba tylko za każdym razem sprawdzać, czy plik istnieje, a następnie testować pozycję.

Możesz dodatkowo skorzystać z funkcji EXISTS(). Jako argument należy wprowadzić nazwę pliku, dzięki czemu uzyskamy liczbę odpowiadającą rozmiarowi pliku.

W tym miejscu możesz mieć wątpliwości, po co jej używać, skoro przekazuje te same dane, co LOF(). Są to jednak tylko pozory, bowiem funkcja informuje w ten sposób, że plik jest już „otwarty" i nie jest pusty. Gdyby było inaczej uzyskamy wartość 0 (zero) i na tym polega różnica w stosunku do poprzedniej funkcji.

- Usuwanie plików

Czasami będziesz potrzebować usunąć niepotrzebny plik z dysku. Tak się dzieje podczas korzystania z danych tymczasowych lub w sytuacji, gdy wystąpi błąd w zapisie. Operacja skasowana pliku jest bardzo łatwa, wymaga tylko wpisania następnej nazwy polecenia – KILLFILE. Jako argument podajemy nazwę pliku. Przykładowo tak jak poniżej:

```
KillFile „Worek:dane/dane3”
```

Ze względów bezpieczeństwa najlepiej zawsze wprowadzać także ścieżkę dostępu. W przeciwnym razie możemy doprowadzić do usunięcia potrzebnego jeszcze pliku. Jeżeli wpis będzie zawierał ścieżkę, nie ma możliwości, aby został pomylony z inną pozycją na dysku.

MAKRODEFINICJE

Tytułowe makrodefinicje to specjalne fragmenty programy pozwalające przyspieszyć jego działanie. Zwykle są spotykane w Asemblerze i działają bez pośrednictwa bibliotek systemowych, a więc z pełną dostępną szybkością procesora.

W Blitz Basicu makrodefinicje można je tworzyć podobnie do procedur, ale inne jest ich przeznaczenie i sposób wywoływania. Fragment programu musi zostać zapisany pomiędzy słowami MACRO i END MACRO, dodatkowo w pierwszej linii należy wprowadzić nazwę. Oto przykład:

```
Macro moje4
      a=5
      NPrint a
End Macro
```

W ten sposób utworzymy makrodefinicję o nazwie „moje4”. Wykonanie w programie uzyskujemy po wpisaniu linii rozpoczynającej się od znaku wykrzyknika. Możemy to osiągnąć następująco:

```
NPrint „Witaj”
!moje4
```

Jak widać, po wykrzykniku należy wpisać po prostu nazwę makrodefinicji. Jej treść musi być umieszczona wcześniej – przed linią wywołującą. Nie różni się to zasadniczo od obsługi innych elementów języka programowania.

Makrodefinicje wyróżniają się jednak sposobem działania od strony uruchamiania programu, bowiem podczas kompilacji treść umieszczona pomiędzy poleceniami MACRO i END MACRO nie będzie kompilowana, lecz umieszczona do wykonania w pamięci.

Tak więc nie będą wykonywane skoki do określonych części listingu, lecz w miejscu naszej nazwy zostaną wstawione polecenia zapisane w makrodefinicji. W praktyce nasz poprzedni program będzie miał poniższą formę:

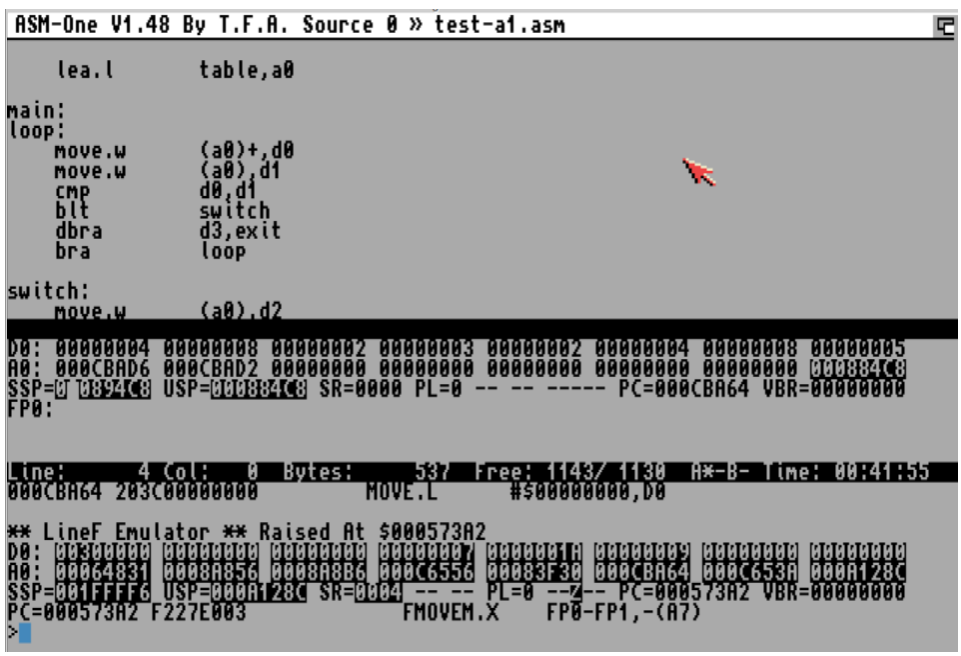
```
NPrint „Witaj”

a=5
NPrint a
```

Dzięki temu jego działanie może być szybsze, co ma szczególnie znaczenie przy pisaniu większych programów, które mają działać na nierozbudowanych modelach Amigi.

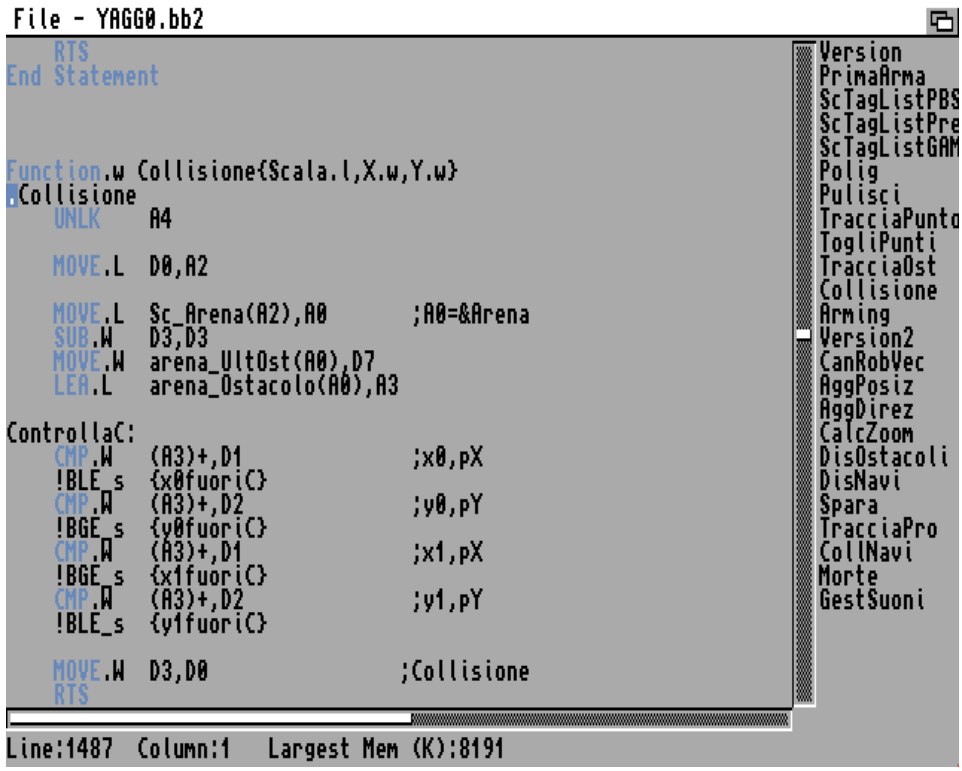
KORZYSTANIE Z ASEMBLERA

Blitz Basic pozwala na używanie kody maszynowego przeznaczonego do wykonania przez procesor, czyli tak zwanego Asemblera. Pisanie w ten sposób fragmentów programu jest dość skomplikowane, ale dzięki temu możesz uzyskać większą szybkość wykonywania i w bardziej bezpośredni sposób wpływać na działanie komputera.

Najłatwiejszym sposobem korzystania z możliwości Asemblera jest umieszczenie instrukcji w ramach procedury. Poszczególne polecenia możesz wpisać podobnie jak w edytorach takich jak „Asm-One". Zobacz jak może to wyglądać w oddzielnym programie:



oraz w edytorze Blitz Basica:



Oto kolejny przykład:

```
Statement moje7{x,y}
      MOVE.l d0,a0
      MOVE a0,d1
      ADD.l d2,a0
      AsmExit
End Statement
```

Zwróć uwagę, że przed zamknięciem struktury umieściliśmy słowo ASMEXIT, które powoduje wyście to „zwykłego" trybu działania programu.

Rezultaty działania możesz również przenosić do różnych rejestrów procesorów serii 68000. Pisaliśmy o nich krótko punkcie „Kompilacja i uruchamianie programu". Rejestry są komórkami pamięci, które służą do przechowywania informacji. Będziemy o tym mówić bardziej szczegółowo w kolejnej książce traktującej o zaawansowanych operacjach zarówno w Blitz Basicu, jak i Asemblerze.

Na razie zapamiętaj jak używać rejestrów w programie. Służą do tego dwa polecenia – GETREG oraz PUTREG. Pierwsze pozwala na umieszczenie wyniku w określonym rejestrze procesora. Obok podajemy nazwę rejestru oraz wartość jaka ma być zapisana. Na przykład poniższa linia:

```
GetReg d0,a
```

spowoduje zapisanie zawartości zmiennej „a" w rejesrtrze D0. Pamiętaj, że możesz korzystać z rejestrów danych od D0 do D7.

Polenie PUTREG działa odwrotnie, to znaczy przenosi wartość zapisaną w rejestrze procesora do zmiennej możliwej do obsługi w programie Blitz Basica. Może to wyglądać tak:

```
PutReg d1,c
```

Tym razem wartość z rejestru D1 zapisujemy w zmiennej o nazwie „c”.

Na razie pojęcie rejestrów, ich wykorzystanie i przypisane im wartości mogą niewiele Ci mówić, ale najważniejszy jest schemat użycia. Procesory serii 68000 posiadają wiele rejestrów, a Blitz Basic ma więcej poleceń związanych z obsługą Asemblerze. Wszystkim szczegółowo zajmiemy się później. Jeśli jednak posiadasz wiedzę z zakresu pisania programów w Asemblerze, powyższe informacje pozwolą Ci już teraz na skuteczne wykorzystanie kodu maszynowego.

INFORMACJE O SYSTEMIE

Z poziomu Blitz Basica możliwe jest bezpośrednie sprawdzenie niektórych elementów komputera i systemu operacyjnego. Może być to przydatne w sytuacji, gdy chcesz uzależnić działanie programu w zależności od różnych procesorów lub określonej wersji Kickstartu. Czasami może być to konieczne, aby program działał prawidłowo, a częściej w ten sposób będziesz mógł uzyskać lepszą wydajność.

Za pomocą funkcji PROCESSOR() możemy zorientować się, jaki procesor główny jest zamontowany w Amidze. Wystarczy przypisać ją do zmiennej, tak jak omawialiśmy to wcześniej, czyli na przykład:

```
a=Processor
NPrint a
```

Nie trzeba podawać żadnych argumentów. W odpowiedzi otrzymamy jedną z możliwych wartości, jak poniżej:

- procesor 68000	0
- procesor 68010	1
- procesor 68020	2
- procesor 68030	3
- procesor 68040	4

Bardzo podobnie sprawdzisz wersję systemu operacyjnego, a dokładniej biblioteki „exec.library". Służy do tego kolejna funkcja EXECVERSION(). Sposób użycia jest taki sam,:

```
b=ExecVersion
NPrint a
```

natomiast tym razem możesz się spodziewać poniższej odpowiedzi:

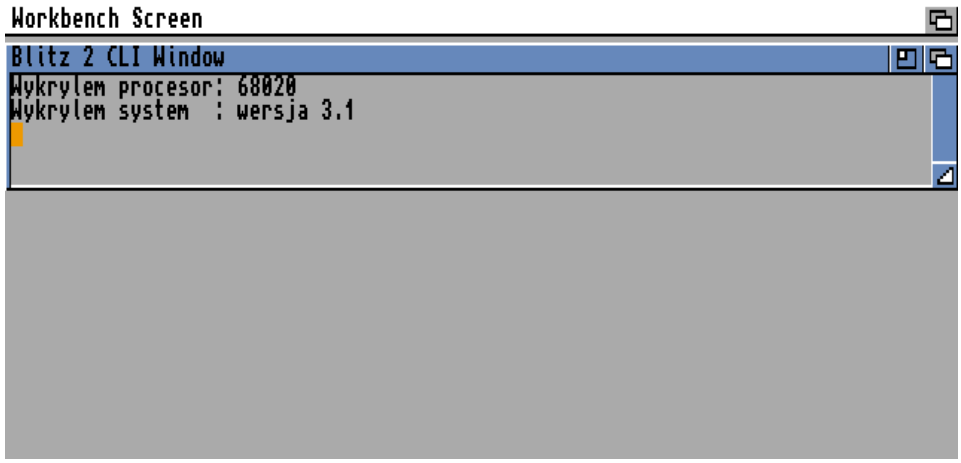
- system 1.2	33
- system 1.3	34
- system 2.0	36
- system 3.0	39
- system 3.1	40

Spójrz na przykładowy efekt działania powyższych funkcji. Tak wygląda listing przewidujący wszystkie wymienione pozycje:

File - test9

```
a=Processor
b=ExecVersion

If a=0 Then a$="68000"
If a=1 Then a$="68010"
If a=2 Then a$="68020"
If a=3 Then a$="68030"
If a=4 Then a$="68040"

NPrint "Wykrylem procesor: ",a$

If b=33 Then b$="wersja 1.2"
If b=34 Then b$="wersja 1.3"
If b=36 Then b$="wersja 2.0"
If b=39 Then b$="wersja 3.0"
If b=40 Then b$="wersja 3.1"

NPrint "Wykrylem system  : ",b$

MouseWait
```

a tak program działa na Amidze 1200 wyposażonej w Kickstart 3.1:



Oczywiście Twój program nie musi wyświetlać odczytanych wartości, dużo lepiej wprowadzić instrukcje warunkowe lub procedury, które będą działać nieco inaczej po wykryciu różnych elementów sprzętu i oprogramowania. W ten sposób program może działać bardziej uniwersalnie i w wielu przypadkach szybciej.

OBSŁUGA BŁĘDÓW

BŁĘDY W PROGRAMIE

Jeżeli piszesz program pozwalający na swobodną obsługę, użytkownik może wywołać błąd. Gdy zastosowane algorytmy nie należą do najprostszych także może się zdarzyć, że wystąpi błąd i Twój program ulegnie zatrzymaniu. Pamiętaj, że nawet jeśli wydaje Ci się to mało prawdopodobne, nie jesteś w stanie przewidzieć wszystkich możliwych sytuacji.

- Przechwytywanie błędów

Zwróć uwagę, że program nie powinien zatrzymywać się, lecz informować odbiorcę o błędzie lub uruchamiać odpowiednią procedurę eliminująca problem. Dlatego istnieje możliwość „przechwycenia" sytuacji problematycznych, aby program mógł dalej działać.

Jest to możliwe za pomocą kilku poleceń, wśród których na początek warto poznać SETERR. Może ona działać, gdy wywołany zostanie dowolny błąd powodujący normalnie zatrzymanie wykonywania programu.

Aby było to możliwe, na początku programu powinieneś umieścić konstrukcję SETERR … END SETERR, na przykład:

```
SetErr
     NPrint „Błąd"
End
End SetErr
```

W przedostatniej linii umieściliśmy polecenie END, bowiem w tej sytuacji program nie powinien przechodzić dalej. Oczywiście zamiast wyświetlenia komunikatu – lub obok funkcji obsługującej błąd – możesz wykonać „skok” do innej części listingu.

Po wykonaniu procedury program informacje o błędzie nie są automatycznie czyszczone, a więc należy je usunąć. Robimy to za pomocą polecenia CLRERR. Wystarczy wpisać jego nazwę w osobnej linii. Jeśli tego nie zrobisz, szybko okaże się, że po jednorazowym wystąpieniu problemu, będzie on ciągle powielany, czego oczywiście musimy uniknąć. W konsekwencji program nie będzie możliwy do uruchomienia.

Spójrz na podobny fragment programu w edytorze:

- Określanie rodzaju błędu

Ogólne „przechwycenie" problemu nie zawsze jest wystarczające. W wielu przypadkach musimy zmienić działanie programu w zależności od rodzaju błędu, jaki wystąpił. To z kolei umożliwia słowo SETINT, któremu trzeba podać wartość z poniższej listy:

```
 0        Pusty bufor transmisji (dotyczy portu
          szeregowego
 1        Błąd odczytu lub zapisu na dysku
 2        Przerwanie programu
 3        Problem związany z jednym z portów
          Amigi
 4        Przerwanie pracy układu Copper
11        Pełny bufor transmisji (znowu
          dotyczy portu szeregowego, jak 0)
12        Błąd oczytyu danych ze stacji
          dyskietek
13        Inny zewnętrzny problem
```

Samo polecenie SETINT nie wystarczy, bowiem polecenia należące do obsługi błędu muszą być zapisane w podobnej konstrukcji jak wcześniej, lecz tym razem całość kończymy za pomocą słowa END SETINT.

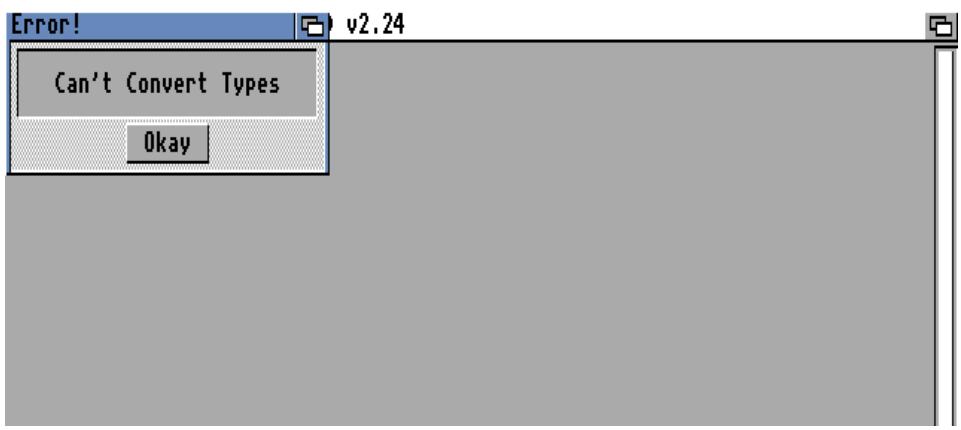
Na podanej liście błędów mamy tutaj wszystkie podstawowe rodzaje. Jeśli chcesz uzyskać więcej informacji lub obsłużyć bardziej szczegółowe problemy, musisz umieścić w programie dodatkowe zmienne, a potem uzależnić działanie w ramach konstrukcji SETINT … END SETINT.

Na przykład, gdy rozpoznane zostanie przepełnienie bufora portu szeregowego, odpowiednia zmienna może zapisywać ilość przesłanych danych i jeśli przynajmniej niektóre pliki są kompletne, wyświetlać odpowiedni komunikat przeznaczony dla użytkownika. Rzecz jasna, zastosowań może być wiele, ale zasada pozostaje zawsze taka sama.

KOMUNIKATY W EDYTORZE

Edytor „Ted” wyświetla informacje o błędach w programie. Mają one postać mniejszego okna w górnej części ekranu. Oto dwa typowe przykłady:

oraz



Wybierz przycisk „Okay”, aby powrócić do edycji listingu i postaraj się go poprawić. W ten sposób masz do dyspozycji krótki opis problemu, który może mieć różnoraki charakter. Możliwe są błędy składniowe, czyli nieprawidłowo wprowadzone polecenia albo zmienne, literówki i inne drobne pomyłki. Błędy mogą być bardziej poważne i dlatego nie zawsze łatwo jest zorientować się w sytuacji, a modyfikacje mogą być trudne do wprowadzenia.

Dlatego powinieneś dowiedzieć się więcej na temat możliwych rodzajów błędów oraz objaśnienie pojawiających się informacji. Są one krótkie, nie mają polskich odpowiedników, a więc możesz mieć wątpliwości jakie mają znaczenie. Dzięki dalszej części rozdziału w większości przypadków nie będziesz musiał zastanawiać się nad rodzajem problemu, ale jego rozwiązaniem.

BŁĘDY SKŁADNIOWE

Błędny związane ze składnią są łatwe do zrozumienia. Polegają na nieprawidłowym wprowadzeniu nazwy polecenia, jego argumentów lub innej części. Wszystko to powoduje, że linia technicznie może być prawidłowa, ale przykładowo ilość argumentów jest zbyt mała albo zostały ustalone złe rodzaje – liczba zamiast ciągu tekstowego.

Tego rodzaju błąd jest sygnalizowany informacją:

```
Syntax Error
```

Trzeba wtedy sprawdzić dokładnie treść problematycznej linii. Możesz też spotkać się z komunikatem o następującej treści:

```
Garbage at End of Line
```

Oznacza to brak istotnego składnika, jak na przykład średnika lub przecinka rozdzielającego poszczególne elementy polecenia. Podczas używania różnych wartości liczbowych możesz skorzystać z nieprawidłowej – zbyt małej lub zbytu dużej linii. W takiej sytuacji zobaczysz błąd:

```
Numeric Over Flow
```

Kolejny problem może wystąpić, gdy używasz linii z poleceniem DATA, ale podane wartości mają rodzaj niezgodny z wykorzystywaną zmienną. Blitz Basic zasygnalizuje to poniższą informacją:

Bad Data

Błąd ten nie mówi zbyt wiele, dlatego sprawdź wpisy w liniach DATA i porównaj je z pozostałą częścią programu, która powoduje odczytywanie danych i przypisywanie ich do zmiennej lub zmiennych.

Pamiętaj, że zła składnia możesz polegać na drobnej pomyłce, tak zwanej „literówce", która może nie być widoczna na pierwszy rzut oka. Ma to miejsce szczególnie często, gdy piszesz dłuższy program i starasz się szybko dokonać analizy działania. Podobne słowa mogą mieć różną składnię, a do zmiany funkcjonowania wystarczy pomyłka tylko jednego znaku.

BŁĘDY LOGICZNE

Działanie programu zależy od zastosowanych poleceń, zmiennych, różnych form algorytmów oraz innych elementów. Zwróć jednak uwagę na to, że program może działać prawidłowo, a mimo wszystko nie realizować założonych funkcji. W związku z tym teoretycznie można go używać, ale nie jest w pełni sprawny.

Takie problemy nazywamy właśnie błędami logicznymi. Są trudne do rozpoznania i poprawienia, bowiem trzeba to zrobić samodzielnie. Sam edytor języka programowania nie wskaże problemu, jako że nie posiada on inteligencji mogącej ocenić Twoje intencje.

Podobne sytuacje występują tym częściej, im bardziej złożony (niekoniecznie dłuższy) jest program. Można się jednak przed nimi bronić stosując kilka prostych zasad. Do najpopularniejszych technik zmniejszających ilość błędów logicznych należą między innymi:

- wprowadzanie zrozumiałych nazw zmiennych, etykiet,
- procedur i innych elementów,

- unikanie używania skrótów, które mogą być niejednoznaczne
- lub uzależnione od innych fragmentów programu,

- tworzenie jak najbardziej prostych struktur, które są łatwe
- do analizy,

- korzystanie z komentarzy opisujących zmienne, cele oraz
- założenia programu, a także oddzielają różne części.

Te punkty to tylko kilka podstawowych schematów postępowania, jednak poprawią one czytelność programu i sprawią, że zrozumienie działania algorytmów będzie zdecydowanie łatwiejsze. Dzięki temu możesz oszczędzić czas podczas testowania programu, a więc osiągniesz swój cel szybciej i bez komplikacji.

BŁĘDY ZWIĄZANE Z INSTRUKCJAMI WARUNKOWYMI

Następny rodzaj błędów jest związany z instrukcjami warunkowymi, czyli liniami typu IF … THEN lub dłuższymi konstrukcjami IF… ENDIF.

Najbardziej typowym problemem jest brak słowa ENDIF, gdy rozpocząłeś linię z poleceniem IF. W takim wypadku zobaczysz następujący komunikat:

```
If Without End If
```

Zwróć uwagę, że może mieć to miejsce także wtedy, gdy korzystać z wielu warunków i umieściłeś słowo ENDIF w niewłaściwym miejscu. Gdy wprowadzasz kolejne warunki w ramach większej struktury IF … ENDIF może się też zdarzyć, że ilość odpowiadających sobie poleceń jest prawidłowa, ale jedno nie za każdym razem jest uwzględniane z uwagi na zmiany działania Twojego warunku po słowie IF. Musisz poprawić linie tak, aby wszystkie ważne linie były zawsze wykonywane lub zmienić pozycję polecenia ENDIF.

Twój warunek może być też zbyt długi. Blitz Basic pozwala zapisywać je w blokach o maksymalnej objętości 32 kilobajtów. Nie jest to mało, ale jeżeli przekroczysz tę wartość wyświetlona zostanie następująca informacja:

```
If Block too Large
```

W takiej sytuacji podziel warunek na kilka mniejszych albo zmodyfikuj program tak, aby nie trzeba było za każdym razem sprawdzać wszystkich części. Można to osiągnąć przez operacje takie jak:

- zmniejszenie ilości zmiennych,
- wstawienie zmiennych tekstowych w celu skrócenia zapisu warunku,
- ustalenie mniej szczegółowych zakresów dla zmiennych liczbowych,
- wykorzystanie tablic,
- utworzenie procedur przekazujących informacje w inny sposób niż wcześniej

i wiele innych. Podaliśmy kilka przykładów, natomiast właściwa modyfikacja będzie zależeć od charakteru Twojego programu. W skrajnych przypadkach będziesz musiał zmienić cały używany algorytm, ale zwykle wystarczające są mniejsze poprawki.

BŁĘDY ZWIĄZANE Z PĘTLAMI

Pętle są także związane z użyciem konkretnych słów w ramach jednej struktury powtarzającej wykonywanie linii. Jeśli zapisana zostanie zbyt duża ilość polecenia FOR lub NEXT w stosunku do pozostałej treści, zobaczysz informację:

```
Duplicate For...Next Error
```

Jeżeli natomiast zabraknie rozpoczynającego słowa FOR, Blitz Basic wyświetli komunikat mówiący, że nie można „zamknąć" pętli, która nie była wcześniej utworzona. Dzieje się to w następujący sposób:

```
Next without For
```

Możesz również wykonać operację odwrotną – wprowadzić FOR bez „zamykającego" słowa NEXT. W takim wypadku zostaniesz o tym poinformowany poniższym napisem:

```
For Without Next
```

Podobne komunikaty występują dla innych rodzajów pętli, mianowicie:

```
Until without Repeat
```

lub

```
Repeat without Until
```

Oczywiście dotyczy to braku jednego z podanych słów ramach struktury REPEAT … UNTIL.

Treść pętli może być zbyt długa, podobnie jak w przypadku instrukcji warunkowych. Możesz się o tym przekonać, gdy wyświetlony zostanie błąd:

```
For...Next Block to Long
```

lub

```
Repeat Block too large
```

Pierwszy dotyczy pętli typu FOR … NEXT, drugi – REPEAT … UNTIL. W obu przypadkach maksymalna długość wynosi 30 kilobajtów. Raczej rzadko spotkasz się z sytuacją przepełnienia miejsca, ale warto wiedzieć o tym limicie.

Pętla FOR … NEXT może zawierać zmienną o nieprawidłowym rodzaju. Zobaczysz wtedy informację jak poniżej:.

```
Bad Type for For...Next
```

Wskazuje ona, iż zmienna obok słowa FOR musi być liczbą, w przeciwnym razie nie będzie możliwe wykonanie kolejnych obiegów pętli.

W pętli typu WHILE … WEND możesz stosować polecenie ELSE. Gdy zostanie wprowadzone w nieprawidłowy sposób, zobaczysz taki napis:

`Illegal Else in While Block`

Zwróć uwagę, że wszystkie pętle oczekują określonych rodzajów zmiennych oraz używanie poszczególnych słów w uporządkowany sposób. Nie możesz zamieniać swobodnie kolejności poleceń, bowiem zawsze jedno oznacza początek, a drugie – koniec pętli.

BŁĘDY ZWIĄZANE Z ETYKIETAMI, PROCEDURAMI ORAZ FUNKCJAMI

Jeśli korzystasz z dużej ilości procedur, możesz popełnić błędy polegające na nieprawidłowym wprowadzeniu argumentów. Gdy zdarzy się, że wpiszesz ich zbyt mało, zobaczysz następującą informację:FO

```
Not Enough Parameters
```

Podobny komunikat zostanie pokazany w stosunku do złej ilości argumentów polecenia lub parametrów funkcji. W tej sytuacji nie próbuj poprawiać linii metodą „chybił-trafił", ale sprawdź poprawną ilość poszczególnych elementów. W przeciwnym razie możesz doprowadzić do nieprzewidzianego działania programu, choć zwykle Twój listing po prostu nie będzie możliwy do uruchomienia.

Parametrów lub argumentów w linii można użyć także za dużo, aby polecenie mogło zostać wykonane. W tej sytuacji pojawi się inny błąd, jak poniżej:

```
Too many parameters
```

Parametry mogą także posiadać nieprawidłowe rodzaje. Gry zastosujesz zmienną nie w tym miejscu, w którym powinna się znaleźć lub spróbujesz utworzyć typ niezgodny z zawartością, na ekranie pojawi się błąd o treści:

```
Illegal Parameter Type
```

Funkcje muszą posiadać zmienne, które nie będą pokrywać się z zapisanymi w głównym programie. W przeciwnym razie program nie zadziała, bowiem wywołany zostanie następujący problem:

```
Duplicate parameter variable
```

W tej sytuacji musisz poprawić zmienne tak, aby wskazywały na pozycje, które nie mają swoich odpowiedników poza funkcją.

Gdy z kolei tworzysz wiele procedur może się zdarzyć, że spróbujesz wprowadzić „jedną w drugiej", czyli zagnieździć treść jednej procedury w ramach stworzonej wcześniej. Taki zapis nie jest dopuszczalny, dlatego zobaczysz błąd jak poniżej:

```
Can't Nest Procedures
```

Sytuacja wygląda podobnie z tablicami, które – o ile mają być dostępne w całym programie – nie mogą być deklarowane w ramach procedury. Taką tablicę musisz przenieść od głównego programu lub stworzyć nowy parametr przekazujący zawartość elementu z określonym indeksem. Jeśli będzie inaczej wywołasz kolejny komunikat:

```
Can't Dim Globals in Procedures
```

Pamiętaj, że procedury należy wywoływać w określony sposób. Nie są do tego przeznaczone polecenia związane z etykietami i podprogramami jak GOTO czy GOSUB. Dlatego próba uruchomienia procedury przy ich użyciu skończy się następującą informacją:

```
Can't Goto/Gosub a Procedure
```

Następny problem jest związany z nazewnictwem określonych fragmentów programu. Gdy spróbujesz wywołać procedurę, która nie istnieje zobaczysz błąd o poniższej treści:

```
Procedure not found
```

Oznacza to, że popełniłeś błąd w nazwie lub wskazałeś część, która nie została utworzona jako procedura. Podobny komunikat otrzymasz w momencie wprowadzenia nazwy etykiety, której nie ma w listingu. Różnicę stanowi początek informacji, która wygląda tak:

```
Label not Found
```

Tak jak mówiliśmy wcześniej, nazwy procedur lub etykiet nie mogą się pokrywać. Nie możesz utworzyć dwóch takich samych pozycji, bowiem nie byłoby wiadomo, do której odnoszą się polecenia w programie. Jeżeli jednak spróbujesz zdefiniować dwie identyczne nazwy, zobaczysz informację dotyczącą procedury:

```
Duplicate Procedure name
```

lub etykiety:

```
Duplicate Label
```

Zwróć również uwagę na to, że nazwy różnych fragmentów programu nie mogą być zupełnie dowolne. Przykładowo nie mogą być identyczne jak nazwy poleceń. Można o tym zapomnieć, bo przecież jest wiele słów możliwych do wprowadzenia do listingu.

Dlatego, jeśli nazwa będzie z różnych powodów nieprawidłowa, na ekranie pojawi się następujący napis:

```
Illegal Procedure Call
```

lub

```
Illegal Label Name
```

Analogicznie do poprzedniego przypadku, pierwszy dotyczy procedury, a drugi – etykiety.

Pamiętaj, że każda funkcja lub procedura – jako kompletna struktura - musi zostać w odpowiedni sposób zakończona. Mogą być to słowa END STATEMENT, o których pisaliśmy w oddzielnym punkcie. Gdy o tym zapomnisz, Blitz Basic wyświetli kolejną informację jak poniżej:

```
Unterminated Procedure
```

Istnieje jeszcze jeden ważny komunikat dotyczący etykiet. Jego treść to:

```
Can't Access Label
```

Mówi on o tym, że niemożliwy jest dostęp do etykiety. W praktyce zwykle pojawia się w sytuacji, gdy nazwa nie została wcześniej zdefiniowana. Możesz to jednak zobaczyć także wtedy, gdy wywoływanie etykiety będzie niemożliwe z innych względów.

BŁĘDY ZWIĄZANE Z OBSŁUGĄ TABLIC

Tablice są specyficznymi zbiorami danych i łatwo można je pomyli ze zwykłymi zmiennymi. Gdy będziesz próbował odwołać się do tablicy, która nie została zdefiniowana zobaczysz informację jak poniżej:

```
Array not found
```

Zwykle ma to związek z użyciem nawiasów w stosunku do elementu, który występuje w programie, ale nie jest tablicą. Innym powodem może być użycie nawiasów klamrowych zamiast zwykłych. Powyższy komunikat może być także zastąpiony dwoma innymi:

```
Array not yet Dim'd
```

lub

```
Array not Dim'd
```

Ich znaczenie jest identyczne. Z drugiej strony możesz zapomnieć, że w programie użyłeś już tablicy o określonej nazwie. Jeżeli będziesz chciał utworzyć drugi raz taką samą tablicę, zostanie wypisana następująca informacja:

```
Array already Dim'd
```

Każda tablica posiada indeksy, według których zapisywane są różne wartości zmiennych. W Blitz Basicu nie możesz używać indeksów złożonych, które mogłyby zapamiętywać większą ilość wartości, na przykład rozdzielonych przecinkiem. Próba wprowadzenia takiego indeksu spowoduje poniższy błąd:

```
Illegal number of Dimensions
```

Oznacza on, że podano nieprawidłowy (nieobsługiwany) indeks i trzeba go poprawić. W tym miejscu mówimy o braku obsługi, bowiem istnieją języki programowania pozwalające tworzyć bardziej rozbudowane tablice.

Indeksy nie mogą być również zmiennymi. Mówiąc precyzyjnie: możesz korzystać ze zmiennych podczas używania tablic w ramach pętli, ale sam indeks musi być zawsze liczbą. Jeżeli wprowadzisz do niego zmienną, wywołany zostanie komunikat o treści:

```
Can't Create Variable inside Dim
```

Więcej na temat obsługi tablic piszemy w oddzielnym punkcie.

BŁĘDY ZWIĄZANE Z MAKRODEFINICJAMI

Podczas tworzenia makrodefinicji również można popełnić przynajmniej kilka błędów. Pierwszy z nich to odwołanie się do pozycji, która nie została zdefiniowana. W takim wypadku zobaczysz poniższy komunikat:

```
Macro not Found
```

Możesz także próbować stworzyć dwie makrodefinicje o tej samej nazwie. Spowoduje to wywołanie poniższej kolejnej informacji o treści:

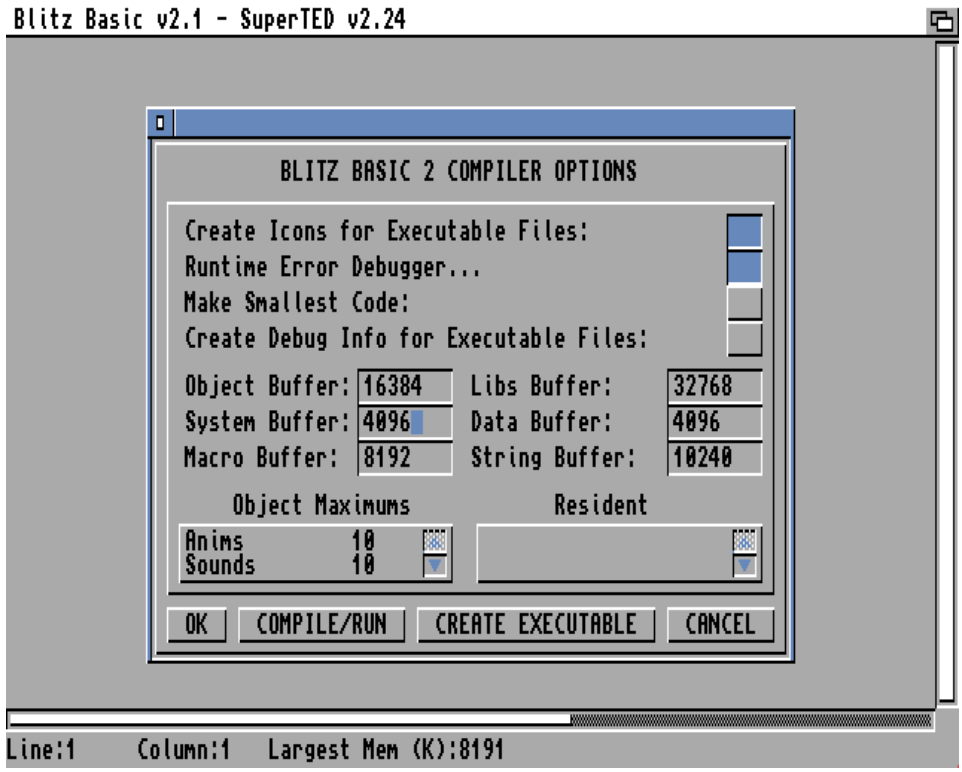
```
Macro already Defined
```

Zawartość struktury może być zbyt długa, co z kolei zostanie zasygnalizowane napisem:

```
Macro too Big
```

Zwróć uwagę, że rozmiar bufora dla makrodefinicji można zmienić w opcjach konfiguracyjnych. Musisz wywołać menu górne „Compiler", a następnie wskazać opcję „Compiler Options...".

Na ekranie zobaczysz okno, w którym widoczny jest kilka pól tekstowych. Zmień wartość widoczną w polu o nazwie „Macro Buffer". Tak jak na naszej ilustracji:



Gdy bufor mimo wszystko zostanie przepełniony możesz zobaczyć inny błąd:

```
Macro Buffer Overflow
```

Może on pojawić się również w sytuacji, gdy Twoja makrodefinicja będzie wykonywana w nieskończonej pętli, co nie powinno mieć miejsca.

Każda makrodefinicja musi być rozpoczęta słowem MACRO i zakończona na ENDMACRO. Jeżeli będzie inaczej program nie zadziała i wyświetlony zostanie komunikat:

```
Macro without End Macro
```

Oznacza on, że w listingu brakuje „zakończenia", czyli polecenia ENDMACRO. Pamiętaj, że w tym wypadku makrodefinicje muszą być tworzone w głównej części programu. Gdy spróbujesz zrobić inaczej Blitz Basic wypisze informację jak poniżej:

```
Can't create Macro inside Macro
```

Makrodefinicje są dość skomplikowane w obsłudze, natomiast Blitz Basic nie zawiera dużej ilości błędów z nimi związanych. Ich wykorzystanie omawiamy szerzej w rozdziale zatytułowanym „Operacje zaawansowane".

INNE BŁĘDY

W rozdziale „Obsluga błędów w programie” piszemy o konstrukcji SETINT … END SETINT. Jeśli nie zostanie wprowadzona prawidłowo, zobaczysz informację:

```
End SetInt without SetInt
```

lub

```
SetInt without End SetInt
```

Pierwsza z nich oznacza, że brakuje początku struktury, czyli słowa SETINT. Druga wskazuje na błędne zakończenie. Jeżeli wydaje Ci się, że wszystkie polecenia zostały wpisane, prawdopodobnie popełniłeś błąd w pojedynczym znaku.

Podobna sytuacja może wystąpić podczas tworzenia kontrukcji SETERR … END SETERR. I tak samo może być wyświetlona informacja o braku jednego z wymaganych słów:

```
End SetErr without SetErr
```

Jeżeli natomiast sprawdzasz rodzaj błędu i wpiszesz nieprawidłowy, czyli nieobsługiwany numer oznaczający problem, możesz się spodziewać poniższego komunikatu:

```
Illegal Interrupt Number
```

Więcej informacji znajdziesz w punkcie zatytułowanym „Określanie rodzaju błędu”.

Podczas odczytywania pliku może wystąpić błąd. Nie jest to sytuacja niebezpieczna, ale dane nie zostaną wczytane, tak więc program nie może być wykonany. Na ekranie pojawi się wtedy następujący komunikat:

```
Error Reading File
```

UWAGI DO KOMUNIKATÓW O BŁĘDACH

Pamiętaj, że wymienione komunikaty o błędach to tylko wybrane informacje, jakie możesz zobaczyć w edytorze Blitz Basica. Gdy pojawi się napis podobny do opisanego, zwróć uwagę na wszystkie wymienione słowa. Podobnie wyglądający komunikat niekoniecznie musi dotyczyć tego samego elementu lub tej samej sytuacji w programie. W trakcie obsługi powinieneś nauczyć się rozpoznawać prawidłowo błędy oraz nazwy poleceń z nimi związanych.

Bardzo ważne są również błędy logiczne, o których pisaliśmy wcześniej. Mogą one bowiem powodować, że wyświetlane informacje będą Cię kierować do innych fragmentów programu, niż faktycznie wymagające poprawek. Dlatego zawsze zacznij analizę od sprawdzenia sposobu wykonywania algorytmu, a dopiero później skupiaj się na konkretnych informacjach o stanie zmiennych lub parametrów funkcji.

ZAKOŃCZENIE

Podobnie jak w poprzedniej książce, założona objętość nie pozwala na omówienie wszystkich możliwości Blitz Basica. Wiele tematów zostało jedynie zasygnalizowanych, dlatego będą kontynuowane w drugiej pozycji. Zajmę się w niej kwestiami takimi jak operacje wykorzystujące bezpośrednią obsługę układów specjalizowanych Amigi, korzystanie z języka skryptowego ARexx, używanie większej ilości składników systemu operacyjnego waz z tworzeniem kompletnego interfejsu użytkownika i wiele innych tematów. Powiem także więcej na temat nowej odsłony języka o nazwie „AmiBlitz", jak również przygotowywaniu programów przeznaczonych dla systemów AmigaOS 4 i MorphOS. Będzie więcej rozbudowanych przykładów, a mniej wyjaśniania podstawowych zasad obsługi i działania. To wszystko znajdzie się w treści osobnej serii zatytułowanej „Programowanie dla zaawansowanych".

Mam nadzieję, że moje rozważania przyczynią się do zwiększenia zainteresowania programowaniem ze strony użytkowników Amigi lub osób uruchamiających popularne emulatory. Stworzenie własnego programu użytkowego czy gry zawsze wymaga wielu godzin nauki i pracy, czasem dostarcza rozczarowań, ale przede wszystkim przynosi wiele satysfakcji i poszerza horyzonty. Blitz Basic istnieje także dla komputerów PC, tak więc nabyte doświadczenie można wykorzystać na polu innych systemów operacyjnych.

DODATEK

INDEKS POLECEŃ I FUNKCJI

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

Volume 231

W

X

Y

Z

INFORMACJE TECHNICZNE

BLITZ BASIC

Autor:
Mark Sibly

Premiera:
1994

System operacyjny:
AmigaOS

Strona internetowa:
www.blitzbasic.com

337