Wykład

Materiały bazują częściowo na slajdach Marata Dukhana

Języki programowania

- Kompilowane
  np. C, C++, Pascal

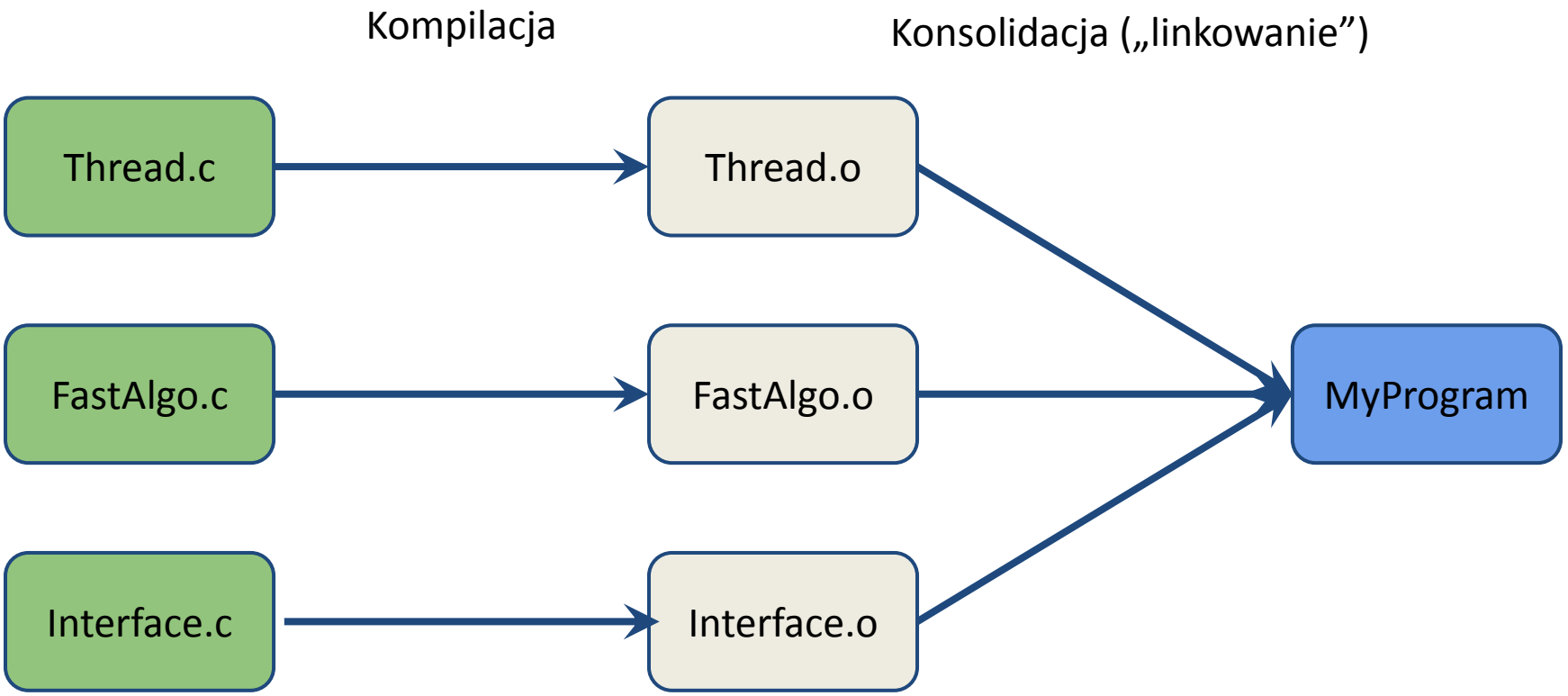
- Interpretowane
  np. JavaScript, PHP, Python, VBA

- Pośrednie
  np. Java, C#

Znane kompilatory C i C++

- GNU C/C++ (gcc/g++)
  - Domyślny pod linuksem
- Clang (clang/clang++)
  - Domyślny dla Maca, czytelne komunikaty błędu
- Intel C/C++ compilers (icc/icpc)
  - Często produkuje wydajniejszy kod dla procesorów Intela
- Microsoft C/C++ compiler (Windows)
  - Domyślne środowisko pod Windows

Kompilacja programu C++



Preprocesor C

Preprocessing: pierwsza faza kompilacji

Instrukcja #define tworzy nowe makro:

```
#define PI 3.14159265358979
```

Makra mogą mieć parametry, np.

```
#define SUM(a,b) ((a)+(b))
```

Pytanie kontrolne: Jakiego typu są a i b?

Preprocessor: Predefiniowane makra

Np. clang zawsze definiuje makro `__clang__`

```
#if defined(__clang__)
    /* Code which uses clang-specific features */
#else
    /* Generic version of the code */
#endif
```

Instrukcje warunkowe preprocesora

```
#if defined(__GNUC__) && (__GNUC__ >= 4)
#elif defined(__GNUC__)
#else
#endif
```

Instrukcje mogą wykorzystywać jedynie liczby całkowite i porównania.

`#ifdef(MACRO)` jest równoważne: `#if defined(MACRO)`

`#ifndef(MACRO)` jest równoważne: `#if !defined(MACRO)`

Dyrektywa	Argumenty	Znaczenie
#define	Łańcuch tekstowy	Definiowanie symbolu
#undef	Łańcuch tekstowy	Anulowanie #define
#include	Nazwa pliku	Włącznie pliku do tekstu źródłowego.
#if	Wyrażenie logiczne	Kompilacja warunkowa. Operandami wyrażeń logicznych mają postać defined(*tekst*)
#elif	Łańcuch tekstowy, wyrażenie logiczne	
#ifdef	Łańcuch tekstowy	
#ifndef		
#endif		

Definicja a deklaracja (*było wcześniej!)*

Deklaracja: „gdzieś w programie jest zmienna numThreads typu int”

```
extern int numThreads;
```

Definicja: „oto zmienna numThreads typu int”

```
int numThreads;
```

(relikt C) Definicja statyczna: „oto zmienna numThreads typu int, ale to nasza tajemnica”

```
static int numThreads;
```

Wielokrotna deklaracja - ok, wielokrotna definicja - błąd

Definicja a deklaracja funkcji

Deklaracja:

```
int getNumThreads();
```

Definicja:

```
int getNumThreads() {
   return sysconf(_SC_NPROCESSORS_CONF);
}
```

```cpp
#include <iostream>
using namespace std;
// first name space

namespace first_space{
   void func(){
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space{
   void func(){
      cout << "Inside second_space" << endl;
   }
}

int main ()
{

   // Calls function from first name space.
   first_space::func();

   // Calls function from second name space.
   second_space::func();
   return 0;
}
```

Pliki źródłowe i nagłówkowe

Typowa praktyka:

- Kiedy tylko możliwe – chowaj funkcje, klasy i zmienne globalne (*static, namespace…)*

- Pozostałe funkcje, klasy i zmienne
  - Deklaracje w plikach nagłówkowych (*.h/*.hpp)
  - Definicje w plikach źródłowych (*.c/*.cpp/*.cxx)

Pliki nagłówkowe

Instrukcja **`#include`** dołącza zawartość pliku nagłówkowego

`#include <header.h>` – szuka w katalogach systemowych (patrz opcja **`-Idirectory`**)

`#include "header.h"` – szuka w aktualnym katalogu

Deklaracja

```
int coefficients( double &, double &, double & );
int roots( double, double, double, double &, double & );
```

Definicja

```
int coefficients( double & A, double & B, double & C ){
  int code;

  cout << "Enter coefficients for a quadratic equation" << endl;
  cout << "(enter A, B and C, or enter Ctrl-d to halt):" << endl;
  cin >> A >> B >> C;

  if (cin.eof())
    code = 0;
  else
    code = 1;
  return code;
}
```

Include guards

Sposób 1

```
#ifndef THREAD_H
#define THREAD_H
typedef int ThreadNumberType;
ThreadNumberType getNumThreads();
#endif /* THREAD_H */
```

Sposób 2

```
#pragma once
typedef int ThreadNumberType;
ThreadNumberType getNumThreads();
```

```
$  g++ plik_zrodlowy.cpp -o program  -O0
```
Brak optymalizacji

```
$  g++ plik_zrodlowy.cpp -o program  -O1
```
Częściowa optymalizacja

```
$  g++ plik_zrodlowy.cpp -o program  -O2
```
Pełna optymalizacja

```
$  g++ plik_zrodlowy.cpp -o program  -O3
```
Agresywna optymalizacja

```
$  g++ plik_zrodlowy.cpp -o program  -Os
```
Optymalizacja rozmiaru

```
$  g++ plik_zrodlowy.cpp -o program  -Ofast
```
Lepiej unikać ☺

`-march=native` - optymalizacja pod konkretny procesor