Pisanie programów wielomodułowych (w wielu plikach)

Wyobraźmy sobie, że piszemy duży projekt. W takim momencie sensowne jest dzielenie programu na funkcjonalne części, z których każda zajmuje się osobnym podproblemem; każda taka część znajduje się w osobnym pliku. Jeśli program jest bardzo duży, to chcemy, żeby za każdym razem kompilacja wszystkich plików nie była konieczna (względy czasowe); chcemy kompilować tylko te pliki, które się zmieniły.

Weźmy modelowy przykład, na początku mamy jednomodułowy program zapisany w pliku `prog.c`:

```
#include <stdio.h>

double wygeneruj_liczbe()
{
    return 123.123;
}

void wypisz_liczbe(double x)
{
    printf ("%lf\n",x);
}

int main()
{
    wypisz_liczbe(wygeneruj_liczbe());
    return 0;
}
```

Program kompilujemy poleceniem `gcc prog.c -o prog.exe`. Jak dotąd wszystko działa. Chcemy jednak podzielić program na moduły.

1 Jak nie należy tego robić

Utwórzmy trzy pliki: `main.c`, `f1.c` i `f2.c` o następujących zawartościach.

Plik `f1.c`:

```
double wygeneruj_liczbe()
{
    return 123.123;
}
```

Plik `f2.c`:

```
#include <stdio.h>

void wypisz_liczbe(double x)
{
    printf ("%lf\n",x);
}
```

Plik `main.c`:

```
#include "f1.c"
#include "f2.c"

int main()
{
    wypisz_liczbe(wygeneruj_liczbe());
    return 0;
}
```

Zauważmy, że polecenie `#include <stdio.h>` zostało przeniesione do pliku `f2.c`. Ma to akurat sens, bo ten plik nagłówkowy jest potrzebny tylko do kompilacji funkcji `printf`.

Program kompilujemy ponownie poleceniem `gcc main.c -o prog.exe`. Co tu jest bez sensu? Po pierwsze dyrektywą `#include` włączamy do programu plik `*.c`. Choć jest to technicznie możliwe, może być bardzo mylące dla czytającego kod; ta technika jest zarezerwowana dla plików nagłówkowych `*.h`. Po drugie i znacznie ważniejsze: **nic** (poza uzyskaniem paru plików więcej) **nie osiągneliśmy**. W szczególności jakakolwiek zmiana w jednym z pliku wymusza konieczność kompilacji całości.

2 Jak to zrobić z sensem

Pierwsza przymiarka do rozwiązania jest następująca. Zmodyfikujmy plik `main.c`:

```
int main()
{
    wypisz_liczbe(wygeneruj_liczbe());
    return 0;
}
```

a następnie skompilujmy poszczególne pliki wydając polecenia:

```
gcc -c -W -Wall f1.c
gcc -c -W -Wall f2.c
gcc -c -W -Wall main.c
```

Zwróćmy uwagę na przełącznik `-c`. Pliki zostaną poddane tylko kompilacji do obiektów wynikowych (plików `*.o`), a nie zostaną jeszcze skonsolidowane (zlinkowane) razem. Opcje `-W -Wall` wyświetlają (bardzo przydatne) ostrzeżenia.

Próba wykonania powyższych poleceń powiedzie się w przypadku dwóch pierwszych, natomiast w przypadku trzeciego otrzymamy komunikat:

```
main.c: In function 'main':
main.c:3: warning: implicit declaration of function 'wypisz_liczbe'
main.c:3: warning: implicit declaration of function 'wygeneruj_liczbe'
```

Komunikat ten wynika z tego, że podczas kompilacji pliku `main.c` kompilator nie wie jak wyglądają funkcje `wypisz_liczbe` i `wygeneruj_liczbe` Co więcej, kompilator „odgadnie", że zwracanym przez nie typem wartości jest `int`!

Obiekty wynikowe łączymy poleceniem `gcc f1.o f2.o main.o -o prog.exe`. Po uruchomieniu dostajemy oczywiście — ze względu na konwersję do typu całkowitego — nieprawdziwy wynik (ja dostałem `-0.268230`).

Musimy zatem poinformować kompilator jak wyglądają *prototypy* tych funkcji, czyli zmienić plik `main.c` w następujący sposób:

```
void wypisz_liczbe(double);
double wygeneruj_liczbe();

int main()
{
    wypisz_liczbe(wygeneruj_liczbe());
    return 0;
}
```

Po kompilacji (wystarczy skompilować tylko plik `main.c`) i konsolidacji otrzymamy poprawnie działający plik `prog.exe`.

Zauważmy, że jeśli zmodyfikujemy np. działanie funkcji `wypisz_liczbe` w pliku `f2.c`, to wystarczy skompilować tylko plik `f2.c` i skonsolidować całość poleceniem `gcc f1.o f2.o main.o -o prog.exe`. Daje to znaczące korzyści, jeśli plików mamy więcej: zawsze wystarczy skompilować tylko te, które uległy zmianie.

3 Jak zrobić to jeszcze lepiej?

Powyższe podejście wystarcza do budowy małych wielomodułowych programów. Wyobraźmy sobie jednak, co dzieje się, jeśli `f1.c` i `f2.c` są dużymi bibliotekami udostępniającymi kilkaset funkcji. Nie chcielibyśmy, żeby programista czyli użytkownik tych bibliotek musiał na początku każdego programu deklarować wszystkich funkcji, których chce w tym programie użyć. Dlatego też można te wszystkie deklaracje (prototypy) funkcji umieścić w pliku nagłówkowym, włączanym przez program główny.

W naszym przykładzie utworzymy dwa plik nagłówkowe `f1.h` i `f2.h` o następującej zawartości.

Plik `f1.h`:

```
double wygeneruj_liczbe();
```

Plik `f2.h`:

```
double wypisz_liczbe(double);
```

Następnie zmodyfikujemy plik `main.c` w następujący sposób:

```
#include "f1.h"
#include "f2.h"

int main()
{
    wypisz_liczbe(wygeneruj_liczbe());
    return 0;
}
```

Dodatkowo należy pamiętać, żeby po każdej modyfikacji plikow `f1.c` i `f2.c`, w trakcie której zmienione zostają nagłówki funkcji, zmodyfikować też odpowiednie pliki nagłówkowe. Żeby o tym nie zapomnieć, warto w plikach `f1.c` i `f2.c` włączać też odpowiednie nagłówki. Pozwoli to wykryć rozbieżności w plikach `*.c` i `*.h` już na etapie ich kompilacji. Ostatecznie więc pliki z funkcjami pomocniczymi powinny wyglądać następująco.

Plik `f1.c`:

```
#include "f1.h"

double wygeneruj_liczbe()
{
    return 123.123;
}
```

Plik `f2.c`:

```
#include <stdio.h>
#include "f2.h"

void wypisz_liczbe(double x)
{
    printf ("%lf\n",x);
}
```

Marcin Bieńkowski