Borland C++ Compiler/Wersja do druku

[zwiń]

Z Wikibooks, biblioteki wolnych podręczników.
> Borland C++ Compiler > Wersja do druku »

Karol Ossowski

Borland C++ Compiler

Aktualna, edytowalna wersja tego podręcznika jest dostępna w Wikibooks, bibliotece wolnych podręczników pod adresem
http://pl.wikibooks.org/wiki/Borland_C%2B%2B_Compiler

Podręcznik ten jest zwięzłym opisem działania borlandowskiego zestawu narzędzi - FreeCommandLineTools do programowania w języku C/C++, w środowisku Windows. Kurs obejmuje instalację pakietu, podstawy użytkowania FreeCommandLineTools oraz opis narzędzi: BCC32 (kompilator), ILINK32 (konsolidator) i MAKE (narzędzie do budowania projektów). Chcemy aby był on zrozumiały również dla początkujących programistów dla których być może jest to pierwszy kompilator C++, jaki zamierzają używać, stąd wiele przykładów i łopatologiczne wręcz tłumaczenie niektórych zagadnień.

---

4. Składnia makefile'a
5. Komentarze
6. Zasady ogólne i szczegółowe
    1. Zasady oglne (*implicit rules*)
    2. Zasady szczegółowe (*explicit rules*)
    3. Zasada "bez zasad" :-)
7. Komendy systemowe
8. Zmienne
    1. Zmiana wartości zmiennej
    2. Specjalne zmienne
9. Dyrektywy
    1. Pliki projektu i konfiguracja MAKE
    2. Instrukcje warunkowe
    3. Ekran
10. MAKE w praktyce

Wstęp

O tym podręczniku

Budowa

Podręcznik ten został podzielony na 4 rozdziały:

- Rozdział pierwszy - **Wprowadzenie** wyjaśnia czym tak naprawdę jest kompilator, w jaki sposób on działa (na przykładzie kompilatora BCC32). Ponadto szczegółowo omawia instalacje pakietu FreeCommandLineTools oraz jego podstawową budowę.
- Rozdział drugi - **BCC32** skupia się na kompilatorze z w/w pakietu. Zawiera on opis większości przydatnych opcji uruchamiania tego narzędzia i praktyczne wskazówki jak go używać.
- Rozdział trzeci - **ILINK32** omawia konsolidator (inaczej linker). Poza opisem rozbudowanej składni parametrów wywołania tego narzędzia (wraz z ważniejszymi opcjami wywołania) dodatkowo zawiera również dość szczegółowo opisaną składnie pliku DEF używanego w pakiecie.
- Rozdział czwarty - **MAKE** przedstawia narzędzie dzięki, któremu kompilacja programów jest bardzo szybka i wygodna. Z tego rozdziału nauczsz się tworzyć borlandowskie makefile'e - używać w nich zmiennych, zasad i dyrektyw, które zautomatyzują proces budowy Twojego programu.

Konwencje

Składnia poleceń, dyrektyw itp. została przedstawiona w schematycznych szablonach:

```
ramka <zmienna1> [coś co nie jest wymagane <zmienna2>] \
dokończenie schematu
```

w podobnych ramkach umieszczono przykłady użycia schematów

```
to jest przykład \
dokończenie przykładu
```

W każdym szablonie mogą pojawić się następujące specjalne symbole:

- `[]`: nawiasy kwadratowe - wewnątrz nich są usytuowane elementy opcjonalne, nie konieczne do prawidłowego użycia danego schematu
- <> nawiasami ostrymi oznaczam elementy, które powinny być zastąpione czymś innym, z reguły to co jest wewnątrz tych nawiasów sugeruje co należy wstawić, w przeciwnym wypadku jest to wyjaśnione poniżej schematu, lub w konkretnym przykładzie.

Specjalnym symbolem jest też: `\` - odwrócony slash (*backslash*) - załamanie linii, które zostało wprowadzone aby nie narazić Was na przewijanie ekranu w poziomie (w żadnym wypadku nie powinno się tego przepisywać!)

Dodatkowo w tekście:

- Elementy schematów i składni poleceń, słowa kluczowe, opcje wywołania jak też nazwy plików zostały wyróżnione `czcionką maszynową`.
- Angielskie, nie przetłumaczone nazwy (zaczerpnięte najczęściej wprost z manuala

FreeCommandLineTools) i trudne pojęcia są zaznaczone *czcionką pochyłą*.
- Nazwy narzędzi/programów są wypisane WIELKIMI LITERAMI
- FreeCommandLineTools od tej chwili będzie nazywane FCLT

Autorzy

Jeśli uważasz siebie za autora tego podręcznika, to dopisz się tutaj.

- Karol Ossowski

Historia

Pierwsza wersja tego podręcznika powstała dla magazynu internetowego (tzw. eZinu) @t (http://www.at.rwi.pl/) (nr'y 32 i 33). Kolejne wersje były publikowane na stronie http://www.viilo.torun.pl/~re_set/bcc.html. Gdzie w pewnym momencie podręcznik został "uwolniony", tzn. wydano go na licencji GFDL, skąd już krótka droga do pojawienia się tutaj ;)

Do zrobienia

W myśl zasady: nigdy nie jest tak, żeby nie mogło być lepiej ;-).

- bardziej rozbudowana notka o edytorach
- opis innych narzędzi z FCLT (np. IMPLIB)
- stworzenie wersji do druku

Propozycje

Zachęcam(y) przede wszystkim do czynienia tego podręcznika lepszym, na własną rękę, ale jeśli nie jesteś w stanie napisać tego o czym myślisz, to tutaj możesz zgłaszać propozycje.

Wprowadzenie

Kompilator? A z czym to się je?

Kompilator to specjalny program, który przekształca twój kod (np. w C++) na plik wykonywalny EXE, który później możesz uruchomić klikając dwukrotnie na jego ikonkę... no to tak w dużym uproszczeniu, (szczegóły w postaci chociażby konsolidatora na razie wam ominę). Niektórzy mylą pojęcia kompilator ze zintegrowanym środowiskiem takim jak np. Visual C++ czy darmowy Dev-C++. To już nie tylko kompilator ale tzw. IDE, czyli taka graficzna otoczka dla kompilatora, debuggera, konsolidatora itp. Skoro już wiesz - idźmy dalej...

Dlaczego warto to mieć na swoim twardzielu

Wielu z was zapytałoby: Po co się męczyć z gołymi kompilatorami skoro można użyć zintegrowanego środowiska? Przede wszystkim taki np. Dev-C++ nie daje Ci 100% konfigurowalności. Z Borland C++ Compiler sam decydujesz np. o poziomie optymalizacji swojego kodu, masz dostęp do wszelkich szczegółów nawet wyboru swojego IDE (edytorów przystosowanych do C++ jest w internecie "od groma"). Poza tym miliony ludzi na całym świecie używają Borland C++ Compiler, bo jest to narzędzie tworzone przez najlepszych profesjonalistów.

Skąd wziąć FCLT?

Po prostu wejść na stronę polskiego oddziału firmy Borland: http://www.borland.pl/ Znaleźć dział Download i ściągnąć. W najnowszej wersji jakieś 8 i pół MB. Przedtem jednak trzeba się zarejestrować... Od tej chwili poza nowym kompilatorem C++ nasza skrzynka będzie "zapychana" listami od firmy Borland, a to Developer Days, a to nowe środowisko, które ostatnio wypuścili, ale spokojnie nie będzie tego dużo, jakoś to przeżyjecie ;).

Instalacja

Po uruchomieniu instalatora warto umieścić nasz kompilator w standardowej lokalizacji (`C:\Borland\Bcc55`) przynajmniej na początku, unikniemy wtedy późniejszych nieporozumień. Następnie wypadałoby połączyć linkera oraz kompilator z bibliotekami i standardowymi plikami nagłówkowymi. W tym celu należy uruchomić notatnik systemowy i wklepać lokalizację `include` (pliki nagłówkowe)i `lib` (biblioteki) tak:

```
-I"C:\Borland\Bcc55\include"
-L"C:\Borland\Bcc55\lib;C:\Borland\Bcc55\lib\PSDK"
```

Analogicznie: jeśli masz gdzieś jeszcze biblioteki powinieneś dopisać jeszcze jeden ';' przy -L i podać ich lokalizacje, to samo z *includami*. **UWAGA** - jeżeli twoja aplikacja będzie uruchamiana pod systemem niższym niż Win2k to musisz dopisać kolejne linijki:

```
-DWINVER=0x0400
-D_WIN32_WINNT=0x0400
```

Teraz powinieneś zapisać ten plik w katalogu bin jako: `bcc32.cfg`. Uważaj aby notatnik nie zapisał go jako

`bcc32.cfg.txt`! Dla konsolidatora warto powiedzieć gdzie są jego biblioteki: stwórz plik `ilink32.cfg` i wpisz do niego:

```
-L"C:\Borland\Bcc55\lib;C:\Borland\Bcc55\lib\PSDK"
```

Teraz trzeba powiadomić system gdzie jest nasze FCLT aby można je było uruchamiać bez podawania lokalizacji. W tym celu musimy do zmiennej środowiskowej `PATH` dodać lokalizację katalogu bin naszego kompilatora. W systemach **Win9x** można to zrobić na 2 sposoby:

- edytować plik `Autoexec.bat` i dopisać do niego linijkę:

  ```
  SET PATH=%PATH%;C:\Borland\Bcc55\Bin
  ```

- uruchomić `msconfig` (Start>Uruchom>`msconfig`) kliknąć zakładkę Autoexec.bat i dodać powyższą linijkę.

W **WinME** są również 2 drogi:

- w `Autoexec.bat` do zmiennej środowiskowej `PATH` dopisać:

  ```
  ;C:\Borland\Bcc55\Bin
  ```

- uruchomić `msconfig` i w zakłdce Środowisko do zmiennej `PATH` dopisać powyższą linijkę

Natomiast w **Win2k/XP** we właściwościach systemu (Właściwości systemu>Zaawansowane>Zmienne środowiskowe): należy dodać zmienną `PATH` i dopisać jej wartość:

```
%PATH%;C:\Borland\Bcc55\Bin
```

W **Windows7** panel sterowania>system>zaawansowane ustawienia systemu, w zakładce zaawansowane wybieramy zmienne środowiskowe, edytujemy zmienną systemu "Path" na koniec linijki dopisujemy **;C:\Borland\Bcc55\Bin**

Co mamy w katalogu bin

BCC32	kompilator c/c++
ILINK32	konsolidator; BCC32 sam go wywołuje jednak ilink32 przydaje do budowy zaawansowanych projektów
BRCC32	kompilator zasobów, czyli narzędzie do kompletowania ikonek, bitmap, czy nawet łańcuchów tekstu, żeby nie zalegały one sobie w katalogu z naszym programem
BRC32	łączy nieskompilowane skrypty zasobów (pliki *.rc) z EXE'kiem
MAKE	świetne narzędzie do zarządzania projektami
TOUCH	zmienia datę i czas ostatniej modyfikacji pliku na bieżącą
CPP32	preprocesor C; bcc32 sam go wywołuje
GREP	przeszukuje pliki tekstowe.
FCONVERT	konwertuje teksty z OEM na ANSI i odwrotnie.
TDUMP	podaje szczegółowe dane na temat pliku EXE (np. funkcje z bibliotek DLL z jakich korzysta).
IMPLIB	na podstawie bibliotek DLL i/lub plików *.def tworzy biblioteki statyczne (*.lib)
TLIB	pozwala na opercje na bibliotekach statycznych (*.lib) (dodawanie/odejmowanie /ekstrakcję/... itd. plików *.obj)

Podstawy użytkowania FCLT

Każde narzędzie dostępne w pakiecie uruchamia się jak sama nazwa wskazuje z wiersza poleceń (*commandline*). Aby uruchomić wiersz poleceń klikamy kolejno Start>Uruchom i w polu tekstowym wpisujemy `command` lub `cmd` (w zależności od systemu). Teraz możemy zacząć wpisywać pierwsze polecenia. Ogólny szablon wywoływania narzędzi FCLT wygląda tak:

```
<tool> [<opcje>] <plik(i)>
```

`<tool>`
: nazwa narzędzia np. BCC32, ILINK32, TDUMP

`<opcje>`
: symbole znakowe z przedrostkiem w postaci myślnika: '-' lub slash'a: '/' - konkretne wytyczne dla danego narzędzia określające sposób jego pracy, mające wpływ na ostateczny wynik działania programu; np. aby kompilować pod Windows trzeba dodać opcje -tW do wywołania kompilatora

`<plik(i)>`
: nazwa/y jednego lub więcej plików źródłowych na podstawie których dane narzędzie będzie generować plik(i) wynikowy/e; jeżeli nie jesteśmy w aktualnej lokalizacji w której znajduje się dany plik źródłowy to trzeba podać jego bezpośrednią ścieżkę.

Elementy `<opcje>` i `<plik(i)>` powyższego szablonu tworzą razem tzw. parametry wywołania programu, natomiast wszystko to co pojawi się na ekranie po jego wywołaniu nazywamy jego standardowym wyjściem (*stdout*). Jeśli program nie natrafi na żadne błędy składniowe w pliku/ach źródłowym/ch to plik(i) wynikowe zapisywane są do lokalizacji, w której aktualnie się znajdujemy niezależnie od tego gdzie znajdowały się plik(i) źródłowe. Wyjątkiem od tej reguły jest program BRCC32.

Jak BCC32 buduje program

1. Najpierw analizowane są parametry wywołania BCC32: kompilator włącza opcje na podstawie których dokonywana jest kompilacja; otwierane są plik(i) źródłowe.

2. Uruchamiany jest tzw. preprocesor, który przetwarza wszystkie dyrektywy w pliku/ach źródłowym/ch poprzedzone hash'em: '#' tzn.:
    - "wkleja" pliki nagłówkowe (*.h i *.hpp), które zadeklarowaliśmy słowem kluczowym `#include`; jeżeli zadeklarowany plik umieściliśmy w nawisach trójkątnych: '<>' to preprocesor szuka pliku we wszystkich lokalizacjach zapisanych przy opcji `-I` w pliku `bcc32.cfg` lub w parametrach wywołania; jeśli natomiast deklarowany plik został umieszczony w podwójnych cudzysłowach: "" to preprocesor szuka go w lokalizacji ze źródłami;
    - podmienia stałe i makropolecenia `#define`
    - wykonuje inne dyrektywy preprocesora takie jak `#pragma`
    - na podstawie dyrektyw: `#ifdef`, `#ifndef`, `#else` i `#endif` decyduje, które kawałki kodu będą kompilowane; jest to tzw. kompilacja warunkowa
3. Następnie program wyszukuje i wypisuje na standardowe wyjście ewentualne błędy i ostrzeżenia
4. Jeśli program nie ma błędów dokonywana jest właściwa kompilacja i jeżeli podczas kompilacji nie wystąpiły błędy w miejscu każdego pliku źródłowego (*.cpp lub *.c) podanego w parametrach wywołania generowane są pliki: *.obj (obiekty)
5. uruchamiany jest konsolidator, czyli program, który łączy pliki *.obj z bibliotekami (*.lib) i zasobami programu (*.RES) (tylko w przypadku programów pod Windows) w program (*.exe) lub bibliotekę dynamiczną (*.dll) w zależności od opcji podanych przy wywołaniu

Na podstawie rozszerzenia plików źródłowych podanych w parametrach wywołania BCC32, kompilator rozpoczyna operacje od punktu 1 (pliki *.c lub *.cpp) albo od punktu 5 (pliki *.obj i *.lib).

Najprostsza kompilacja

Po takiej dawce teorii pora na praktyczne zastosowanie kompilatora BCC32. Skopiujcie tradycyjny kod *Hello World* do notatnika:

```
#include <iostream.h>

int main()
{
  cout << "Hello World z Borland C++ Compiler!";
  getchar();
 return 0;
};
```

Zapiszcie ten program jako `hello.cpp` do katalogu `c:\kod`. Teraz wystarczy w wierszu poleceń "wejść" w tą lokalizację:

```
c:
cd c:\kod
```

skompilować plik:

```
bcc32 hello.cpp
```

...i `hello.exe` jest już gotowy do odpalenia. Prawda że proste? ;)

Edytory

Ten sposób kompilacji jest prosty, ale sprawdza się tylko w przypadku małych programów. Pisanie w mało

zaawansowanym edytorze takim jak Notatnik bywa uciążliwe. Przed prawdziwym zajęciem się programowaniem z narzędziami firmy Borland warto postarać się o jakiś zaawansowany edytor mający cechy IDE, czyli z podświetlaną składnią C++, makrami obsługującymi kompilator itp. Ja osobiście używam SynTextEditor (http://syn.sourceforge.net). Atutami tego programu są przede wszystkim duża uniwersalność i rozbudowane makra. Polecam też IDE Relo (http://www.fifsoft.com/relo/), jest świetnie przystosowane do obsługi BCC32, jednak nie pozwala na taką swobodę jak SynTextEditor. Wielu programistów używa też edytora V-IDE (http://www.objectcentral.com/vide.htm). Spis również innych edytorów przystosowanych do pracy z BCC32 jest na stronie: http://personal.sirma.bg/Jogy/bcc55.html

W internecie jest wiele edytorów dla programistów wystarczy je tylko poszukać i znaleźć coś dla siebie.

</noinclude>

BCC32

Wywołanie

```
BCC32 [<opcje>] <src>
```

`<opcje>`
: określamy sposób kompilacji wpisując odpowiednie symbole znakowe (ich opis jest poniżej); sam BCC32 ma jakieś 230 opcji więc jest w czym wybierać ;) muszą być one jedna od drugiej oddzielone spacją.

`<src>`
: tu umieszczamy wszystkie pliki źródłowe *.cpp lub (*.obj i *.lib) w zależności na jakim etapie kompilacji jesteśmy oddzielone spacjami.

Zakładając, że jesteśmy w katalogu ze źródłami przykładowe wywołanie BCC32 może wyglądać tak:

```
bcc32 -M -v -tW Hello.cpp klasa.cpp
```

Opcje

Źródła

opcja	opis
`-P`	kompiluj wszystkie pliki jako źródła C++ bez znaczenia na ich rozszerzenia
`-H=<nazwa>`	zdefiniuj nazwę `<nazwa>` pliku nagłówkowego
`-Hh=<plik>`	zaprzestań prekompilacji po pliku nagłówkowym `<plik>`
`-E<nazwa>`	zdefiniuj nazwę asemblera
`-T<opcja>`	poślij opcje `<opcja>` do asemblera
`-I<lokalizacja/e>`	możesz tutaj zdefiniować dodatkową lokalizacje (poza tymi wpisanymi w `bcc32.cfg`) plików nagłówkowych (jeżeli jest ich więcej niż jedna, to użyj ';' jako seperatora)
`-L<lokalizacja/e>`	to samo co wyżej tyle że z bibliotekami
`-v`	kompilacja w trybie DEBUG (jeżeli masz zamiar debugować program musisz dodać tę opcję)

Wynik

opcja	opis
`-tW`	aplikacja Win32 (EXE)
`-tWM`	aplikacja wielowątkowa
`-tWC`	program konsolowy dla Windows (EXE)
`-tWD`	biblioteka DLL
`-e<nazwa>`	zdefiniuj nazwę pliku EXE jaki ma być generowany
`-o<nazwa>`	zdefiniuj nazwę pliku OBJ jaki ma być generowany
`-l<opcja>`	poślij opcje `<opcja>` do wywołania konsolidatora (vel linkera)
`-S`	zamień podane źródło w C/C++ na kod asemblera
`-c`	kompiluj do OBJ, nie wywołuj konsolidatora
`-n<lolkalizacja>`	zapisz plik(i) wynikowe kompilacji do katalogu `<lokalizacja>`

Optymalizacja

Kompilator BCC32 pozwala nam na optymalizację naszego kodu źródłowego:

opcja	opis
`-Oc`	eliminuj duplikaty deklaracji
`-O1`	optymalizuj pod względem wielkości pliku wynikowego
`-O2`	optymalizuj program pod wzg. szybkości działania
`-G`	połączenie opcji `-O1` i `-O2`
`-Od`	nie optymalizuj

Wyjście

Podczas kompilacji w linii poleceń pojawia się wiele komunikatów/ostrzeżeń, na które nie musisz zwracać uwagi (ale często są one bardzo pomocne w poszukiwaniu błędów w kodzie źródłowym).

opcja	opis
`-q`	pomijaj nagłówek kompilatora
`-w`	pokazuj wszystkie ostrzeżenia (używaj gdy w kodzie źródłowym jest błąd)
`-w<xxx>`	pokazuj ostrzeżenie `<xxx>` (zajrzyj do manuala po liste ostrzerzeń)
`-w-<xxx>`	nie pokazuj ostrzeżenia `<xxx>` (zajrzyj do manuala po liste ostrzerzeń)
`-w-par`	nie pokazuj ostrzeżenia *"parametr, parametr is never used"*
`-w-rvl`	nie pokazuj ostrzeżenia *"function shuld return a value"*

Response & configuration files

Gdyby tak wpisywać te wszystkie opcje przy każdorazowej kompilacji, to człowiek by zwariował :), dlatego dla wygody programisty twórcy FCLT wymyślili tzw. ***configuration files***, w których możesz umieszczać standardowe (dla Ciebie) opcje BCC32. Już jeden taki plik stworzyłeś podczas instalacji kompilatora, nosił on nazwę `bcc32.cfg` i jego zawartość jest "wklejana" za każdym razem w parametry wywołania BCC32. Zaglądając do niego możesz dodać swoje osobiste opcje, które chcesz aby były dodawane przy każdym wywołaniu BCC32. Jeżeli chcesz, możesz zrobić jakieś dodatkowe pliki *.cfg i wtedy inaczej kompilować

jedne pliki, a inaczej drugie. Warto np. utworzyć pliki:

- `win.cfg`
- `dos.cfg`

i umieścić tam odpowiednie opcje w zależności pod jaki system chcemy kodować. Kompilacja z uwzględnieniem danego pliku konfiguracyjnego (oczywiście `bcc32.cfg` nie musisz już dodawać) powinno odbywać się w następujący sposób:

```
bcc32 +[<lokalizacja>\]<nazwa>.cfg [<opcje>] <src>
```

Na przykład:

```
bcc32 +C:\win.cfg -q Hello.cpp
```

Taki `win.cfg` może przykładowo zawierać linijkę:

```
-tW -c -w-par -w-rvl
```

Poza *configuration files* mamy również możliwość utworzenia ***response file*** (z ang. plik reagowania czy jakoś tak :/). Różnią się one od plików konfiguracyjnych tym, że można w nich umieszczać dodatkowo nazwy plików, które chcesz skompilować, więc używaj ich przy pracy z większymi projektami, gdy nie chce Ci się przy każdym wywołaniu wpisywać nazwy wszystkich opcji i plików wchodzących w skład projektu. Dołączenie *response file'a* wygląda tak:

```
bcc32 @[<lokalizacja>\]<plik>
```

Na przykład:

```
bcc32 @C:\projekt\project.txt
```

Wtedy w pliku `project.txt` możesz wpisać:

```
-tW -c -q -w-par -w-rvl Hello.cpp klasa.cpp
```

BCC32 i ILINK32

Jak wiemy z rozdziału 1-ego BCC32 sam wywołuje ILINK32 (konsolidator). Dzięki temu proces konsolidacji programu możemy również przeprowadzić przy pomocy BCC32. Czasami jest to niezbędne gdy chcemy w programie użyć jakiś dodatkowych bibliotek. W takim przypadku postępujemy wg schematu:

1. kompilujemy pliki *.cpp z opcją `-c`
2. wywołujemy BCC32 podając w parametrach wywołania wszystkie pliki *.obj jakie otrzymaliśmy po etapie kompilacji oraz dodatkowe biblioteki które chcemy użyć w programie

A oto przykład:

kompilacja (pkt. 1):

```
bcc32 -c hello.cpp
```

konsolidacja (pkt. 2):

```
bcc32 hello.obj shell32.lib user.lib
```

Kompilator automatycznie pośle to polecenie do konsolidatora, a my wcale nie musimy uczyć się składni ILINK32!

Zasoby?

Jeśli chcemy dołączyć zasoby do swojego programu to musimy napisać odpowiedni skrypt z rozszerzeniem *.rc, a następnie skompilować go narzędziem BRCC32 z FCLT:

```
brcc32 <zasoby>.rc
```

gdzie `<plik>` oznacza nazwę nie skompilowanego skryptu zasobów (bez rozszerzenia). BRCC32 wygeneruje wówczas plik `<zasoby>.RES`. Trzeba pamiętać, żeby w głównym pliku źródłowym projektu (tam gdzie znajduje się funkcja `WinMain()`) na samym początku dopisać linijkę z deklaracją tego pliku:

```
#pragma resource "<zasoby>.RES"
```

Teraz można już spokojnie zacząć kompilację wiedząc, że zasoby będą dołączone do pliku EXE.

ILINK32

Wprowadzenie

Tym razem zajmiemy się bardziej zaawansowanym narzędziem do programowania, czyli ILINK32. Jest to jak sama nazwa wskazuje linker (bardziej po polsku: konsolidator). Linker, czyli program, który łączy wszystkie "półprodukty" utworzone z plików źródłowych - pliki *.obj z bibliotekami, zasobami programu (o ile je uprzednio stworzymy) itp. Tak naprawdę bez konsolidatora nie byłoby programu takiego, jaki sobie wyobrażamy, ponieważ to on łączy wszystkie elementy na które składa się program w jeden plik EXE (lub w bibliotekę). Mniej doświadczonych programistów już teraz ostrzegam, że bawienie się w samodzielne wywoływanie ILINK32 będzie od was wymagało dodatkowej roboty, którą wcześniej "odwalał" za was kompilator (BCC32), ale ILINK32 w zamian daje jeszcze większy wpływ na to, jak wasz ostateczny program będzie wyglądał.

Wywołanie

W przypadku ILINK32 wywołanie jest znacznie bardziej skomplikowane niż w BCC32:

```
ILINK32 [<opcje>] <rozbiegówka> <obiekt(y)>, [<wynik>], [<mapa>], <lib>, [<defile
```

`<opcje>`
: tak jak w przypadku BCC32, tak samo konsolidacji możemy nadać wiele opcji, z których ważniejsze opisuje w następnym podrozdziale.

`<rozbiegówka>`
: spotkałem się kiedyś z takim określeniem i skoro sam nie potrafię tego jakoś sensownie nazwać będę używał tego terminu; `<rozbiegówka>` to specjalny plik *.obj, który musisz dodać do Twojego kodu w zależności jaką aplikację chcesz uzyskać:
  - `c0w32.obj` jeśli ma to być zwykła aplikacja Win32 GUI (*Graphic User Interface*)
  - `c0x32.obj` jeśli chcesz utworzyć program uruchamiany w konsoli
  - `c0d32.obj` jeśli masz zamiar wygenerować bibliotekę Win32 DLL

  W przypadku kompilacji przez BCC32 nie musisz o tym pamiętać, kompilator automatycznie dodaje ten plik po wpisaniu odpowiedniej :opcji, jeśli sam wywołujesz ILINK32 - Ty musisz zrobić.

`<obiekt(y)>`
: plik(i) *.obj generowane przez kompilator wpisujesz tutaj oddzielając każdy od siebie spacją. Jeśli ktoś nie wie, jak zmusić kompilator do kompilacji pliku tylko do *.obj (bez samodzielnego wywoływania ILINK32), to przypominam, że wystarczy dodać do linii poleceń BCC32 opcję -c

`<wynik>`
: możesz tu sam zdefiniować nazwę swojego pliku wynikowego; może to być zarówno program (*.exe) jak i biblioteka statyczna (*.lib) lub dynamiczna (*.dll)

`<mapa>`
: w tym miejscu możesz wpisać nazwę pliku MAP, który wygeneruje ILINK32 (jeżeli tego nie zrobisz linker nazwie go tak jak twój wynik (obcinając rozszerzenie). Plik MAP przechowuje adresy poszczególnych segmentów aplikacji i adresy funkcji Twojego programu, jest tworzony domyślnie.

`<lib>`
: po prostu biblioteka/i które masz zamiar użyć w programie; możesz dodać każdą o ile znajduje się w lokalizacji podanej przy opcji `-L` w parametrach wywołania ILINK32 lub w pliku `ilink32.cfg`. Dokumentacja FCLT określa prawidłową kolejność w jakich powinno się te biblioteki podawać:

1. bilioteki pomocne w analizowaniu i debugowaniu kodu (tzw. *code guards*)

2. inne (własne) biblioteki
3. biblioteka `import32.lib` - jeśli ma to być program dla Windows
4. biblioteki operacji matematycznych
5. tzw. *runtime libaries* (taka rozbiegówka, ale tym razem z rozszerzeniem lib) w przypadku programów dla Windows będzie to `cw32.lib`

   w praktyce aby skompilować zwykły program dla Windows należy podać tu przynajmniej: `import32.lib cw32.lib`

`<defile>`
: tutaj określasz plik DEF; w nim możesz określić wiele parametrów Twojego programu, które dokładnie opisuje poniżej.

`<zasoby>`
: zasoby, ale w wersji skompilowanej (*.res) - wcześniej musisz użyć kompilatora zasobów

Bardzo ważną rzeczą w wywoływaniu ILINK32 są **przecinki**, nawet jeśli nie chcesz wpisywać wszystkich parametrów masz obowiązek zaznaczyć wszystkie przecinki! Oto przykładowe wywołanie ILINK32 generujące aplikację Win32 GUI:

```
ILINK32 /aa /Tpe /Gk /t /M c0w32.obj Helo.obj klasa.obj,\
Hello.exe,,import32.lib cw32.lib,,zasoby.RES
```

Opcje

Źródła

opcja	opis
`@<nazwa>`	tu możesz zdefiniować nazwę `<nazwa>` response file dla ILINK32
`/j<lokalizacja>`	(dodatkowa) lokalizacja plików *.obj
`/L<lokalizacja>`	dodatkowa lokalizacja bibliotek

Wynik

opcja	opis
`/aa`	aplikacja *Windows GUI*
`/ap`	aplikacja *Windows Console*
`/ad`	sterownik urządzenia
`/Tpe`	program *.exe
`/Tpd`	biblioteka dynamiczna *.dll
`/Gl`	biblioteka statyczna (*.lib)
`/Gn`	ILINK32 przy każdej konsolidacji generuje 4 pliki (*.ilc, *.ilf, *.ild, *.ils), dzięki którym szybciej konsoliduje Twój program, jeżeli nie chcesz aby zaśmiecał Ci tym katalogu użyj tej opcji
`/Gk`	kiedy pojawią się jakieś błędy podczas konsolidacji linker automatycznie usuwa plik wynikowy(!) co czasami może być niepożądaną operacją, aby temu zapobiec możesz użyć tej opcji

Plik MAP

ILINK32 pozwala na określenie stopnia zaawansowania pliku MAP:

opcja	opis
`/x`	nie generuj pliku MAP
`/M`	generuj mapę najmniej zaawansowaną
`/m`	generuj mapę publiczną, przydatną podczas debugowania programu
`/s`	generuj szczegółową mapę

Ostrzeżenia

opcja	opis
`-q`	pomijaj nagłówek ILINK32
`/t`	pokaż czas spędzony na linkowanie
`-w-dup`	nie pokazuj ostrzeżenia "duplicate symbol"

Plik DEF

Plik DEF określa wiele aspektów programu, które możesz zdefiniować w zwykłym pliku ASCII, w Notatniku (pamiętając, żeby plik ten miał rozszerzenie *.def). Aspekty te określamy wpisując odpowiednie selektory i w tej samej linii dopisując do nich wartości. Standardowo jeśli nie określimy przy wywoływaniu ILINK32 pliku DEF, będzie on wyglądał tak:

```
CODE      PRELOAD MOVEABLE DISCARDABLE
DATA   PRELOAD MOVEABLE MULTIPLE   ; (for applications)
      PRELOAD MOVEABLE SINGLE   ; (for DLLs)
HEAPSIZE       4096
STACKSIZE      1048576
```

Już w tym kodzie widzisz 4 selektory i określone dla nich wartości. Są to podstawowe oznaczenia pliku DEF (`CODE`, `DATA`, `HEAPSIZE` i `STACKSIZE`). Od razu można się domyślić co one oznaczają, ale aby zbytnio nie mącić przejdę teraz do omówienia tych i innych parametrów pliku DEF.

CODE

Jak wiemy plik EXE jest podzielony na dwa podstawowe segmenty:

- kod (ciąg instrukcji np. w C++ które sprawiają, że program "żyje")
- zasoby (ikony, bitmapy, menu itp.)

Selektor CODE pliku DEF określa specyficzne opcje dla tego pierwszego segmentu (kod):

wartość selektora	standard	opis
PRELOAD	nie	kod jest wczytywany, gdy wywoływany program jest wczytywany
LOADONCALL	tak	kod jest wczytywany, kiedy jest wywoływany przez program
EXECUTEONLY	nie	kod może być tylko wykonywany
EXECUTEREAD	tak	kod może być czytany i wykonywany
FIXED	tak	segment pozostaje w pamięci
MOVEABLE/	nie	segment może być przeniesiony
DISCARDABLE	nie	segment może być odrzucony kiedy nie jest potrzebny (zakłada opcję MOVEABLE).
NONDISCARDABLE	tak	segment nie może być odrzucony

kolumna standard określa domyślną konfiguracje binarki

DATA

Parametr DATA określa zachowanie się segmentu zasobów:

wartość	standard	opis
NONE	nie	nie ma segmentu DATA
SINGLE	tak dla DLL	jeden segment zasobów jest tworzony i współdzielony dla wszystkich procesów
MULTIPLE	tak dla EXE	każdy segment zasobów jest tworzony dla jednego procesu
READONLY	nie	segemnt może być tylko czytany
READWRITE	tak	segment może być i czytany i nadpisywany
PRELOAD	nie	zasoby są wczytywane kiedy moduł, który ich używa jest wczytywany
LOADONCALL	tak	segment zasobów jest wczytywany kiedy jest po raz pierwszy przetwarzany (ta opcja jest ignorowana przez aplikacje 32-bitowe)
SHARED	nie	jedna kopia jest współdzielona między wszystkie procesy
NONSHARED	tak dla DLL	kopia segmentu jest wczytywana dla każdego procesu potrzebującego zasoby

HEAPSIZE

Jak sama nazwa wskazuje wpisujesz tutaj wielkość sterty (standardowo: 4096). Jeżeli w Twoim programie jest wiele obiektów zadeklarowanych na stercie możesz zwiększyć tę liczbę.

STACKSIZE

Tym razem określamy wielkość stosu (tu znajdują się wszystkie zmienne lokalne i parametry funkcji), dlatego wielkość tego parametru powinna być nieporównywalnie większa (standardowo: 1048576).

NAME

Nazwa pliku *.exe, który jest wynikiem konsolidacji, dodatkowo możesz tu umieścić po spacji jeden z dwóch znaczników:

`WINDOWAPI`
: jeżeli Twoja aplikacja będzie korzystać z WinAPI (to samo, co opcja linkera `/aa`)

`WINDOWCOMPAT`
: jeżeli Twój program będzie uruchamiany w konsoli (to samo, co opcja linkera `/ap`)

LIBRARY

Nazwa biblioteki DLL, która jest wynikiem konsolidacji i tutaj masz do wyboru 2 znaczniki:

`INITGLOBAL`
: biblioteka jest wczytywana do pamięci tylko raz

`INITINSTANCE`
: biblioteka jest wczytywana za każdym razem kiedy jest potrzebna

EXPORTS

Tego selektora mogą używać tylko biblioteki Jego wartością są nazwy wszystkich funkcji udostępniane przez bibliotekę. Więcej o tym selektorze przeczytasz w artykule: Biblioteki DLL w FCLT (http://www.viilo.torun.pl/~re_set/txt/dll.html)

IMPORTS

Wartością tego selektora są funkcje importowane z bibliotek zadeklarowane w kodzie źródłowym. I o tym selektorze przeczytasz w artykule o bibliotekach DLL (http://www.viilo.torun.pl/~re_set/txt/dll.html).

DESCRIPTION

Ujęty w pojedynczy cudzysłów: *opis programu (mogą tu się znajdować również np. prawa autorskie do programu). Oto jak może wyglądać plik*

DEF w praktyce:

```
NAME        Hello WINDOWAPI
DESCRIPTION '(c) 2003 by Karol Ossowski'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE

HEAPSIZE       4096
STACKSIZE      1048576
```

Zasoby

Jak już pisałem aby sprowadzić skrypt zasobów z rozszerzenia *.rc do skompilowanego pliku *.RES należy użyć brcc32 z katalogu BIN. Wywołanie brcc32 będzie wyglądać przykładowo tak:

```
brcc32 <zasoby>.rc
```

Nie będę się zagłębiał tutaj w opcje, jakich może użyć ten kompilator (jeśli Cię one interesują możesz sprawdzić je w dokumentacji FCLT), ważne że prze konwertuje on nasz plik `<zasoby>.rc` na

<zasoby>.RES. Dopiero teraz możesz go dołączyć do wywołania ILINK32.

MAKE

Wprowadzenie

MAKE pozwala zautomatyzować proces budowania projektu programistycznego poprzez stawianie odpowiednich zasad dla plików źródłowych i wywoływaniu komend systemowych. Poza tym program ten znacznie przyspiesza proces budowy programu, ponieważ wykonuje operacje tylko na plikach, które zostały zmodyfikowane od czasu ostatniej kompilacji, ma to szczególnie duże znaczenie w przypadku dużych projektów.

Plik Makefile

Najważniejszy plik dla MAKE. W nim znajdują się wszelkie zasady operacji na źródłach, wywołania programów, zmienne i instrukcje odpowiedzialne za zarządzanie budową projektu. Utworzyć go można w najprostszym edytorze tekstowym (np. w systemowym Notatniku). Należy pamiętać jednak, aby plik zapisać jako `makefile.mak` lub `makefile` (bez rozszerzenia). To, którą opcję wybierzesz jest już obojętne.

Wywołanie

Wywołanie MAKE jest nieco inne niż pozostałych narzędzi Borlanda. W tym przypadku kluczową rolę odgrywa lokalizacja na dysku, z której wywołujesz program. W lokalizacji tej musi się znajdować plik makefile wraz ze skopiowanym (z katalogu \bin naszych FCLT) `make.exe`. To ważne! Inaczej programy wywoływane z makefile'a nie będą działały poprawnie. Będziesz wywoływać właśnie tę kopię make'a, więc najpierw trzeba wejść w jej lokalizację:

```
CD <lokalizacja>
MAKE [<opcje>] [<wynik>]
```

`<opcje>`
: MAKE ma mało opcji, a ponadto w większości z nich można zastąpić odpowiednimi komendami w makefile'u; więcej na ten temat w rozdziale: Plik projektu i konfiguracja MAKE

`<wynik>`
: czyli plik(i) wynikowy/e Twojego projektu, pamiętaj że nie musisz ich tutaj wymieniać

w praktyce wywołanie MAKE może wyglądać tak:

```
CD c:\Hello
MAKE -a -s
```

Składnia makefile'a

Składnia makefile to już swego rodzaju język programowania do budowy projektów programistycznych. Możemy rozróżnić jej kilka podstawowych elementów:

komentarze
: po prostu komentarze, tak jak w innych językach programowania

zasady
: zasady wg których MAKE dobiera pliki źródłowe i wynikowe po wypełnieniu których wywołuje...

komendy systemowe "siłą" MAKE jest możliwość wywoływania nie tylko narzędzi Borland-a, ale także komend DOS'owych i innych programów takich jak np. kompilator assemblera.

zmienne
: tak jak w każdym języku programowania "kontenery" do których możemy przypisać łańcuchy znaków.

dyrektywy
: chociaż będę je opisywał na końcu nie powinny być lekceważone, to jedne z najważniejszych elementów w każdym języku programowania, tak samo jest i tutaj.

A tak wygląda przykładowy makefile, z którego korzystasz na co dzień prawdopodobnie nawet o tym nie wiedząc, czyli `builtins.mak` z BIN'a:

```
#
# Inprise C++Builder - (C) Copyright 1999 by Borland International
#
CC       = bcc32
RC       = brcc32
AS       = tasm32

!if $d(__NMAKE__)
CXX      = bcc32 -P
CPP      = bcc32 -P
!endif

.asm.obj:
      $(AS) $(AFLAGS) $&.asm

.c.exe:
      $(CC) $(CFLAGS) $&.c

.c.obj:
      $(CC) $(CFLAGS) /c $&.c

.cpp.exe:
      $(CC) $(CFLAGS) $&.cpp

.cpp.obj:
      $(CC) $(CPPFLAGS) /c $&.cpp

.rc.res:
      $(RC) $(RFLAGS) /r $&

.SUFFIXES: .exe .obj .asm .c .res .rc

!if !$d(BCEXAMPLEDIR)
BCEXAMPLEDIR = $(MAKEDIR)\..\EXAMPLES
!endif
```

Czarna magia? W następnych podrozdziałach będzie mowa o najważniejsze elementach składni pliku makefile.

Aby to co pisze nie było tylko czystą teorią wymyśliłem projekt *HelloProject*, którego makefile będziemy rozwijać w miarę postępów w nauce. Jak zorganizować sobie miejsce pracy nad tym projektem? Należy utworzyć nowy katalog z nazwą projektu: `C:\Hello\` Następnie trzeba tam skopiować wszystkie jego pliki źródłowe:

- `Hello.cpp`
- `klasa.cpp`
- `klasa.h`
- `zasoby.rc`
- `DEF.def`
- `makefile.mak`

Wynikiem tego projektu będzie windowsowy EXE'k: `Hello.exe`. Nieważne, co będzie robił ten program, to tylko ćwiczenie, resztę pozostawiam Tobie, Twojej wyobraźni i inwencji. MAKE będziemy wywoływać tam, gdzie jest makefile, czyli z katalogu `c:\Hello\`. Tam też powinna znaleźć się kopia programu MAKE.

Komentarze

W makefile'u tak jak w C++, również możesz używać komentarzy jedno liniowych. Wszystko to co znajdzie się w jednej linii po symbolu hash'a: # MAKE ignoruje.

```
# to jest komentarz
```

Zasady ogólne i szczegółowe

Zasady regulują wywoływanie narzędzi w makefile'u. Ogólnie rzecz biorąc zasady określają jakie pliki ma przetwarzać MAKE, ewentualnie gdzie one się znajdują, gdzie ma się znaleźć wynik operacji. Ważną kwestią w zrozumieniu zasad jest fakt, że zasada to tylko prawo, za wypełnienie którego odpowiedzialny jesteś Ty. Pamiętaj też że zasada przekazuje tylko informacje dla MAKE, natomiast komendy, które za jej pomocą wywołujesz rządzą się już swoimi prawami. Zarówno przy komendach, jak i przy samych zasadach można używać symbolu \ jeśli zabraknie Ci miejsca w linii:

```
<txt>\
<dok. txt'u>
```

Ogólny szablon zasady wygląda tak:

```
<zasada>
 <komenda>
 [<komenda>]
 [...]
```

Pamiętaj tylko o przynajmniej jednej spacji przed komendą, aby nie znajdowała się ona tuż przy początku linii - to błąd! Błędem jest też nie zapisanie samej zasady przy początku linii. Jednym słowem musi to wyglądać tak, jak na powyższym szablonie. Zasada zacznie "wchodzić w życie" (wywoływać swoje komendy), jeśli jej *pliki zależne* (źródłowe) będą różniły się datą i czasem ostatniej modyfikacji z datą *plików wynikowych* lub jeśli *pliki wynikowe* w ogóle nie będą istnieć. Np. jeśli chcemy skompilować plik `Hello.cpp` za pomocą zasad to kompilator zostanie wywołany kiedy `Hello.cpp` i `Hello.obj` będą miały inne daty modyfikacji lub jeśli plik `Hello.obj` nie będzie istniał. Jak widać w tytule tego rozdziału zasady możemy podzielić na dwa typy: ogólne i szczegółowe (*implicit & explicit rules*).

Zasady ogólne (*implicit rules*)

Zasady te tyczą się wszystkich plików z określonym rozszerzeniem (ewentualnie możesz określić folder, w

którym MAKE ma zezwalać na operacje na nich). Zasady te zakładają tylko zmianę rozszerzenia, a nazwy plików pozostawiają. *Implicit rule* wygląda tak:

```
[{<lokalizacja1>}].<roz1>[{<loklalizacja2>}].<roz2>:
 <komenda/y>
```

gdzie:

`<lokalizacja1>`
: lokalizacja/e plików zależnych (źródłowych) (w przypadku gdy jest ich więcej jako separatora używamy średnika:';')

`<roz1>`
: rozszerzenie plików źródłowych

`<lokalizacja2>`
: lokalizacja/e na dysku plików wynikowych (docelowych) operacji (gdzie mają się one znaleźć) oddzielone jedna od drugiej średnikiem:';'

`<roz2>`
: rozszerzenie plików wynikowych

Na przykład:

```
{C:\src}.cpp{C:\obj}.obj:
 bcc32 -c c:\src\hello.cpp
 copy hello.obj c:\obj
```

Przejdźmy teraz do analizy tych trzech linijek kodu. Pozwoli Ci to lepiej zrozumieć działanie zasad make'a. W tym przypadku zasada zezwala na operacje na plikach *.cpp tylko z katalogu `c:\src`, a ich wynik w magiczny sposób musi znaleźć się w `c:\obj`, oczywiście nie da się tego dokonać bez pomocy systemowego słowa kluczowego COPY albo użycia w parametrach wywołania kompilatora opcji -n. Jeśli nie określisz w zasadzie tych lokalizacji, to dozwolone będą operacje na wszystkich plikach na całym Twoim twardzielu. Określając w zasadzie lokalizacje plików źródłowych nie łudź się że zostanie to przekazane komendom które wywołujesz, dlatego podczas kompilacji zapisałem plik `hello.cpp` wraz jego z lokalizacją, oczywiście nie musiałbym tego robić, gdybyśmy odpalili MAKE z lokalizacji `c:\src`.

Zasady szczegółowe (*explicit rules*)

Zasady szczegółowe określają z jakich plików źródłowych powstaje plik wynikowy. W tym przypadku musimy podać dokładne nazwy plików a nie jak to było w przypadku zasad ogólnych samego rozszerzenia plików. Spójrzmy na składnię:

```
<wynik(i)>:[:] [{<lokalizacja>}]<src>
 <komenda/y>
```

`<wynik(i)>`
: nazwa pliku/ów wynikowego/ych, który ma być wygenerowany

`<lokalizacja>`
: lokalizacja/e plików źródłowych oddzielone od siebie średnikami:';'

`<src>`
: plik(i) źródłowy/e

Przykład powinien rozwiać wszelkie wątpliwości:

```
Hello.exe: {C:\obj}Hello.obj klasa.obj
 ilink32 /aa /Tpe c0w32.obj C:\obj\Hello.obj C:\obj\klasa.obj,\
Hello.exe,,cw32.lib import32.lib,,
```

Zasada zakłada że `Hello.obj` i `klasa.obj` znajdują się tylko i wyłącznie w katalogu `c:\obj`. Tak jak w zasadach ogólnych tak i tutaj ta lokalizacja nie ma nic wspólnego z komendami - i w tym przypadku musisz podać lokalizacje każdego ze swoich plików źródłowych (oczywiście jeśli znajdują się one tam gdzie makefile nie musisz tego robić). Zmieni się to dopiero po wprowadzeniu do kodu inteligentnych zmiennych które będą potrafiły czerpać informacje z zasad, ale o tym później.

Wszystkie pliki docelowe (wynikowe) z zasad ogólnych powinny być automatycznie plikami źródłowymi w zasadzie szczegółowej. Inaczej MAKE danej zasady ogólnej nie weźmie pod uwagę. Pokaże to na przykładzie:

```
.cpp.obj:
 bcc32 -c Hello.cpp

.rc.res:              # MAKE nie zwraca na tą zasadę uwagi, bo nie dodałeś
 brcc32 zasoby.rc     # pliku zasoby.res do src zasady szczegółowej

Hello.exe: Hello.obj
 ilink32 /aa /Tpe c0w32.obj Hello.obj klasa.obj,Hello.exe,,\
cw32.lib import32.lib,,zasoby.res
```

W tym przypadku konsolidator zgłosi błąd, że nie może znaleźć pliku zasoby.res, nic dziwnego - BRCC32 wcale nie został wywołany.
MAKE zakłada, iż zawsze dysponujemy tylko jednym plikiem wynikowym, ale i temu można zaradzić tworząc symboliczne pliki wynikowe o tak:

```
ALL: <PlikWynikowy1> <PlikWynikowy2> [<Plikwynikowy3> [...]]
```

Teraz nic nie stoi na przeszkodzie, aby wcześniejszy niefortunny kod zamienić na poniższy:

```
ALL: zasoby.res Hello.exe

.cpp.obj:
 bcc32 -c Hello.cpp

zasoby.res: zasoby.rc
 brcc32 zasoby.rc

Hello.exe: Hello.obj
 ilink32 /aa /Tpe c0w32.obj Hello.obj klasa.obj,Hello.exe,,\
cw32.lib import32.lib,,zasoby.res
```

Oczywiście to już od Ciebie zależy jakich zasad wolisz używać. Moim zadaniem jednak warto użyć wielu zasad ogólnych, a tylko jedną szczegółową, chyba że musisz w jednym projekcie mieć dwa pliki EXE (lub DLL). Jest też możliwość przypisania wielu zasad do jednego pliku wynikowego używamy wtedy podwójnego dwukropka:

```
mylib.lib:: obj1.obj obj2.obj
  tlib mylib +obj1.obj +obj2.obj
```

```
mylib.lib:: obj3.obj obj4.obj
  implib mylib +lib3.obj +lib4.obj
```

Najpierw do biblioteki statycznej mylib.lib dodawane są `obj1.obj` i `obj2.obj`, następnie w drugiej zasadzie do tej samej biblioteki oddajemy: `obj3.obj` oraz `obj4.obj`.

Zasada "bez zasad" :-)

Jest jeszcze jeden typ zasad. Właściwie to nie wiem, czy można by nazywać to zasadą, bo nie określa żadnych praw dostępu do plików. Jej składnia jest miej więcej taka:

```
<nazwa>:
 <komenda/y>
```

Przydaje się ona do wywołania narzędzi typu debugger lub TOUCH które nie czynią żadnych zmian w rozszerzeniu pliku zatem nie można ich normalnie wywołać z jakichkolwiek wcześniejszych zasad. Poza tym zasada ta zawsze "wejdzie w życie", gdyż nie mamy tu żadnych plików zależnych ani wynikowych. Pewnym jej ograniczeniem jest to że po zakończeniu jej działań MAKE także kończy. Cóż, nie można mieć wszystkiego ;). Dla przykładu podaje wywołanie programu TOUCH:

```
TOUCH:
 cd c:\Hello
 touch *.cpp *.rc
```

Komendy systemowe

Na co nam zasady skoro i tak nigdy nie zostaną poparte działaniami? - na nic. Dlatego, aby coś zaczęło działać trzeba to...wywołać :). Jak już pisałem oprócz zwykłych wywołań narzędzi z FCLT można tu też umieszczać komendy systemowe, czy włączać inne program takie jak np. kompilator assemblera, czy swój własny program już po skompilowaniu. Pamiętaj jednak, że komendy nie mogą występować w makefile'u od tak sobie, muszą one zawierać się w jakiejś zasadzie tylko w ten sposób można wywoływać narzędzia przez MAKE, inaczej program (MAKE) zgłosi błąd. Poza tym przy wywołaniu 'czegoś' można użyć następujących przedrostków:

`@`	wywołanie danego narzędzia nie ma być wyświetlane w linii poleceń, MAKE zawsze przed zastosowaniem jakiejś komendy najpierw wyświetla na ekranie jej wywołanie, ten przedrostek temu zapobiega
`-`	kontynuuj działnia nawet jeśli narzędzie zgłosi błędy np. zapis: `-bcc32 -c hello.cpp` sprawi, że nawet jeśli w kodzie źródłowym `hello.cpp` będzie błąd MAKE nie zakończy procesu budowy projektu.
`-<liczba>`	w miejsce `<liczba>` wpisz liczbę, po zwróceniu której make ma przestać dalej działać (wywoływać); np.: `-0bcc32 -c hello.cpp` da nam zakończenie działania programu gdy BCC32 zakończy działanie kodem wyjścia równym 0

Właściwie to już możesz spróbować napisać swój własny, prosty makefile, który zautomatyzuje wywołanie kompilatora, kompilatora zasobów i konsolidatora.

```
#----------------------------
# (c) 2003 by Karol Ossowski
#----------------------------
# >>   Hello Project     << #
#----------------------------

.cpp.obj:
 BCC32 -tW -c -q -w-par -w-rvl Hello.cpp klasa.cpp

.rc.res:
 BRCC32 zasoby.rc

Hello.exe: Hello.obj klasa.obj zasoby.res
 ILINK32 /aa /Tpe /Gk /t /q -w-dup c0w32.obj Hello.obj\
klasa.obj,Hello.exe,,cw32.lib import32.lib,DEF.def,\
zasoby.res
```

Ten makefile jest dość prymitywny, jednak automatyzuje on już całą budowę projektu *HelloProject*. Spróbuj go odpalić samemu.

Zmienne

W makefile'u możemy deklarować zmienne gdzie umieścimy pewne łańcuchy znaków, później te "stringi" (z ang. *string* znaczy łańcuch) będziemy mogli wykorzystać wpisując nazwę zmiennej. Przypisywanie wartości do zmiennej wygląda tak:

```
<nazwa> = [<łańcuch_znaków>]
```

Zwroć uwagę, że przypisywany łańcuch nie jest ujęty w "" tak jak to jest w C/C++, wszystko to co znajdzie się za znakiem równości: = w jednej linii zostanie przypisane do zmiennej <nazwa>. np.:

```
SRC = Hello.cpp klasa.cpp
```

Wykorzystanie tej zmiennej w makefile'u będzie za to przypominać języki skryptowe takie jak Perl czy Bash:

```
.cpp.obj:
 bcc32 -c $(SRC)
```

I tak każda zmienna przez Ciebie zadeklarowane będzie poprzedzane '$' i ujęte w nawiasy '()', ale analizując te dwa przykłady już możesz odkryć pożyteczność tego wynalazku: deklarując na początku zmienną `SRC` nie musisz pamiętać już wszystkich plików źródłowych, które wykorzystasz w swoim programie, nawet jeśli projekt się rozrośnie nie będziesz musiał dopisywać w każdym wywołaniu korzystającym ze źródeł jeszcze jednego pliku, ale wystarczy, że na samym początku dopiszesz jeszcze jeden plik *.cpp, a zmienna będzie wtedy wskazywała już na dwa pliki źródłowe.
Jak już pisałem zmienne to po prostu łańcuchy znaków, więc możemy przypisać im również opcje, jakie chcemy wykorzystać przy wywoływaniu narzędzi:

```
C_FLAGS = -tW -c -v -w-par -w-rvl
```

Teraz wywołanie BCC32 będzie w makefile'u wyglądać tak:

```
.cpp.obj:
 bcc32 $(C_FLAGS) $(SRC)
```

Zmiana wartości zmiennej

W każdym zapamiętanym w zmiennej "stringu" można dokonać zamiany "podstringów". Robimy to korzystając z następującego szablonu:

```
$(<nazwa_zm>:<stary_str>=<nowy_str>)
```

Na przykład można to wykorzystać do zmiany rozszerzenia plików z jakich korzystamy:

```
SRC = Hello.cpp klasa.cpp
$(SRC:.cpp=.obj)
```

Teraz nie musisz pisać już dodatkowej zmiennej z wynikami konslolidacji - zmienna `$(SRC)` możemy wykorzystać do wywołania kompilatora, a do wywołania konsolidatora - `$(SRC:.cpp=.obj)`.

Specjalne zmienne

Istnieją jeszcze zmienne specjalne, które są już stworzone przez MAKE i możesz je używać w komendach tuż po zasadach. Są one bardzo wygodne:

zmienna	przekazywane dane w zasadach ogólnych	przekazywane dane w zasadach szczegółowych
$*	plik źródłowy (bez rozszerzenia) z lokalizacją	plik wynikowy (bez rozszerzenia) z lokalizacją
$<	plik źródłowy+rozszerzenie z lokalizacją	plik wynikowy+rozszerzenie z lokalizacją
$:	lokalizacja plików źródłowych	lokalizacja pliku wynikowego
$.	plik źródłowy+rozszerzenie	plik wynikowy+rozszerzenie
$&	plik źródłowy	plik wynikowy
$@	plik wynikowy+rozszerzenie z lokalizacją	to samo co $<
$**	to samo co $<	wszystkie pliki źródłowe+rozszerzenie
$?	to samo co $<	stare pliki źródłowe

Przyznajcie, że zmieniają one wręcz filozofie zasad, które stają się nie tylko prawami dostępu do plików, ale i źródłem informacji dla komend np. o lokalizacji plików. Przykład użycia specjalnych makr:

```
{c:\cpp}.cpp.obj:
 bcc32 -q -c $<
```

ten kod spowoduje kompilacje wszystkich plików *.cpp w katalogu `c:\cpp`, a pliki *.obj, które wygeneruje umieści w lokalizacji w której znajduje makefile. To nie wszystko - zmiennym specjalnym można też zmieniać wartości stosując specjalne znaczniki. Znaczniki te są najczęściej używane przy zmiennych: `$@` i `$<`. Dodanie takiego znacznika modyfikującego wartość danej zmiennej powinno wyglądać tak:

```
$(<znak specjalnego makra>:[<znacznik>])
```

Oto typy znaczników do standardowych makr MAKE'a:

znacznik	przekazywane dane o pliku	przykład użycia	przykładowy rezultat dla pliku `C:\Proj\kod.cpp`
D	lokalizacja	$(<D)	C:\Proj\
F	nazwa i rozszerzenie	$(@F)	kod.cpp
B	nazwa	$(<B)	kod
R	lokalizacja,nazwa	$(@R)	C:\Proj\kod

i jeszcze przykład:

```
{c:\cpp}.cpp{c:\obj}.obj:
 bcc32 -q -c -n$(@D) $<
```

ten fragment skryptu spowodowałby kompilację plików *.cpp z katalogu `c:\cpp` i zapisywałby pliki *.obj do katalogu `c:\obj`.

Twoje zmienne powinny być w szczególności używane przy zasadach jakie będziesz stawiał, a zmienne specjalne w komendach np.:

```
$(BIN): $(SRC:.cpp=.obj)
 ILINK32 $(L_FLAGS) c0w32.obj $**,$.,,cw32.lib import32.lib $(LIB),$(DEF),$(RES)
```

Używając zmiennych `$**` lub `$?` w zasadzie szczegółowej (*explicit rule*) można do komendy dodać przedrostek **&**, który będzie oznaczał, że komenda odnosi się tylko do jednego pliku z pola `<src>` (normalnie makra `$**` i `$?` w zasadach szczegółowych oznaczają wszystkie pliki źródłowe, co uniemożliwi pracę niektórych komend) np.:

```
hello.exe: hello.obj klasa.obj
 copy $** c:\obj
```

Po takiej operacji komenda systemowa COPY zgłosi błąd, ponieważ `$**` wskazuje na pliki `hello.obj` i `klasa.obj`, a COPY może obsługiwać tylko jeden plik. Ten błąd da się naprawić:

```
hello.exe: hello.obj klasa.obj
 &copy  $** c:\obj
```

Teraz system wykona dwie operacje zamiast jednej:

```
copy hello.obj c:\obj
copy klasa.obj c:\obj
```

..aha przed przystąpieniem do kolejnego rozdziału rozwiniemy nasz *HelloProject* o obsługe zmiennych. Aby zachować pewien porządek w katalogu `c:\Hello` najpierw stworzymy następujące podkatalogi,a w nich zawrzemy pliki do projektu *HelloProject*:

`.\src\` <-- Hello.cpp klasa.cpp
`.\src\hdr\` <-- klasa.h
`.\src\res\` <-- zasoby.res
`.\obj\` <-- makefile.mak DEF.def make.exe *.obj *.li?
`.\bin\`

Teraz proszę zerknąć na ten kod:

```
#----------------------------
# (c) 2003 by Karol Ossowski
#----------------------------
# >>   Hello Project     << #
#----------------------------

DIR = c:\Hello

C_FLAGS = -tW -c -q -w-par -w-rvl
L_FLAGS = /aa /Tpe /Gk /t /q -w-dup

SRC = Hello.cpp klasa.cpp
HDR = klasa.h            # linijka odana dla porządku ;)
RES = zasoby.res
DEF = DEF.def
LIB =
BIN = Hello.exe

{$(DIR)\src}.cpp{$(DIR)\obj}.obj:
 BCC32 $(C_FLAGS) -I$(DIR)\src\hdr $<
```

```
{$(DIR)\src\res}.rc{$(DIR)\obj}.res:
 BRCC32 $<
 @COPY $:$(@F) $(DIR)\obj
 @DEL  $:$(@F)

$(BIN): {$(DIR)\obj}$(SRC:.cpp=.obj) $(RES)
 ILINK32 $(L_FLAGS) c0w32.obj $(SRC:.cpp=.obj),$.,,cw32.lib\
import32.lib $(LIB),$(DEF),$(RES)
 COPY $. $(DIR)\bin
 DEL $.
```

Jak widać makefile bardzo zyskał na funkcjonalności i co najważniejsze na uniwersalności: dzięki zmiennym cały czas możesz zmieniać skład plików projektu zasadniczo nie ingerując w zasady. Nawet jeśli w projekcie nie przewidujesz zasobów to nie będzie tragedii: operacja kompilacji zasobów nie będzie miała miejsca a plik nie zostanie dodany przez konsolidator. Poza tym zmienna `DIR` nie została stworzone bez powodu. Dzieki niej możliwa jest także budowa nowego, całkowicie innego projektu, którego wynikiem będzie `$(BIN)`, plikami źródłowymi `$(SRC)` itd. Oczywiście przy tej operacji trzeba zachować tę samą strukturę podkatalogów projektu i rozmieszczenia w nich typów plików.
Teraz małe sprostowanie tak dziwnego zapisu kompilacji zasobów: BRCC32 nie zachowuje się jak większość programów z FCLT ponieważ zostawia plik wynikowy w lokalizacji ze źródłami.
Jest pewien mankament tego makefile'a: jeśli nie masz plików zależnych zasady szczegółowej MAKE się o nie upomni, choć mają one dopiero powstać w trakcie trwania makefile'a. Jest na to rada: przed pierwszym (później nie ma już takiej potrzeby) uruchomieniem makefile'a "dotknij" programem TOUCH wszystkie pliki źródłowe. Np. w tym projekcie będzie to wyglądać tak:

```
cd c:\Hello\obj
touch zasoby.res klasa.obj Hello.obj
cd c:\Hello\src
touch *.cpp
cd .\res
touch *.rc
```

Można rozwiązać to w ten sposób... ale jest znacznie bardziej funkcjonalna dyrektywa...

Dyrektywy

Dyrektyw/słów kluczowych w borlandowskim makefile'u jest mniej niż np. w C++ ale wystarczająco dużo, aby efektywnie zarządzać projektem. W tym przypadku spisałem niemal wszystkie wyrażenia (chyba że sam któregoś nie rozumiałem). Nie musisz się ich uczyć na pamięć, wystarczy, że będziesz miał te 5 tabelek zawsze pod ręką. Warto jednak na początek je przeczytać, żeby przekonać się jakie są możliwości słów kluczowych w MAKE. Ważną zasadą przy wykorzystywaniu dyrektyw jest to aby MAKE nie pomylił danej dyrektywy z wywołaniem programu. Jeśli więc chcemy dyrektywa umieścić w polu komend jakiejś zasady to trzeba ją postawić **na początku wiersza** w innym przypadku zostanie ona potraktowana jako komenda systemowa, a to może oznaczać tylko kłopoty...

Pliki projektu i konfiguracja MAKE

nazwa	wiersz poleceń	opis
.autodepend	-a	sprawdzaj pliki nagłówkowe przed kompilacją i jeśli zostaną zmodyfikowane kompiluj jeszcze raz pliki które z nich korzystają
.noautodepend	-a-	nie sprawdzaj plików nagłówkowych
.cacheautodepend	-c	przechowuj w pamięci podręcznej pliki wchodzące w skład projektu i jeśli nie zostaną w poczynione żadne zmiany nie wykonuj na nich operacji ponownie
.nocacheautodepend	-c-	nie przechowuj w pamięci podręcznej plików wchodzących w skład projektu
.keep	-K	zachowuj pliki tymczasowe tworzone podczas działania programu MAKE
.nokeep	-K-	nie przechowuj plików tymczasowych które są tworzone podczas działania MAKE'a
.ignore	-i	ignoruj wartość jaką zwróci komenda
.noIgnore	-i-	nie ignoruj wartości jaką zwróci komenda
.silent	-s	nie pokazuj na ekranie wywołania narzedzia
.nosilent	-s-	pokazuj na ekranie wywołanie narzędzia lub komendę jaką wykonuje system
.swap	-S	wyczyść swoją pamięć zanim zaczniesz wywoływać narzędzia (ta instrukcja jest dobra podczas operacji na dużych plikach)
.noswap	-S-	nie czyść pamięci przed wywoływaniem narzędzi
	-r	ignoruje zasady jakie podaje nam standardowy plik makefile BUILTINS.mak znajdujący się w katalogu BIN kompilatora

szablon	opis
.precious: <PlikWynikowy>	jeśli któryś z programów "padnie" MAKE wyrzuca swój plik wynikowy. ta komenda temu zapobiega
.suffixes: .<roz1> [.<roz2> [.<roz3>]..]	twórz pliki najpierw z rozszerzeniem: <roz1> później z <roz2>, a na końcu z <roz3> (itd.); ta dyrektywa jest analizowana przez zasady ogólne i określa ona porządek tworzenia PlikówDocelowych
.path.<roz> = <lokalizacja>	szukaj pliku z rozszerzeniem <roz> w podanej <lokalizacji> (ta dyrektywa niweluje problem z wcześniejszym makefile'em)
!include [<lokalizacja>] <Nazwapliku>	dodaj tekst do obecnego makefile'a z pliku <NazwaPliku> (działa jak makro-instrukcja #include w C/C++)
!undef <nazwa_zmiennej>	"wyrzuć" zmienną <nazwa_zmiennej>

Instrukcje warunkowe

szablon	opis
`!ifdef <nazwa_zmiennej> <operacje>`	jeśli zmienna `<nazwa_zmiennej>` jest zadeklarowana wykonaj `<operacje>`
`!ifndef <nazwa_zmiennej> <operacje>`	jeśli zmienna `<nazwa_zmiennej>` nie jest zadeklarowane wykonaj `<operacje>`
`!if <warunek> <operacje>`	jeśli `<warunek>` zostanie spełniony wykonaj `<operacje>`
`!else <operacje>`	w przeciwnym wypadku wykonaj `<operacje>` (musi występować z `!if` lub `!ifdef` lub `!ifndef`)
`!elif <warunek> <operacje>`	w przeciwnym wypadku, jeśli `<warunek>` jest spełniony wykonaj `<operacje>` (musi występować z `!if` lub !ifdef, `!ifndef`, `!else`)
`!endif`	kończy instrukcję warunkową

Instrukcję warunkową warto bardziej wnikliwie zanalizować, ponieważ jak sądzę będziesz ją często używał(a).

- Każda instrukcja warunkowa musi się kończyć dyrektywą `!endif`
- Przy określaniu warunków instrukcji warunkowej możemy użyć między innymi następujących operatorów:

operator	opis	operator	opis
-	negacja	||	logiczne OR
+	dodawanie	!	logiczne NOT
-	odejmowanie	&&	logiczne AND
*	mnożenie	>=	większe lub równe
/	dzielenie	<=	mniejsze lub równe
==	równe	>	większe
!=	nierówne	<	mniejsze

- Aby sprawdzić czy zmienna jest zadeklarowana (tj. czy je wcześniej stworzyliśmy) można użyć specjalnego **znacznika d**. Zastosowanie go jest równoważne z użyciem dyrektywy !ifdef:
  - `!ifdef <zmienna>` to, to samo co `!if $d(<zmienna>)`
  - `!ifndef <zmienna>` to, to samo co `!if !$d(<zmienna>)`

Ekran

szablon	opis	rezultat
`!error <komunikat>`	polecenie, po natrafieniu na które MAKE kończy działanie i wyświetla na ekranie rezultat...	`Fatal makefile <numer_linii>: Error directive: <komunikat>`
`!message <komunikat>`	jeśli MAKE natrafi na to polecenie, wyświetla na ekranie rezultat...	`<komunikat>`

W obu dyrektywach "ekranowych" można używać zmiennych do reprezentacji komunikatów.

MAKE w praktyce

Teraz pora na wzbogacenie naszego makefile'a o dyrektywy i tym samym doprowadzenie skryptu do

ostatecznej wersji. Przeanalizuj ten kod, a na pewno rozjaśni Ci się w głowie, o czym była mowa przez ostatnie 25kB :). Do tego projektu potrzebujesz jeszcze jeden plik: `HelloProject.txt` który jako jedyny powinien się znaleźć w katalogu głównym projektu. Nasz ostateczny makefile będzie się bowiem składał z dwóch plików pierwszy to przed chwilą utworzony `HelloProject.txt`, a drugi to wszystkim znany `makefile.mak` (połączone są one dyrektywą `!include`). Dodatkowo w katalogu `c:\Hello\src` powinieneś utworzyć jeszcze jeden folder i zamieścić tam pliki *.asm:

.\asm <-- ewentualne wstawki asmowe (*.asm)

Teraz możesz przejść do analizy.

c:\Hello\HelloProject.txt:

```
#----------------------------
# (c) 2004 by Karol Ossowski
#----------------------------
# >>   Hello Project     << #
#----------------------------

#Tryb budowy projektu:
# 1 - Release
# 2 - Debug
# 3 - Rebuilt
TRYB = 1

SRC = Hello.cpp klasa.cpp      #nie wstawiaj przecinków
HDR = klasa.h
ASM =
RES = zasoby.res
LIB =
BIN = Hello.exe
DEF = DEF.def

C_FLAGS = -tW -c -q -w-par -w-rvl
L_FLAGS = /aa /Tpe /Gk /t /q -w-dup
```

c:\Hello\obj\makefile.mak:

```
DIR = c:\Hello                  #nie wstawiaj na końcu '\'

!include $(DIR)\HelloProject.txt

.silent
.autodepend
.cacheautodepend
.suffixes: .cpp .asm .rc .res .obj .exe

.path.obj = $(DIR)\obj
.path.res = $(DIR)\obj
.path.def = $(DIR)\obj

!if $(TRYB)== 2
 DEBUG=-v
!message_________.:Tryb DEBUG:._________
!elif $(TRYB)== 1
 !message_________.:Tryb RELEASE:._________
!elif $(TRYB)== 3
 !message odbudowywanie....
```

```
REBUILT:
 TOUCH $(DIR)\src\*.cpp
 TOUCH $(DIR)\src\res\*.rc
 TOUCH $(DIR)\src\asm\*.asm
 DEL $(DIR)\obj\*.il*
!else
!error NIE wybrano trybu budowy projektu
!endif

#Zasoby#
{$(DIR)\src\res}.rc{$(DIR)\obj}.RES:
  BRCC32 $<
  COPY $:$(@F) $(DIR)\obj#kopiowanie plików wynikowych do \obj
  DEL  $:$(@F)           #likwidowanie powyższych plików w \src

#Asembler#
{$(DIR)\src\asm}.asm{$(DIR)\obj}.obj:
  TASM32 $<

#Kompilacja#
{$(DIR)\src}.cpp{$(DIR)\obj}.obj:
  BCC32 $(C_FLAGS) $(DEBUG) -I$(DIR)\src\hdr $<

#Konsolidacja#
$(BIN): $(SRC:.cpp=.obj) $(ASM:.asm=.obj) $(RES)
  ILINK32 $(L_FLAGS) $(DEBUG) c0w32.obj $(SRC:.cpp=.obj)\
$(ASM:.asm=.obj),$.,,cw32.lib import32.lib $(LIB),$(DEF),$(RES)
!if $(TRYB)==2
  TD32 $(BIN)
!else
  COPY $. $(DIR)\bin
  $(DIR)\bin\$(BIN)
!endif
```

Przyznam, że ten makefile jest dość zaawansowany jak na materiał dla świeżo upieczonych użytkowników MAKE'a ale chciałem pokazać jak najwięcej możliwości tego narzędzia. Kod ten składa się z dwóch części:

1. interfejs (`HelloProject.txt`) - miejsce, które będziesz stale używał(a) w czasie pracy nad projektem. Możesz deklarować tam nowe pliki źródłowe, odznaczać opcje linkera i kompilatora, i określać tryb budowy projektu;
2. implementacja (`makefile.mak`) - część, w której zawarte są same komendy budowy projektu, ten segment jest tworzony tylko raz podczas pisania makefile'a i od tego czasu raczej nie powinieneś go ruszać;

Tak naprawdę skrypt ten jest na tyle uniwersalny, że możesz używać go w wielu projektach (byle była zachowana odpowiednia struktura katalogów). Do makefile'a dodałem zasadę kompilującą w wstawki assemblerowe. Program TASM32, który jest kompilatorem assemblera niestety nie należy do FCLT, to wciąż komercyjny produkt Borlanda. Opcję tą dodałem tylko po to, aby makefile był już w 100% kompletny. W kodzie użyłem jeszcze jednego programu spoza FCLT. Jest to Turbo Debugger (TD32).TD32 to świetny darmowy debugger który powinieneś mieć w swojej kolekcji, jest do ściągnięcia z ze internetowej Borlanda (http://www.borland.pl%7Cstrony).

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document

may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous

versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License,

and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of

following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Źródło „http://pl.wikibooks.org/w/index.php?title=Borland_C%2B%2B_Compiler/Wersja_do_druku&oldid=191632”
