Fragment wykładu z języka C (2002-2009)

Piotr Szwed
pszwed@agh.edu.pl

Program make

- Typowy program w języku C/C++ składa się z wielu odrębnych modułów (jednostek translacji). Ich liczba może dochodzić do kilkuset. Każdy z modułów jest kompilowany do postaci wynikowej OBJ; następnie wszystkie moduły są konsolidowane w jeden program wykonywalny (EXE) lub bibliotekę.
- Optymalizacja procesu kompilacji polega na tym, aby kompilować wyłącznie zmodyfikowane moduły. Podstawowym kryterium jest atrybut określający datę i czas ostatniej modyfikacji. Jeżeli plik OBJ nie istnieje lub jest on starszy niż plik źródłowy C/C++, wówczas powinien zostać on ponownie skompilowany.
- Następnie uaktualnione pliki OBJ są powtórnie konsolidowane.
- Nowoczesne kompilatory oferują dodatkową opcję: *inkremetalnego linkowania*. Proces konsolidacji polega na wymianie w programie wykonywalnym wyłącznie tych fragmentów kodu, który został zmodyfikowany. Nie jest przeprowadzana pełna konsolidacja.
- Zazwyczaj IDE (*integrated development environment*) ma wbudowane funkcje, które określają, które moduły należy skompilować. Przed powstaniem tego typu środowisk rolę tę pełnił program *make*. Bardzo często programy IDE generują plik konfiguracyjny (*makefile*) i następnie wywołują program *make* podczas kompilacji.

Działanie programu make

Program *make* uruchamia się podając nazwę pliku konfiguracyjnego. (Bardzo często pliki konfiguracyjne mają rozszerzenie *mak*). Jeżeli nazwa nie podana, standardowo czytany jest plik o nazwie *makefile* z bieżącego katalogu.

Przykład makefile

```
# komentarz
hello.exe : hello.c
    ccompiler.exe hello.c
```

Makefile specyfikuje *zależności* pomiędzy plikami. Tutaj istnieje zależność pomiędzy plikiem wykonywalnym `hello.exe` oraz plikiem źródłowym `hello.c`. Plik `hello.exe` jest nazywany obiektem docelowym (ang. *target*). Po dwukropku umieszcza się jeden lub więcej obiektów, od których obiekt docelowy jest zależny (ang. *dependency*). Jeżeli obiekt docelowy (`hello.exe`) nie istnieje lub jest starszy obiekty, od których jest uzależniony (plik `hello.c`), wykonywana jest *reguła* podana bezpośrednio po zależności. W podanym przykładzie regułą jest uruchomienie programu `ccompiler.exe` z argumentem `hello.c`.
Większość programów *make* wymaga, aby linia, w której opisane są reguły rozpoczynała się pojedynczym znakiem tabulacji.

Makra w programie make

W makefile można posługiwać się makrami. Makra nadają nazwy łańcuchom tekstowym. Aby rozwinąć makro, należy umieścić je wewnątrz nawiasów i poprzedzić znakiem `$`.
Przykład

```
# komentarz
CC = ccompiler.exe
hello.exe : hello.c
```

```
    $(CC) hello.c
```

Makra pozwalają na łatwiejszą modyfikację makefile. Na przykład nazwa i położenie kompilatora oraz jego opcje mogą być wprowadzone tylko raz.
W programie make istnieją także predefiniowane makra związane ze specyfikacjami plików występujących wyłącznie zależnościach.

`$@` Pełna ścieżka dla obiektu (pliku) docelowego.

`$*` Ścieżka i nazwa dla obiektu (pliku) docelowego bez rozszerzenia.

`$**` Wszystkie pliki, od których bieżący obiekt docelowy jest zależny.

`$?` Wszystkie pliki, od których bieżący obiekt docelowy jest zależny z późniejszym czasem modyfikacji (późniejszą pieczątką czasową, ang. *timestamp*).

`$<` Plik, od którego bieżący obiekt docelowy jest zależny z późniejszą pieczątką czasową.

Reguły dla rozszerzeń

Bardzo często wykonanie reguły polega na stworzeniu pliku o tej samej nazwie, ale zmienionym rozszerzeniu. Możemy skonstruować zależności, które będą działały dla plików o różnych nazwach, ale o znanych rozszerzeniach.
Przykład

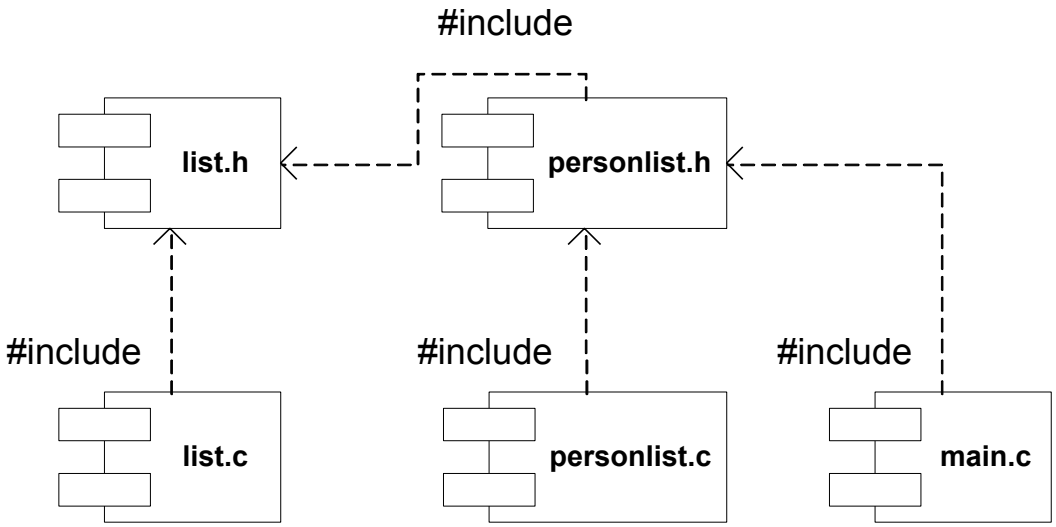
```
CC = ccompiler.exe
OBJFLAG = -o # generate obj flag
INCLUDE = c:\ccompiler\include
LINK = linker.exe
.SUFFIXES: obj. c.
.c.obj:
    $(CC) -I$(INCLUDE) $(OBJFLAG) $*.c
test.exe : main.obj list.obj personlist.obj
    $(LINK) -o test.exe main.obj list.obj
personlist.obj
```

Powyższy przykład:

- Ustala zależność pomiędzy `test.exe` i modułami `main.obj list.obj personlist.obj`. Aby stworzyć plik wykonywalny należy wywołać program określony przez makro `LINK`.
- Aby stworzyć plik o rozszerzeniu *obj* z pliku p rozszerzeniu *c*, należy wywołać kompilator `$(CC)` , podając standardową kartotekę skąd należy czytać włączane pliki `$(INCLUDE)` z flagą `$(OBJFLAG)` generacji plików *obj*.
- Jeżeli którykolwiek z plików `main.c`, `list.c`, lub `personlist.c` będzie starszy niż odpowiadający im plik *obj*, wówczas zostanie on skompilowany. Plik *obj* będzie wówczas młodszy niż plik `test.exe`; wykonana zostanie reguła `$(LINK) -o test.exe main.obj list.obj personlist.obj`

Makefile i włączane pliki

Rozważmy przykład, w którym występują pliki nagłówkowe i pliki źródłowe.



- Plik `list.c` włącza za pomocą dyrektywy `#include` plik `list.h`.

- Plik `personlist.h` włącza `list.h`
- Plik `personlist.c` włącza `personlist.h`, a więc pośrednio także `list.h`.
- Plik `main.c` włącza `personlist.h`, a więc pośrednio także `list.h`.

Zastosowanie reguł opartych na rozszerzeniach plików nie zapewnia rekompilacji modułów w przypadku modyfikacji plików nagłówkowych. Na przykład po zmodyfikowaniu `personlist.h` należy powtórnie skompilować `personlist.c` oraz `main.c`, natomiast po modyfikacji `list.h` wszystkie moduły.

Poprawnie skonstruowany plik konfiguracyjny *makefile* powinien uwzględniać także zależności pomiędzy plikami zawierającymi kod źródłowy i plikami nagłówkowymi.

Przykład

```
CC = ccompiler.exe
OBJFLAG = -o # generate obj flag
INCLUDE = c:\ccompiler\include
LINK = linker.exe
list.obj : list.c list.h
    $(CC) -I$(INCLUDE) $(OBJFLAG) list.c
personlist.obj : personlist.c personlist.h
list.h
    $(CC) -I$(INCLUDE) $(OBJFLAG) personlist.c
main.obj : main.c personlist.h list.h
    $(CC) -I$(INCLUDE) $(OBJFLAG) main.c
test.exe : main.obj list.obj personlist.obj
    $(LINK) -o test.exe main.obj list.obj
personlist.obj
```

Tego typu programy *makefile* są zazwyczaj generowane automatycznie przez IDE w procesie przeszukiwania drzewa zależności (ang. *scanning dependencies*). Podczas przeszukiwania włączane są kolejne pliki i analizowane zawarte w nich dyrektywy `#include`.

Sztuczny obiekt docelowy

Bardzo często dostarczana aplikacja składa się z pewnej liczby plików wykonywalnych lub bibliotek DLL. W jednym pliku konfiguracyjnym *makefile* możemy określić, jak zbudować je wszystkie. W tym celu specyfikujemy sztuczny obiekt docelowy (ang. *pseudotarget*) uzależniony od nich wszystkich. Odpowiadająca mu reguła jest pusta.

Przykład:

```
example1.exe : lista modułów
    reguła tworząca example1.exe
example2.exe : lista modułów
    reguła tworząca example2.exe
example3.exe : lista modułów
    reguła tworząca example3.exe
all: example1.exe example2.exe example3.exe
```

Aby stworzyć sztuczny obiekt docelowy `all`, wykonywane będą reguły tworzące `example1.exe`, `example2.exe` oraz `example3.exe`. Plik `all` nigdy nie powstanie, ponieważ reguła jest pusta.

Przy tak skonstruowanym pliku *makefile* możliwe jest dalej generowanie indywidualnych plików wchodzących w skład aplikacji (lub plików wynikowych *obj*). Jednym z parametrów komendy *make* jest nazwa obiektu docelowego.

- w wyniku wywołania `make example2.exe` po przeczytaniu pliku *makefile* wykonane zostaną reguły budujące wyłącznie program wykonywalny `example2.exe;`
- w wyniku wywołania `make` lub `make all` wykonane zostaną reguły budujące wszystkie pliki określone przez

zależność:

```
all: example1.exe example2.exe example3.exe
```

Podsumowanie

Program *make* lub wzorowane na nim rozwiązanie stsowane w środowiskach IDE, jest narzędziem umożliwiającym optymalizację procesu kompilacji dużych programów, składających się z większej liczby modułów.

Kryterium decydującym o tym, czy dany moduł powinien zostać skompilowany jest czas modyfikacji (pieczątka czasowa) pliku źródłowego lub włączanych do niego plików nagłówkowych.

Poprawne działanie programu *make* (a także IDE) jest uzależnione od poprawnych informacji o aktualnym czasie stacji roboczej i czasie modyfikacji plików źródłowych.

Program *make* może mieć bardziej ogólne zastosowanie wykraczające poza proces kompilacji programów w języku C/C++. Na przykład sporządzanie kopii zapasowych, archiwizacja plików, zmiana formatu plików, itd.