Kompilacja i scalanie programów C w linii poleceń (LINUX)

Uwaga!
W poniższym tekście każde wyrażenie typu <x> oznacza miejsce wstawienia odpowiedniej nazwy opisywanej przez x. Symbol [x] oznacza występowanie x zero lub jeden raz.

Kompilowanie plików źródłowych C do postaci plików obiektów wymaga użycia opcji „-c".
gcc -c <nazwa_pliku>.c

W wyniku otrzymuje się plik obiektu
<nazwa_pliku>.o

Jeśli plik nagłówka (.h) nie znajduje się w bieżącym katalogu ani w żadnym z katalogów zawierających nagłówki standardowych bibliotek to przy kompilacji stosowana jest opcja –I <katalog_naglowka> . (duże 'i')
gcc –c –I <katalog_naglowka> <nazwa_pliku>.c

Aby wytworzyć kod produkcyjny korzystamy z ustawienia 3 poziomów optymalizacji poprzez opcję „-Ox" (x=0,1,2.3). Poziom „-O2" jest odpowiedni dla większości programów.

Scalanie plików obiektów w program wykonywalny wymaga użycia opcji „-o"
gcc –o <nazwa_celu> <nazwa_pliku1>.o <nazwa_pliku2>.o …

<nazwa_celu> oznacza nazwę pliku wykowywalnego, która może, ale nie musi mieć żadnego konkretnego rozszerzenia (takiego jak np. „exe" w systemie Windows).
W wielu systemach uniksowych można połączyć kompilację i scalanie plików w jednym poleceniu:
gcc –o <nazwa_celu> <nazwa_pliku1>.c <nazwa_pliku2>.c …

Archiwum (bilbioteka statyczna) to zestaw wielu plików obiektów przechowywanych w pojedynczym pliku o rozszerzeniu „.a". Tworzy się je poleceniem „ar" z opcjami „cr":
ar -crs lib<nazwa_bilbioteki>.a <nazwa_pliku1>.o <nazwa_pliku2>.o …

Archiwum dołącza się na etapie scalania plików obiektów w program wykonywalny z wykorzystaniem opcji „-L" i „-l". (małe 'el')
„-L<nazwa_katalogu>": wskazuje położenie niestandardowe pliku biblioteki – poza typowo przeszukiwanymi katalogami „/lib", „/usr/lib" i katalogami opisanymi w ścieżkach zmiennej środowiskowej „LD_LIBRARY_PATH".
„-l<nazwa_biblioteki>": opcja scalania wybranej biblioteki z naszym programem wykonywalnym (celem).
gcc –o <nazwa_celu> <nazwa_pliku1>.c <nazwa_pliku2>.c … -L<nazwa_katalogu_biblioteki> -l<nazwa_biblioteki>

Kompilacja i scalanie programów C poprzez polecenie „make” (LINUX)

Polecenie „make” szuka w bieżącym katalogu pliku tekstowego o nazwie *„makefile”* lub *„Makefile”* w podanej kolejności. Plik ten zawiera reguły opisujące dla „make” co budować (wykonywać) i w jaki sposób. Reguły pliku make mają następującą formę ogólną:

<nazwa_celu>: <zależność> [<zależność>] […]
<tabulator> <polecenie>
[<tabulator> <polecenie>]
[…]

Cel jest plikiem binarnym(wykonywalnym) lub obiektowym (.o), który chcemy utworzyć.
Polecenia są krokami takimi jak wywołania kompilatora lub poleceń powłoki koniecznymi dla utworzenia celu.
Jeśli cel nie istnieje „make” go buduje zgodnie z poleceniem. W przeciwnym razie porównuje daty tworzenia celu z datami jego zależności. Jeśli są one późniejsze niż cel (przynajmniej jedna) cel podlega przebudowaniu, bo zmienił się jakiś kod.
Przykład:

```
prog.e: m1.o m2.o
        gcc –o prog.e m1.o m2.o

m1.o:  m1.c m1.h
        gcc –c m1.c

m2.o:  m2.c m1.h
        gcc –c m1.c

clean:
        rm *.o *.e
```

Wybrane zmiene automatyczne stosowane w plikach “make”:
$@ -symboliczna nazwa pliku celu w regule
$* -rdzeń nazwy pliku (bez rozszerzenia po kropce)
$< -nazwa pliku pierwszej zależności od reguły
$^ -lista wszystkich zależności w regule
$? - lista zależności nowszych niż cel

Zmienne lokalne w plikach „make” sa przypisywane różnym poleceniom na początku skryptu.
Przypisanie takie (nierekursywne) ma postać:
<zmienna>:=<polecenie lub wartość>
np. :
CC := gcc
CFLAGS := -g

Odwołanie do takich zmiennych występuje w formie $(<zmienna>)
np. :
$(CC)
$(CFLAGS)

Ogólna reguła kompilacji plików źródłowych C w poleceniu „make” z użyciem zmiennych automatycznych może mieć postać:

```
.c.o:
        $(CC) $(CFLAGS) –c $*.c
```

Będzie ona niejawnie cytowana przy tworzeniu celu:

<plik_wykonywalny>: <plik>.o <plik>.o …
$(CC) $(CFLAGS) $^ -o $@

<plik>.o: <plik>.c <plik>.c …
<plik>.o: <plik>.c <plik>.h …

Przykład prostego pliku ”Makefile” z użyciem reguł niejawnych, zmiennych lokalnych i zmiennych automatycznych:

```
CC := gcc
CFLAGS := -g

.c.o:
        $(CC) $(CFLAGS) –c $*.c

prog.e: m1.o m2.o
        $(CC) $(CFLAGS) $^ -o $@

m1.o: m1.c m1.h
m2.o: m2.c m2.h

clean:
        rm –f *.o *.e
```

W jednym pliku „Makefile” można definiować wiele różnych celów – kompilować i scalać wiele programów. Wywołanie „make” odbywa się z odpowiednim parametrem stanowiącym nazwę celu i może zawierać opcje:

make *[-<opcje>] <nazwa_celu lub plik_wykonywalny>*
np.:
make prog.e
make clean.

Wybrane opcje polecenia “make”(nie muszą występować):
-n : polecenia są składane i wyświetlane, ale nie wykonywane (dobre do testów)
-I <katalog> : katalog do poszukiwania plików make poza katalogiem bieżącym (duże 'i')
-s : (silent) make nie wypisuje poleceń na ekranie
-f <plik> : nazwa pliku make inna niż „makefile” lub „Makefile”
-k : nie przerywa działania jeśli nie uda się zbudować jednego z celów
-d : (debug) wyświetlane są informacje debugowania
-W<plik> : wykonuje się tak jakby wymieniony plik był zmodyfikowany (do testowania)

Uwaga!
W systemie UNIX BSD należy zastosować polecenie gmake (GNU make) aby móc budować pliki „make” w formie charakterystycznej dla Linuxa.