Podział programu na moduły

Informatyka

Materiały

- Szczegółowe informacje dotyczące wymagań odnośnie podziału na moduły: **http://www.cs.put.poznan.pl/wcomplak/BFILES/C_W_5.PDF**

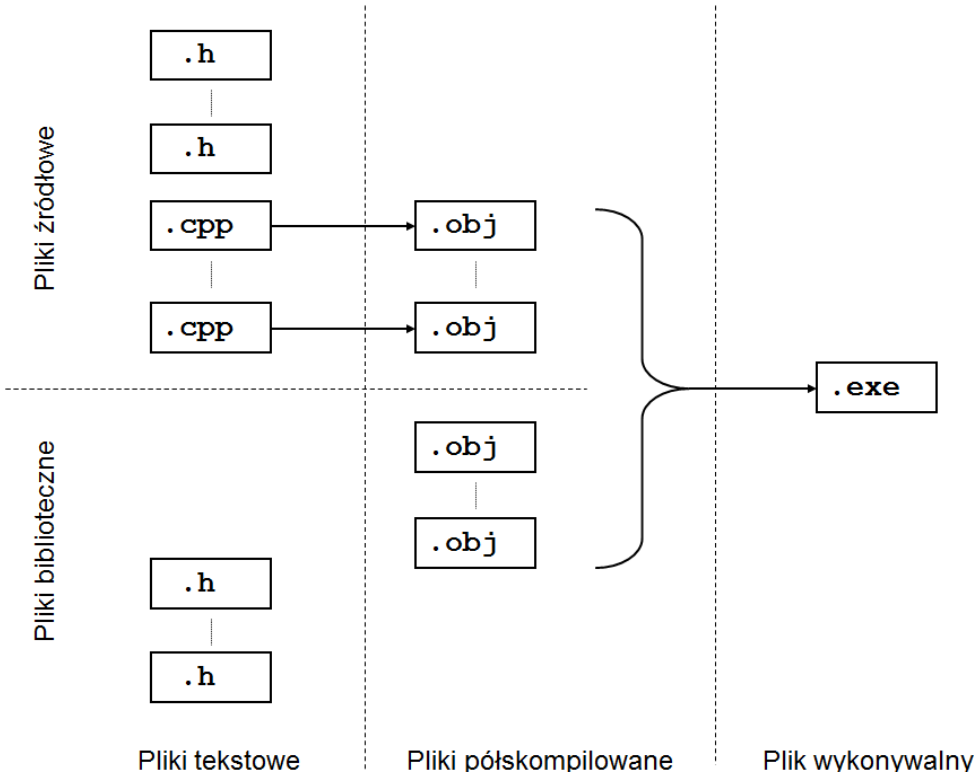
Podział programu na moduły pozwala na:

- Stworzenie programu, który jest bardziej czytelny. Duży program pisany jako jedna całość jest nieczytelny (brak pośrednich form organizacji poza podziałem na funkcje), łatwiej w nim także popełnić błędy (zmodyfikowanie zmiennej globalnej, zależność działania jednej funkcji od ubocznych efektów działania drugiej).
- Łatwiejsze wykorzystanie poszczególnych modułów w innych programach**.**
- Kompilowanie poszczególnych modułów niezależnie. Jeśli dokonamy zmian w jednym z modułów to nie będziemy musieli kompilować ponownie całości, tylko ten jeden plik.

Modułem będziemy nazywać fragment programu kompilowany jako jeden plik.

Każdy moduł składa się z:

- Interfejsu (tego co dany plik udostępnia) czyli pliku nagłówkowego (*.h lub .hpp)
- Implementacji (realizacji) czyli pliku implementacyjnego (*c. lub *.cpp). Na początku takiego pliku należy włączyć odpowiadający implementacji plik nagłówkowy (Np. dla pliku *module.c* będzie to plik *module.h*).



Każdy plik nagłówkowy powinien być napisany tak, aby można go było kilkakrotnie włączać do dowolnego modułu i w dowolnym miejscu.

Pliki nagłówkowe mogą zawierać:

- Stałe jawne (np. EOF w pliku *stdio.h*)
- Funkcje-makra np. *getchar()* jest z reguły definiowana jako zamiennik *getc(stdin)* czy funkcje z rodziny *ctype.h*
- Deklaracje funkcji, np. plik *string.h* zawiera deklaracje funkcji operujących na łańcuchach.
- Definicje struktur, np. standardowe funkcje we/wy korzystają ze struktury zawierającej informacje o pliku i związanym z nim buforze (plik *stdio.h*).
- Definicje typów, np. typ FILE jest wskaźnikiem do struktury zdefiniowanym w pliku *stdio.h* (za pomocą *typedef* lub dyrektywy *#define*). W plikach nagłówkowych zdefiniowane są również typy *size_t* oraz *time_t*.

Wielu programistów opracowuje własne pliki nagłówkowe i wykorzystuje je w swoich programach. Niektóre z nich są przeznaczone do specjalnych zastosowań, inne można

dołączać do niemal każdego programu. Dołączane pliki mogą same zawierać dyrektywy *#include*.

Pliki nagłówkowe mogą zawierać deklaracje zmiennych zewnętrznych współużytkowanych przez kilka plików. Uważane jest to jednak za rozwiązanie wadliwe stylistycznie, ponieważ sprawdzenie, czy dana zmienna została zadeklarowana wymaga w tym przypadku otwarcia dodatkowego pliku.
Mimo to pliki nagłówkowe nadają się dobrze do przechowywania danych typu *const static*. Kwalifikator *const* chroni przed przypadkowymi modyfikacjami, a słowo *static*, że każdy z plików dołączających plik nagłówkowy otrzyma własną kopię stałej – nie trzeba więc pamiętać o umieszczeniu w jednym pliku deklaracji definiującej, a we wszystkich pozostałych deklaracji nawiązujących (za pomocą słowa kluczowego *extern*).

Przykład nieprawidłowego podziału na moduły (nie wystąpi błąd kompilacji)

Efekt końcowy: cały kod zostanie wklejony do pliku *main.cpp*, a następnie dopiero skompilowany (modularność w poniższym przykładzie nie występuje).

```
// kod zawarty w pliku Part1.cpp
double Recip(double x)
{
	if(x != 0)
		return 1.0/x;
	else
		return 0;
}

// kod zawarty w pliku Part2.cpp
int Power(int n)
{
	if (n < 0 || n > 12)
		return 0;

	if (n == 0)
		return 1;
	else
		return n * Power(n-1);
}

// kod zawarty w pliku main.cpp

#include <stdio.h>
#include "Part1.cpp"
#include "Part2.cpp"

int main(int argc, char* argv[])
{
```

```
	int k;
	printf("Liczba calkowita : ");
	scanf("%d",&k);
	printf("\n%lf\t%d\n\n", Recip(k), Power(k));

	return 0;
}
```

Przykład prawidłowego podziału na moduły

Efekt końcowy: poszczególne moduły zostaną skompilowane oddzielnie, połączenie nastąpi na etapie linkowania.

```
//kod zawarty w pliku nagłówkowym Part1.h
double Recip(double x);


// kod zawarty w pliku Part1.cpp
#include "Part1.h"

double Recip(double x)
{
	if(x != 0)
		return 1.0/x;
	else
		return 0;
}


//kod zawarty w pliku nagłówkowym Part2.h
int Power(int n);


// kod zawarty w pliku Part2.cpp
#include "Part2.h"

int Power(int n)
{
	if (n < 0 || n > 12)
		return 0;

	if (n == 0)
		return 1;
	else
		return n * Power(n-1);
}


// kod zawarty w pliku main.cpp
#include <stdio.h>
#include "Part1.h"
#include "Part2.h"

int main(int argc, char* argv[])
{
	int k;
```

```
        printf("Liczba calkowita : ");
        scanf("%d", &k);
        printf("\n%lf\t%d\n\n", Recip(k), Power(k));

        return 0;
}
```