Make - Tworzenie Makefile

Autor: ***mgr inż. Bartosz Baliś***

Wstęp

Narzędzie **make** służy do zarządzania kompilacją projektów składających się z wielu plików źródłowych.

Aby używać make należy napisać skrypt o nazwie **Makefile** lub **makefile**, w którym opisane są:

- zależności pomiędzy plikami źródłowymi i plikami wynikowymi,
- sposób tworzenia plików wynikowych z plików źródłowych.

Następnie przy pomocy polecenia **make** kompilujemy projekt.

Make usprawnia kompilację, gdyż samodzielnie decyduje, które z plików źródłowych mają być przekompilowane (sprawdzając daty ostatniej modyfikacji).

Budowa pliku Makefile

Plik Makefile składa się głównie z reguł. Reguła ma następującą budowę:

```
CEL: SKŁADNIKI
        KOMENDA
```

Gdzie CEL to nazwa pliku docelowego, który jest tworzony z plików wymienionych jako SKŁADNIKI, zaś KOMENDA podaje komendę, która tworzy plik docelowy CEL z plików składowych SKŁADNIKI

Przykład reguły:

```
hello: hello.c aux.c
        gcc hello.c aux.c -o hello
```

Reguła ta określa sposób tworzenia pliku wykonywalnego hello z plików hello.c i aux.c

UWAGA *Przed komendą musi obowiązkowo wystąpić znak tabulacji.*

Przetwarzanie pliku składającego się z wielu reguł.

Make dąży do tego, żeby utworzyć plik docelowy znajdujący się w pierwszej regule w pliku Makefile. szystkie pozostałe reguły w pliku Makefile są pomocnicze i zwykle podają sposób utworzenia składników z reguł poprzednich.

Przykład pliku Makefile składającego się z kilku reguł:

```
hello: hello.o aux.o
	gcc hello.o aux.o -o hello

hello.o: hello.c
	gcc -c hello.c -o hello.o

aux.o: aux.c
	gcc -c aux.c -o aux.o
```

W tym wypadku głównym plikiem docelowym jest plik hello. Jego składniki to pliki hello.o i aux.o. Make szuka tych plików w katalogu bieżącym, jeśli jednak ich nie znajdzie to szuka reguł, które podają sposób w jaki te pliki utworzyć. W powyższym przykładzie są to dwie kolejne reguły. Make korzysta z nich, żeby utworzyć składniki reguły głównej i dopiero wtedy może utworzyć główny plik docelowy hello

Zmienne w pliku Makefile

W pliku Makefile można definiować zmienne, tak jak w poniższym przykładzie:

```
OBJS=hello.o aux.o

hello: $(OBJS)
	gcc $(OBJS) -o hello

hello.o: hello.c
	gcc -c hello.c -o hello.o

aux.o: aux.c
	gcc -c aux.c -o aux.o
```

W tym przykładzie zdefiniowana została zmienna o nazwie OBJS i nadana jej została wartość 'hello.o aux.o'. Odwołanie do zdefiniowanej zmiennej odbywa się przy pomocy znaku dolara i nawiasów okrągłych. W tym przypadku raz zdefiniowana zmienna została użyta dwukrotnie. Jeśli będziemy chcieli dodać coś do listy składników pliku hello to dzięki użyciu zmiennej wystarczy to zrobić raz - w definicji zmiennej OBJS

Zmienne standardowe

W Makefile można używać wielu zmiennych zdefiniowanych standardowo. Najczęściej używane zmienne standardowe:

- `CC` - nazwa kompilatora języka C
- `CXX` - nazwa kompilatora języka C++
- `CFLAGS` - opcje kompilatora języka C
- `CXXLAGS` - opcje kompilatora języka C
- `LFLAGS` - opcje dla linkera

Zmienne standardowe mają pewną predefiniowaną wartość (np. zmienna CC ma predefiniowaną wartość 'cc'), którą jednak można zmieniać. Oto przykład:

```
CC=gcc
CFLAGS=-g
LFLAGS=
```

```
OBJS=hello.o aux.o

hello: $(OBJS)
	$(CC) $(LFLAGS) $(OBJS) -o hello

hello.o: hello.c
	$(CC) $(CFLAGS) -c hello.c -o hello.o

aux.o: aux.c
	$(CC) $(CFLAGS) -c aux.c -o aux.o
```

Zmienne automatyczne

Zmienne automatyczne są to specjalne zmienne, które przechowują wartości zmieniające się dynamicznie w trakcie wykonywania Makefile'a, np. nazwa pliku docelowego aktualnie przetwarzanej reguły. Najczęściej używane zmienne automatyczne:

- `<` - aktualnie przetwarzany plik z listy składników (patrz przykład)
- `@` - nazwa pliku docelowego
- `^` - składniki

```
CC=gcc
CFLAGS=-g
OBJS=hello.o aux.o

hello: $(OBJS)
	$(CC) $(LFLAGS) $^ -o $@

hello.o: hello.c
	$(CC) $(CFLAGS) -c $< -o $@

aux.o: aux.c
	$(CC) $(CFLAGS) -c $< -o $@
```

Więcej o regułach

Reguły z wzorcem.

```
CC=gcc
CFLAGS=-g
OBJS=hello.o aux.o

hello: $(OBJS)
	$(CC) $(LFLAGS) $^ -o $@

$(OBJS): %.o: %.c
	$(CC) -c $(CFLAGS) $< -o $@
```

Reguły domyślne.

Make posiada pewne standardowe reguły do tworzenia pewnych typów plików. W poniższym przykładzie pominięto regułę tworzenia plików '.o' z plików '.c'. Make posiada jednak standardowe, domyślne reguły tworzenia plików .o. W tym przypadku make odnajdzie w katalogu bieżącym pliki '.c', z których utworzy pliki '.o' przy pomocy polecenia '$(CC) -c'.]

```
CC=gcc
CFLAGS=-g
OBJS=hello.o aux.o

hello: $(OBJS)
	$(CC) $(LFLAGS) $^ -o $@
```

Reguła clean

Każdy dobry plik makefile powinien posiadać regułę, która usuwa pliki pośrednie, tymczasowe, etc. powstałe podczas kompilacji. W powyższych przykładach plikami pośrednimi są pliki '.o'. Powszechnie przyjęło się, że taka reguła ma nazwę 'clean'. Oto przykład:

```
CC=gcc
CFLAGS=-g
OBJS=hello.o aux.o

hello: $(OBJS)
	$(CC) $(LFLAGS) $^ -o $@

clean:
	rm -f *.o
```

Zauważmy, że reguła clean ma pustą listę składników. Aby wywołać regułę clean używamy polecenia make clean

Więcej plików wynikowych

```
CC=gcc

all: prog1 prog2

prog1: prog1.c
	$(CC) $(CFLAGS) $(LFLAGS) prog1.c -o prog1

prog2: prog2.c
	$(CC) $(CFLAGS) $(LFLAGS) prog2.c -o prog2

clean:
	rm -f prog1.o prog2.o
```

Reguła .PHONY

Reguły 'clean' i 'all' są regułami specjalnymi w tym sensie, że 'clean' i 'all' nie są nazwami plików. Gdyby jednak zdarzyło się, że w katalogu bieżącym istnieje plik o nazwie 'all' lub 'clean' to reguły te mogłyby nie działać! Aby temu zapobiec trzeba by poinstruować make'a, że 'all' i 'clean' nie są nazwami plików. Do tego celu służy specjalna reguła o nazwie '.PHONY'.

```
CC=gcc

all: prog1 prog2
```

```
prog1: prog1.c
	$(CC) $(CFLAGS) $(LFLAGS) prog1.c -o prog1

prog2: prog2.c
	$(CC) $(CFLAGS) $(LFLAGS) prog2.c -o prog2

clean:
	rm -f prog1.o prog2.o

.PHONY: all clean
```