

Podstawy programownia (w języku C++)

Wstęp do programowania

Marek Marecki

Polsko-Japońska Akademia Technik Komputerowych

16 października 2020

VON NEUMANN

1. CPU
2. RAM
3. pamięć masowa
4. I/O

OVERVIEW

Rys historyczny

Wstęp do komputerów

Wstęp do języków programowania

Składniki języka

C++

OVERVIEW

Rys historyczny

Wstęp do komputerów

Wstęp do języków programowania

Składniki języka

C++

I/O

CONTROL FLOW - CONCURRENCY AND PARALLELISM

SKŁADNIKI JĘZYKA C++

Współbieżność w C++ jest realizowana za pomocą *wątków*. Tym samym mechanizmem jest realizowana *równoległość* przetwarzania (*parallelism*).

Współbieżność można też zaimplementować na własną rękę, ale wymaga to znacznie większego nakładu pracy.

CONTROL FLOW - CONCURRENCY AND PARALLELISM (std::thread)

SKŁADNIKI JĘZYKA C++

```
auto display_greeting(std::string const name) -> void
{
    std::cout << ("Hello, " + name + "!\n");
}

auto t1 = std::thread{display_greeting, "Joe"};    // Armstrong
auto t2 = std::thread{display_greeting, "Bjarne"}; // Stroustrup

/*
 * Threads must be joined into the parent thread, or
 * the program will crash.
 */
t1.join();    // joining thread is blocked until joined
              // thread terminates
t2.join();
```

CONTROL FLOW - NONDETERMINISM

SKŁADNIKI JĘZYKA C++

Niedeterminizm jest nieodłączną cechą równoległości – nie mamy gwarancji w jakiej kolejności będą względem siebie wykonywać się operacje w *różnych* wątkach.

Niedeterminizm wewnątrz wątku możemy uzyskać generując liczby losowe. W tym celu można użyć `std::random_device` lub odczytać n bajtów z pliku `/dev/urandom`.

CONTROL FLOW - NONDETERMINISM (std::random_device)

SKŁADNIKI JĘZYKA C++

```
std::random_device rd;
std::uniform_int_distribution<int> d20 (1, 20);

constexpr auto CRITICAL_SUCCESS = 20;
constexpr auto CRITICAL_FAILURE = 1;

auto const x = d20(rd);

if (x == CRITICAL_SUCCESS) {
    std::cout << "you kill the monster in a single blow!\n";
} else if (x == CRITICAL_FAILURE) {
    std::cout << "you wound yourself with your own sword!\n";
} else {
    std::cout << "roll for damage.\n";
}
```

CONTROL FLOW - EXCEPTIONS

SKŁADNIKI JĘZYKA C++

Mechanizmem dedykowanym sygnalizacji i obsługi błędów w C++ są *wyjątki*. Wyjątek może być rzucony (zasygnalizowany) słowem kluczowym `throw`; obsługa wyjątków odbywa się w bloku `try-catch`.

C++ pozwala na użycie dowolnego typu jako wyjątku.

CONTROL FLOW - EXCEPTIONS

SKŁADNIKI JĘZYKA C++

```
auto search_your_feelings(std::string father) -> void
{
    if (father == "Darth Vader") {
        throw std::string{"NO!!! NO!!!"};
    }
}

/* ... */

try {
    luke.search_your_feelings(lord_vader);
} catch (std::string const& error) {
    std::cerr << ("operation failed: " + error + '\n');
}
```

DATA STRUCTURES - BIBLIOTEKA STANDARDOWA

SKŁADNIKI JĘZYKA C++

Biblioteka standardowa (ang. *standard library*) języka C++ zawiera wiele struktur danych takich jak `std::vector` (sekwencja o zmiennej długości), `std::queue` (kolejka FIFO), `std::map` (struktura mapująca klucze na wartości), `std::string` (napis), `std::pair`, itd.

Warto używać struktur (a także funkcji) z biblioteki standardowej żeby oszczędzić sobie pracy.

DATA STRUCTURES - WŁASNE TYPY DANYCH

SKŁADNIKI JĘZYKA C++

Programista C++ może definiować również własne typy danych: struktury i klasy, oraz wyliczenia.

Klasy (class) różnią się od struktur (struct) tylko i wyłącznie tym, że ich pola są domyślnie publiczne.

Wyliczenia słabe (enum) są typu int, ich wartości są globalne, i mają automatycznie zdefiniowane operacje arytmetyczne (np. sumę bitową). Są przydatne przy definiowaniu flag, które można łączyć.

Wyliczenia silne (enum class) różnią się od słabych tym, że są "swojego własnego" typu, ich wartości nie są globalne, oraz nie mają automatycznie zdefiniowanych operacji arytmetycznych. Są przydatne przy definiowaniu rozdzielnych stanów.

DATA STRUCTURES - STRUKTURY (struct)

SKŁADNIKI JĘZYKA C++

```
struct being_with_legs {
    std::string const name;
    size_t const legs;

    being_with_legs(std::string, size_t);
};

being_with_legs::being_with_legs(std::string n, size_t l)
    : name{std::move(n)}
    , legs{l}
{}

/* ... */

auto const snake = being_with_legs{ "snake", 0 };
auto const human = being_with_legs{ "human", 2 };
auto const spider = being_with_legs{ "spider", 8 };
```

DATA STRUCTURES - WYLICZENIA (enum class)

SKŁADNIKI JĘZYKA C++

```
enum class meal_kind {  
    BREAKFAST,  
    DINNER,  
    SUPPER,  
};  
  
auto is_most_important_meal_of_the_day(meal_kind const meal) -> bool  
{  
    return (meal == meal_kind::BREAKFAST);  
}
```

DATA STRUCTURES - WYLICZENIA (enum)

SKŁADNIKI JĘZYKA C++

```
enum some_flags_type {
    SOME_FLAG_READ,
    SOME_FLAG_WRITE,
    SOME_FLAG_NONBLOCK,
    SOME_FLAG_BUFFERED,
    SOME_FLAG_UNBUFFERED,
};

constexpr auto SOME_FLAG_DEFAULT = SOME_FLAG_READ
                                   | SOME_FLAG_WRITE
                                   | SOME_FLAG_BUFFERED;

// We want read-only, non-blocking, unbuffered descriptor.
auto const mode = SOME_FLAG_READ
                  | SOME_FLAG_NONBLOCK
                  | SOME_FLAG_UNBUFFERED;
```

I/O

SKŁADNIKI JĘZYKA C++

C++ korzysta z mechanizmów I/O dostarczanych przez API systemu operacyjnego (np. Linux), ale część z nich opakowuje w swoje własne abstrakcje zapewniając programom przenośność.

I/O - STANDARD STEAMS

SKŁADNIKI JĘZYKA C++

W momencie uruchomienia dla większości programów tworzone są 3 standardowe strumienie: wejścia, wyjścia, i błędów.

`std::cin` standardowy strumień wejścia, służy do odczytu danych podawanych przez użytkownika w konsoli tekstowej (*file descriptor 0*)

`std::cout` standardowy strumień wyjścia, służy do prezentacji wyników działania programu w konsoli tekstowej (*file descriptor 1*)

`std::cerr` standardowy strumień błędów, służy do prezentacji błędów działania i awarii programu w konsoli tekstowej (*file descriptor 2*)

I/O - STANDARD STEAMS

SKŁADNIKI JĘZYKA C++

```
{
    std::string line;

    // read a line of text from standard input
    std::getline(std::cin, line);
}

// display a message to inform user of what is happening...
std::cerr << "connecting to server...\n";

// ...or notify them about errors
std::cerr << "connection failed\n";

// display results of program's work
std::cout << downloaded_data << '\n';
```

I/O - FILES

SKŁADNIKI JĘZYKA C++

C++ definiuje typy `std::ifstream` (*input file stream*) i `std::ofstream` (*output file stream*) w bibliotece standardowej.

Jeśli ich interfejs nie jest wystarczający zawsze można użyć interfejsu platformy, np. wywołań systemowych definiowanych przez standard POSIX – `open(3)`, `write(3)`, `read(3)`, i `close(3)`.

I/O - FILES (std::ifstream AND std::ofstream)

SKŁADNIKI JĘZYKA C++

```
auto path = std::string{"./data.txt"};

{
    // write line to a file
    auto out = std::ofstream{ path };
    if (out.good()) {
        out << "Hello, World!\n";
    }
}

{
    // read line from a file
    auto in = std::ifstream{ path };
    if (in.good()) {
        auto line = std::string{};
        std::getline(in, line);
        std::cout << line << "\n";
    }
}
```


I/O - SIEĆ

SKŁADNIKI JĘZYKA C++

Komunikacja po sieci odbywa się z wykorzystaniem mechanizmów I/O dostarczanych przez standard POSIX – `socket(2)`, `bind(2)`, `listen(2)`, `accept(2)`, `connect(2)`, `inet_pton(3)`, `connect(3)`, `shutdown(3)`, `close(3)`.

I/O - SIEĆ (NAGŁÓWKI)

SKŁADNIKI JĘZYKA C++

Pliki nagłówkowe, które należy dodać na początku pliku z kodem źródłowym żeby móc bez przeszkód korzystać z funkcji pozwalających na komunikację po sieci:

```
#include <arpa/inet.h>
#include <endian.h>
#include <netinet/ip.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
```


