

Schemat konstrukcja pliku Makefile

Bogdan Kreczmer

`bogdan.kreczmer@pwr.wroc.pl`

Zakład Podstaw Cybernetyki i Robotyki
Instytut Informatyki, Automatyki i Robotyki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Copyright©2008 Bogdan Kreczmer

Niniejszy dokument zawiera materiały do wykładu dotyczącego programowania obiektowego. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych prywatnych potrzeb i może on być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Niniejsza prezentacja została wykonana przy użyciu systemu składu $\text{\LaTeX}$ oraz stylu beamer, którego autorem jest Till Tantau.

Strona domowa projektu Beamer:

`http://latex-beamer.sourceforge.net`

- 1 Od drzewa podceli do pliku Makefile
 - Drzewo podceli
 - Konstrukcja pliku Makefile

Stan początkowy

glowny.cpp

```
#include "modul.hh"

int main()
{
    Wywiel("Hejka!!!");
}
```

modul.hh

```
#ifndef MODUL_HH
#define MODUL_HH

void Wywiel(const char*);

#endif
```

modul.cpp

```
#include <iostream>
#include "modul.hh"
using namespace std;

void Wywiel(const char* Napik)
{
    cout << Napik << endl;
}
```

Program składa się z modułu głównego i pomocniczego.

Stan początkowy

glowny.cpp

```
#include "modul.hh"

int main()
{
    Wyswetl("Hejka!!!");
}
```

modul.hh

```
#ifndef MODUL_HH
#define MODUL_HH

void Wyswetl(const char*);

#endif
```

modul.cpp

```
#include <iostream>
#include "modul.hh"
using namespace std;

void Wyswetl(const char* Napis)
{
    cout << Napis << endl;
}
```

Moduł główny i pomocniczy korzystają z pliku "modul.hh".

Stan początkowy i cel

a.out

```
01001 1011 0101 01101011 1  
101101101 0101 101011100  
01001 1011 0101 01101011 1  
1 1011 0101 01101011 1001 0  
101101101 0101 101011100  
01001 1011 0101 01101011 1
```

glowny.cpp

```
#include "modul.hh"  
  
int main()  
{  
    Wywiel("Hejka!!!");  
}
```

modul.hh

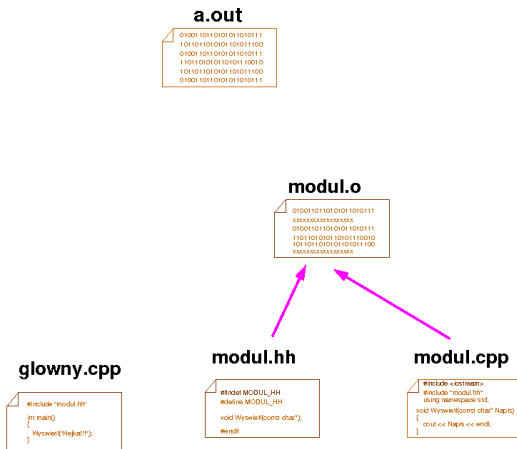
```
#ifndef MODUL_HH  
#define MODUL_HH  
  
void Wywiel(const char*);  
  
#endif
```

modul.cpp

```
#include <iostream>  
#include "modul.hh"  
using namespace std;  
  
void Wywiel(const char* Napic)  
{  
    cout << Napic << endl;  
}
```

Celem jest zbudowanie pliku wykonywalnego "a.out".

Etap pośredni – kompilacja modułu



W tym celu należy utworzyć plik "modul.o"

Etap pośredni – kompilacja modułu

a.out

```
01001 1011 0101 01101011 1
101101101 0101 1010111100
01001 1011 0101 01101011 1
11011 0101 01101011 1001 0
101101101 0101 1010111100
01001 1011 0101 01101011 1
```

modul.o

```
01001 1011 0101 0101 1010111
XXXXXXXXXXXXXXXXXXXX
01001 1011 0101 0101 1010111
11011 0101 101011110010
1011011 0101 011010111100
XXXXXXXXXXXXXXXXXXXX
```

g++ -c -Wall -pedantic modul.cpp

glowny.cpp

```
#include "modul.hh"
int main()
{
    Wyswiel("Hejka!!!");
}
```

modul.hh

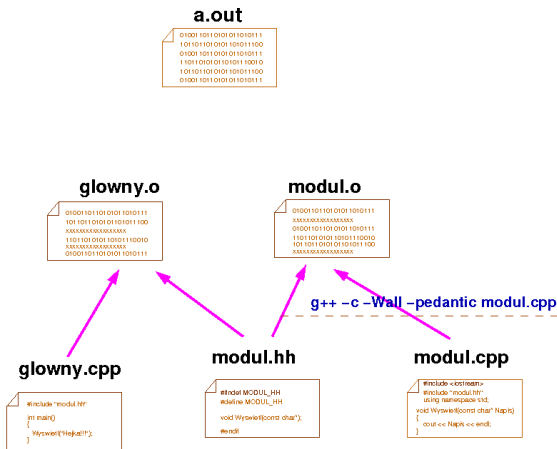
```
#ifndef MODUL_HH
#define MODUL_HH
void Wyswiel(const char*);
#endif
```

modul.cpp

```
#include <iostream>
#include "modul.hh"
using namespace std;
void Wyswiel(const char* Napik)
{
    cout << Napik << endl;
}
```

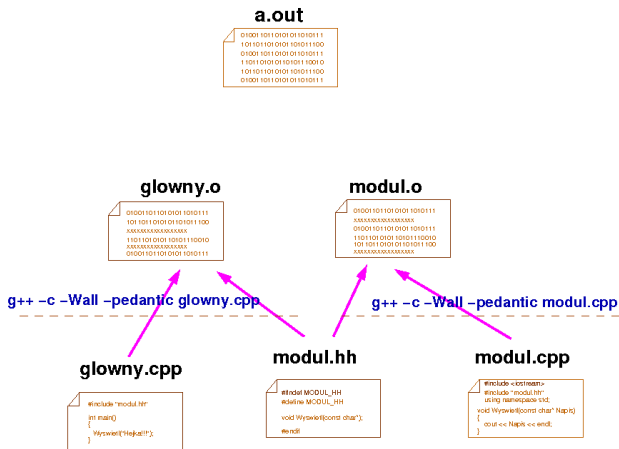
Przepisem na utworzenie tego pliku jest kompilacja.

Etap pośredni – kompilacja modułu głównego



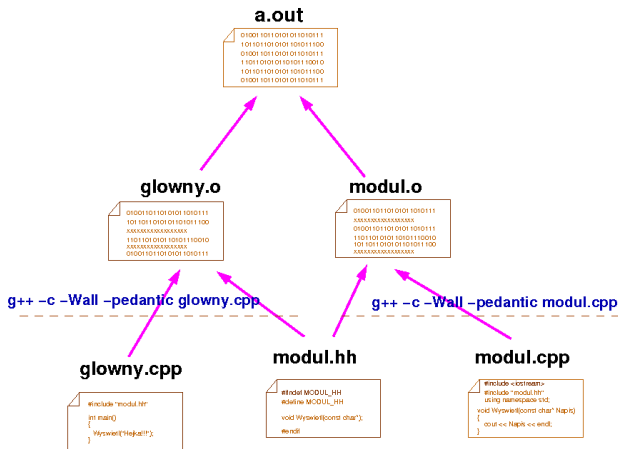
Podobnie należy utworzyć plik "główny.o"

Etap pośredni – kompilacja modułu głównego



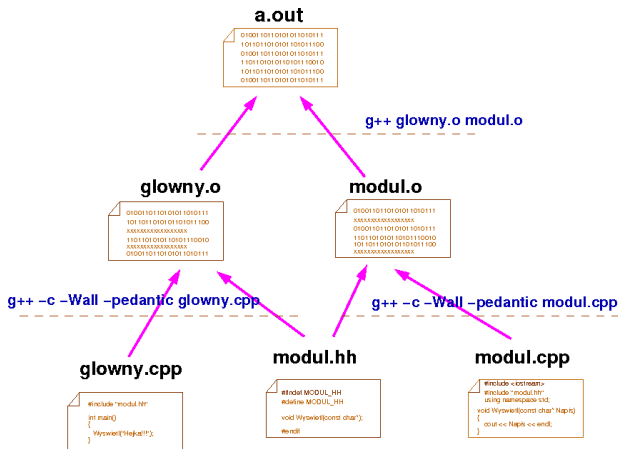
Należy więc skompilować plik "glowny.cpp".

Etap pośredni – konsolidacja programu



Teraz należy zbudować finalny plik.

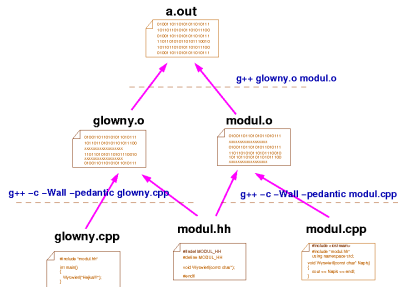
Etap pośredni – konsolidacja programu



Realizujemy to poprzez konsolidację plików typu *object*.

Plik Makefile

Makefile

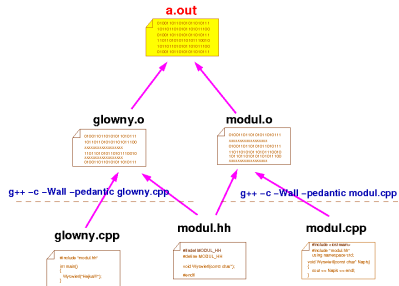


Na podstawie stworzonego schematu kolejnych operacji zbudujemy plik Makefile.

Plik Makefile

Makefile

a.out:

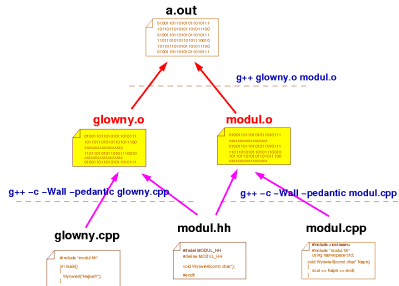


Podajemy zasadniczy cel, który ma być utworzony

Plik Makefile

Makefile

```
a.out: glowny.o modul.o
```



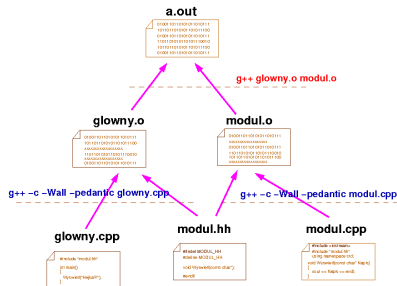
Wypisujemy podcele od których on zależy.

Plik Makefile

Makefile

```
a.out: glowny.o modul.o
```

```
TAB g++ glowny.o modul.o
```



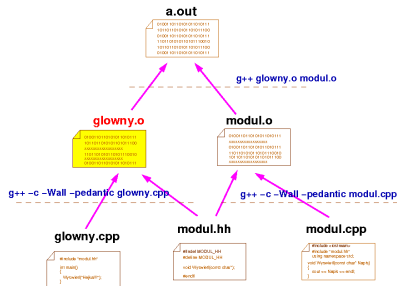
Podajemy przepis utworzenia (lub aktualizacji) celu `"a.out"`. Należy pamiętać, że linię rozpoczynamy od **znaku tabulacji**.

Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o
```

```
glowny.o:
```



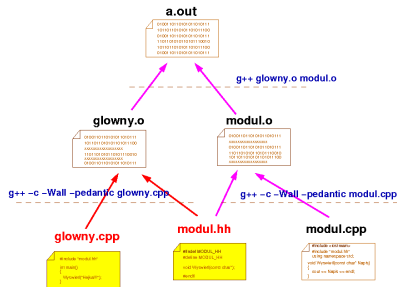
Podobnie robimy dla podcelu `"glowny.o"` i następnych podceli (w tym przypadku `"modul.o"`)

Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o
```

```
glowny.o: glowny.cpp modul.hh
```



Wypisujemy podcele od których on zależy.

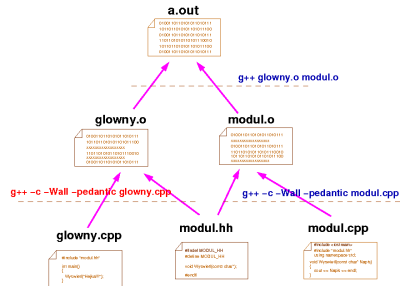
Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o
```

```
glowny.o: glowny.cpp modul.hh
```

```
TAB g++ -c -Wall -pedantic glowny.cpp
```



Podajemy przepis utworzenia (lub aktualizacji) celu `"glowny.o"`.

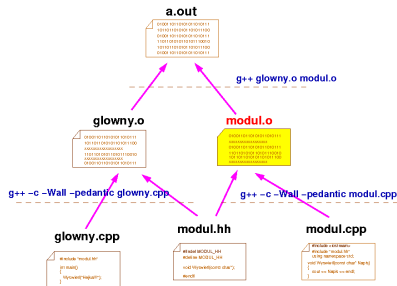
Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o

glowny.o: glowny.cpp modul.hh
    g++ -c -Wall -pedantic glowny.cpp

modul.o:
```



Analogicznie postępujemy dla podcelu "modul.o".

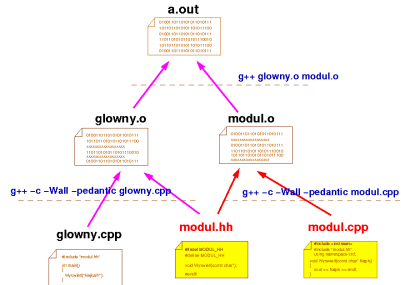
Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o

glowny.o: glowny.cpp modul.hh
    g++ -c -Wall -pedantic glowny.cpp

modul.o: modul.hh modul.cpp
```



Wypisujemy podcele od których on zależy.

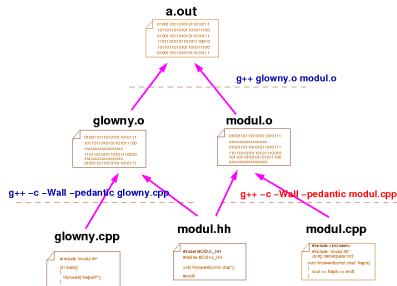
Plik Makefile

Makefile

```
a.out: glowny.o modul.o
    g++ glowny.o modul.o

glowny.o: glowny.cpp modul.hh
    g++ -c -Wall -pedantic glowny.cpp

modul.o: modul.hh modul.cpp
    g++ -c -Wall -pedantic modul.cpp
```



Podajemy przepis utworzenia (lub aktualizacji) celu `"modul.o"`.

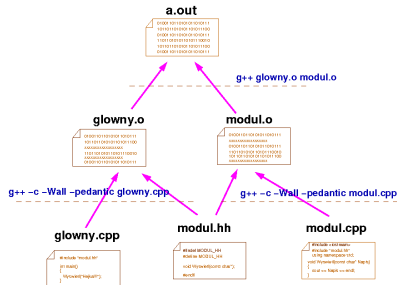
Plik Makefile

Makefile

```
a.out: glowny.o modul.o
      g++ glowny.o modul.o

glowny.o: glowny.cpp modul.hh
      g++ -c -Wall -pedantic glowny.cpp

modul.o: modul.hh modul.cpp
      g++ -c -Wall -pedantic modul.cpp
```



Tak napisany plik Makefile umożliwia automatyczne przeprowadzenie kompilacji i konsolidacji aplikacji. Jednak nie pozwala na jej uruchomienie.

Plik Makefile

Makefile

```
__start__: a.out
```

```
a.out: glowny.o modul.o
```

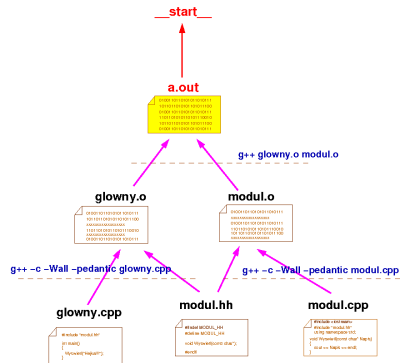
```
g++ glowny.o modul.o
```

```
glowny.o: glowny.cpp modul.hh
```

```
g++ -c -Wall -pedantic glowny.cpp
```

```
modul.o: modul.hh modul.cpp
```

```
g++ -c -Wall -pedantic modul.cpp
```



Dodanie nowego podcelu, który nigdy nie zostanie utworzony, pozwala wymusić uruchomienie aplikacji, gdy zostanie ona poprawnie zbudowana.

Plik Makefile

Makefile

```
__start__: a.out
```

```
TAB ./a.out
```

```
a.out: glowny.o modul.o
```

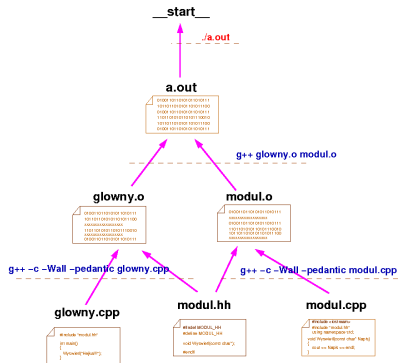
```
    g++ glowny.o modul.o
```

```
glowny.o: glowny.cpp modul.hh
```

```
    g++ -c -Wall -pedantic glowny.cpp
```

```
modul.o: modul.hh modul.cpp
```

```
    g++ -c -Wall -pedantic modul.cpp
```



Jako przepis utworzenia celu `__start__` wpisujemy polecenie uruchomienia aplikacji.

Plik Makefile

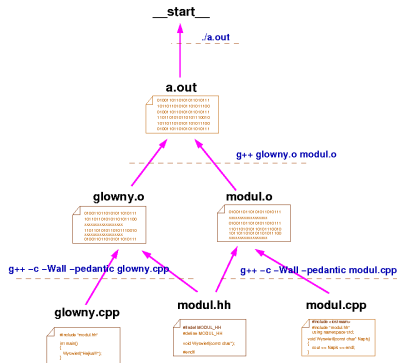
Makefile

```
__start__: a.out
        ./a.out

a.out: glowny.o modul.o
        g++ glowny.o modul.o

glowny.o: glowny.cpp modul.hh
        g++ -c -Wall -pedantic glowny.cpp

modul.o: modul.hh modul.cpp
        g++ -c -Wall -pedantic modul.cpp
```



Należy pamiętać, że ostatnią linię tekstu w pliku należy zakończyć znakiem przejścia do nowej linii (tzn. ostatnią linią w pliku musi być linia pusta).

Plik Makefile – Postać finalna

```
__start__: a.out  
    ./a.out
```

```
a.out: glowny.o modul.o  
    g++ glowny.o modul.o
```

```
glowny.o: glowny.cpp modul.hh  
    g++ -c -Wall -pedantic glowny.cpp
```

```
modul.o: modul.hh modul.cpp  
    g++ -c -Wall -pedantic modul.cpp
```

Koniec prezentacji