Podstawy programownia (w języku C++)

Ćwiczenia

Marek Marecki

1 stycznia 2021

Spis treści

Spis listingów

1 Podstawy

```
#include <iostream>

auto main() -> int
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Listing 1: Hello, World!

1.0.0.1 Hello, World! Zmodyfikuj program z listingu 1 tak żeby wyświetlał twoje imię i nazwisko, lub jakiś inny wybrany tekst.

```
#include <iostream>
#include <string>

auto ask_user_for_integer(std::string const prompt) -> int
{
    if (not prompt.empty()) {
        std::cout << prompt;
    }
    auto value = std::string{};
    std::getline(std::cin, value);
    return std::stoi(value);
}
```

Listing 2: Hello, World!

1.0.0.2 Dodawanie Wykorzystując funkcję z listingu 2 napisz program, który pobierze od użytkownika dwie liczby i doda je do siebie. Wynik wydrukuj na `std::cout`.

1.0.0.3 Mnożenie Wykorzystując funkcję z listingu 2 napisz program, który pobierze od użytkownika dwie liczby i pomnoży je przez siebie. Wynik wydrukuj na `std::cout`.

1.0.0.4 Większa liczba Wykorzystując funkcję z listingu 2 napisz program, który pobierze od użytkownika dwie liczby i wydrukuje większą z nich. Wynik wydrukuj na `std::cout`.

1.0.0.5 Wartość absolutna Napisz program, który pobierze od użytkownika liczbę i poda jej wartość absolutną. Wynik wydrukuj na `std::cout`.

```
./program 2 2
2 == 2
./program 0 3
0 < 3
./program 1 -1
1 > -1
```

Listing 3: relacja między liczbami

1.0.0.6 Relacja między liczbami Wykorzystując funkcję z listingu 2 napisz program, który pobierze od użytkownika dwie liczby i wydrukuje relację między nimi tak jak na listingu 3. Wynik wydrukuj na `std::cout`.

1.0.0.7 Dodatnia-nieujemna-ujemna Napisz program, który pobierze od użytkownika liczbę i poda następujący wynik:

1. `1` jeśli liczba jest dodatnia
2. `0` jeśli liczba jest zerem
3. `-1` jeśli liczba jest ujemna

Wynik wydrukuj na `std::cout`.

1.0.0.8 Największa Napisz program, który pobierze od użytkownika trzy liczby i wydrukuje największą. Wynik wydrukuj na `std::cout`.

2 Pętle

2.0.0.1 Lista liczb Napisz program, który pobierze od użytkownika dwie liczby (`a` i `b`), a następnie wydrukuje listę liczb większych lub równych `a` i mniejszych od `b`. Wynik wydrukuj na `std::cout`.

2.0.0.2 Lista liczb (2) Rozwiń program z poprzedniego zadania tak żeby pobierał trzecią liczbę (`c`) i drukował jedynie liczby podzielne przez `c`. Upewnij się, że program odrzuci `c` równe 0. Wynik wydrukuj na `std::cout`.

2.0.0.3 Lista liczb (3) Rozwiń program z zadania 2.0.0.1 tak żeby pobierał liczbę `s` i użył jej jako kroku pętli. Upewnij się, że program działa też dla ujemnej liczby `s`. Upewnij się, że program odrzuci krok o wartości 0. Wynik wydrukuj na `std::cout`.

2.0.0.4 Liczba pierwsza Napisz program, który pobierze od użytkownika liczbę i sprawdzi czy jest ona liczbą pierwszą. Wynik wydrukuj na `std::cout`.

2.0.0.5 Suma liczb pierwszych Napisz program, który pobierze od użytkownika liczbę i sprawdzi czy jest ona liczbą pierwszą. Jeśli tak, to niech poda sumę liczb pierwszych mniejszych lub równych podanej liczbie. Wynik wydrukuj na `std::cout`.

```
./program 2 2 0 3 8 -1
2 == 2
2 > 3
2 < 3
2 < 8
2 > -1
```

Listing 4: relacja między liczbami (2)

2.0.0.6 Relacja między liczbami (2) Rozwiń program z zadania 1.0.0.6 tak, żeby porównywał więcej liczb naraz, tak jak na listingu 4. Wynik wydrukuj na `std::cout`.

2.0.0.7 Suma podzielnych Napisz program, który pobierze od użytkownika dwie liczby: limit i dzielnik. Niech program obliczy sumę wszystkich liczb większych od zera, ale mniejszych lub równych *limitowi*, które są podzielne przez *dzielnik*. Wynik wydrukuj na `std::cout`.

```
for (auto i = 0; i < 42; ++i) {
    // do something
}
```

Listing 5: pętla for

```
auto i = 0;
while (i < 42) {
    // do something
    ++i;
}
```

Listing 6: pętla while

```
auto i = 0;
do {
    // do something
    ++i;
} while (i < 42);
```

Listing 7: pętla do-while

2.0.0.8 Silnia (for) Wykorzystując pętlę `for` (patrz listing 5) napisz program, który pobierze od użytkownika liczbę i obliczy jej silnię. Wynik wydrukuj na `std::cout`.

2.0.0.9 Silnia (while) Wykorzystując pętlę `while` (patrz listing 6) napisz program, który pobierze od użytkownika liczbę i obliczy jej silnię. Wynik wydrukuj na `std::cout`.

2.0.0.10 Silnia (do-while) Wykorzystując pętlę `do-while` (patrz listing 7) napisz program, który pobierze od użytkownika liczbę i obliczy jej silnię. Wynik wydrukuj na `std::cout`.

```
./program-prostokat 2 4
****
****
```

Listing 8: prostokąt z gwiazdek

```
./program-odwrocony-trojkat 4
*
**
***
****
```

Listing 9: trójkąt gwiazdek

```
./program-odwrocony-trojkat 4
****
***
**
*
```

Listing 10: odwrócony trójkąt gwiazdek

2.0.0.11 Rysowanie figury (prostokąt) Wykorzystując dowolną pętlę napisz program, który pobierze z wiersza poleceń wymiary prostokąta i narysuje go. Wynik wydrukuj na `std::cout`. Przykładowe uruchomienie na listingu 8.

2.0.0.12 Rysowanie figury (trójkąt) Wykorzystując dowolną pętlę napisz program, który pobierze z wiersza poleceń wymiary trójkąta i narysuje go. Wynik wydrukuj na `std::cout`. Przykładowe uruchomienie na listingu 9.

2.0.0.13 Rysowanie figury (odwrócony trójkąt) Wykorzystując dowolną pętlę napisz program, który pobierze z wiersza poleceń wymiary „odwróconego trójkąta” (tj. niech wierzchołek będzie na dole, patrz listing 10) i narysuje go. Wynik wydrukuj na `std::cout`.

```
./program-pusty-kwadrat 4
****
*  *
*  *
****
```

Listing 11: pusty kwardat

2.0.0.14 Rysowanie figury (pusty kwadrat) Napisz program, który pobierze z wiersza poleceń wymiary figury, a potem narysuje „pusty kwardat" (patrz listing 11). Wymiar nie może być mniejszy niż 3. Wynik wydrukuj na `std::cout`.

3 Tablice i wskaźniki

Dla każdego zadania w funkcji `main()` przetestuj działanie kodu.

3.1 Tablice

```
int array_of_int[2]; // array with two elements
array_of_int[0] = 1; // first element now contains 1
array_of_int[1] = 2;

// read an element and print it on standard output
std::cout << array_of_int[0] << "\n";

// change value of an element
int i = 0;
array_of_int[i] = 42;
```

Listing 12: tworzenie i użycie tablicy

3.1.0.1 Inicjalizacja Napisz funkcję `auto init(int a[], int n) -> void`, która zainicjalizuje zerami tablicę `a` o rozmiarze `n`.

3.1.0.2 Inicjalizacja (2) Napisz funkcję `auto iota(int a[], int n, int start) -> void`, która zainicjalizuje tablicę `a` o rozmiarze `n` kolejnymi liczbami całkowitymi zaczynając od `start`.
Przykład: dla wywołania `iota(a, 4, 5)` tablica `a` zawierałaby liczby 5, 6, 7, i 8.

3.1.0.3 Suma Napisz funkcję `auto asum(int a[], int n) -> int`, która zwróci sumę liczb w tablicy `a` o rozmiarze `n`.

3.1.0.4 Minimum Napisz funkcję `auto amin(int a[], int n) -> int`, która zwróci indeks najmniejszej wartości w tablicy `a` o rozmiarze `n`.

3.1.0.5 Maksimum Napisz funkcję `auto amax(int a[], int n) -> int`, która zwróci indeks największej wartości w tablicy `a` o rozmiarze `n`.

3.1.0.6 Przeszukiwanie Napisz funkcję `auto search(int a[], int n, int needle) -> int`, która w tablicy `a` o rozmiarze `n` będzie szukać wartości równej `needle`. Jeśli tablica zawiera taką wartość niech funkcja zwróci jej indeks; w przeciwnym wypadku niech zwróci -1.

3.1.0.7 Sortowanie Napisz funkcję `auto sort_asc(int a[], int n) -> void`, która ułoży liczby w tablicy `a` o rozmiarze `n` w kolejności rosnącej. Użyj dowolnego algorytmu.

3.1.0.8 Sortowanie (2) Napisz funkcję `auto sort_desc(int a[], int n) -> void`, która ułoży liczby w tablicy `a` o rozmiarze `n` w kolejności malejącej. Użyj dowolnego algorytmu.

3.1.0.9 Sortowanie (szybkie) Napisz funkcję `auto quicksort(int a[], int n) -> void`, która posortuje tablicą `a` o rozmiarze `n` za pomocą algorytmu Quicksort[1].

[1] `https://en.wikipedia.org/wiki/Quicksort`

3.2 Wskaźniki

```
auto x  = int{42};  // an integer
auto xp = &x;       // a pointer to an integer (new style)
int* xo = &x;       // a pointer to an integer (old style)
```

Listing 13: pobranie wskaźnika

```
auto x = int{42};
auto xp = &x;   // xp contains the address of the x variable
auto xd = *xp;  // xd contains a copy of the variable to which
                // xp is pointing

*xp = 64;       // x contains 64 after this assignment
```

Listing 14: dereferencja wskaźnika

3.2.0.1 Pobranie wskaźnika Napisz program, w którym w funkcji `main()` utworzysz zmienną typu `std::string`, której wartością będzie `Hello, World!`. Pobierz wskaźnik i wydrukuj adres tej zmiennej w pamięci.
Wynik wydrukuj na `std::cout`.

3.2.0.2 Dereferencja Napisz funkcję `print()`, która będzie jako parametr przyjmować wskaźnik na `std::string`. W funkcji `print()` wydrukuj adres, na który wskazuje wskaźnik oraz napis stojący za tym wskaźnikiem, np. „`1781f89a980 = Hello, World!`”. W funkcji `main()` napisz kod, który wywołuje funkcję `print()`. Wynik wydrukuj na `std::cout`.

```
./program-zamiana
42 64
64 42
```

Listing 15: zamiana

3.2.0.3 Zamiana Napisz funkcję `swap()`, która będzie jako parametr przyjmować dwa wskaźniki na `int`.

W funkcji `main()` napisz kod, który wywołuje funkcję `swap()`. Wydrukuj wartość dwóch testowych liczb przed i po zamianie (patrz listing 15).

Wynik wydrukuj na `std::cout`.

3.2.0.4 `memset(3)` Zaimplementuj funkcję `memset(3)`. Jej opis znajduje się w podręczniku użytkownika systemu. Można go wyświetlić używając polecenia ‘`man 3 memset`’.

3.2.0.5 `memcpy(3)` Zaimplementuj funkcję `memcpy(3)`.

3.2.0.6 `memfrob(3)` Zaimplementuj funkcję `memfrob(3)`.

3.2.0.7 `memrev()` Zaimplementuj funkcję `memrev(void* s, size_t n)`, która odwróci kolejność bajtów w obszarze pamięci o rozmiarze `n`, na który wskazuje wskaźnik `s`.

3.2.0.8 `memrand()` Zaimplementuj funkcję `memrev(void* s, size_t n)`, która wypełni losowymi bajtami obszar pamięci o rozmiarze `n`, na który wskazuje wskaźnik `s`.

```
auto some_function(int const) -> void;

auto fp = &some_function; // take address of some_function
(*fp)(42);               // dereference is needed for the call
```

Listing 16: wskaźnik do funkcji

3.3 Tablice i wskaźniki

3.3.0.1 Wywołanie przez wskaźnik Napisz funkcję o podanej sygnaturze:
`auto call_with_random_int(void (*fp)(int const)) -> void;`
Niech funkcja `call_with_random_int()` pobiera wskaźnik na funkcję (patrz listing 16), a następnie wywoła ją podając jej losową liczbę całkowitą jako argument.

3.3.0.2 `all()` Napisz funkcję o podanej sygnaturze:
`auto all(void* a[], size_t n, bool (*fp)(void*)) -> bool;`
Niech zwraca ona `true` jeśli dla każdego elementu tablicy `a` o rozmiarze `n` funkcja, której adres przekazany jest w parametrze `fp` zwraca `true`.

3.3.0.3 `any()` Napisz funkcję o podanej sygnaturze:
`auto any(void* a[], size_t n, bool (*fp)(void*)) -> bool;`
Niech zwraca ona `true` jeśli dla co najmniej jednego elementu tablicy `a` o rozmiarze `n` funkcja, której adres przekazany jest w parametrze `fp` zwraca `true`.

3.3.0.4 Drukowanie Napisz funkcję `auto fpprint(void* a[], size_t n, void (*fp)(void*)) -> void`, która wydrukuje elementy tablicy `a` o rozmiarze `n`. Elementy powinny być drukowane używając funkcji, której adres przekazany jest we wskaźniku `fp`. Używanie wskaźników do funkcji pokazane jest na listingu 16.

Użyj tej funkcji do wydrukowania tablicy wartości `int`, oraz tablicy wartości `std::string`. Będzie to wymagało napisania dodatkowych funkcji drukujących odpowiednie te typy, ale pobierających je przez wskaźnik `void*`: `auto print_int(void*) -> void`, `auto print_str(void*) -> void`.

Wynik (dla tablicy dwóch napisów) powinien wyglądać mniej więcej tak:

```
[0] = Hello
[1] = World
```

3.3.0.5 Wyszukiwanie Napisz funkcję o podanej sygnaturze:
`auto fpsearch(void* a[], size_t n, bool (*fp)(void*, void*), void* needle) -> ssize_t;`
Niech ta funkcja przeszuka tablicę `a` o rozmiarze `n` i zwróci indeks elementu, który będzie równy wartości, której adres przekazany jest w parametrze `needle`.

Do porównaniem wartości należy wykorzystać funkcję, której adres przekazany jest w parametrze `fp`. Funkcja ta otrzymuje dwa wskaźniki `void*`, które wskazują na dwie wartości - element tablicy, i poszukiwaną wartość.

3.3.0.6 Sortowanie Napisz funkcję o podanej sygnaturze:
`auto fpsort(void* a[], size_t n, int (*fp)(void*, void*)) -> void;`
Niech ta funkcja posortuje tablicę `a` o rozmiarze `n`. Do porównywania elementów należy wykorzystać funkcję, której adres przekazany jest w parametrze `fp`.

4 Struktury danych

Dla każdego zadania powinna być napisana funkcja `main()`, która będzie testować działanie zaimplementowanej struktury (lub struktur) danych wywołując jej funkcje składowe.

O ile nie jest powiedziane inaczej, w funkcji `main()` dane (np. długości boków, ilości żyć, itd.) mogą być wpisane „na sztywno" podczas tworzenia obiektów danej struktury danych i nie muszą być wczytywane od użytkownika.

```
struct A_type {
    std::string member_variable;
    int const member_constant;

    auto member_function() -> std::string;
};

auto A_type::member_function() -> std::string
{
    return (member_variable
            + ": "
            + std::to_string(member_constant));
}
```

Listing 17: struct

```
struct Foo {
    std::string const bar;

    // ctor's name must be the same as struct's name
    Foo(std::string);
};

Foo::Foo(std::string b)
    : bar{std::move(b)}
{}
```

Listing 18: konstruktor

4.0.0.1 Kwadrat Zaprojektur strukturę danych reprezentującą kwadrat. Niech posiada ona jedno stałe pole typu `float`, które będzie reprezentowało długość boku kwadratu. Wartość musi być inicjalizowana w konstruktorze.

Struktura ma posiadać dwie funkcje składowe:

1. `auto area() const -> float` zwracającą pole kwadratu
2. `auto draw() const -> void` rysującą kwadrat na ekranie (tak jak w zadaniu 2.0.0.11)

4.0.0.2 Prostokąt Zaprojektur strukturę danych reprezentującą prostokąt. Niech ma ona dwa stałe pola reprezentujące długości dwóch boków prostokąta.

Struktura ma posiadać dwie funkcje składowe:

1. `auto area() const -> float` zwracającą pole prostokąta
2. `auto draw() const -> void` rysującą prostokąt na ekranie (tak jak w zadaniu 2.0.0.11)

4.0.0.3 Prostokąt (2) Zmodyfikuj kod poprzedniego zadania tak, aby możliwe było dodanie funkcji składowej `auto scale(float const x, float const y) -> void`. Funkcja `resize()` ma skalować rozmiar boków prostokąta przez pewien mnożnik.
Przykłady:

1. `rect.scale(1.5f, 1.5f)` wydłuży oba boki prostokąta o 50%
2. `rect.scale(2.0f, 0.5f)` wydłuży jeden bok prostokąta dwukrotnie, a drugi skróci dwukrotnie

4.0.0.4 Temperatura (°C) Zaprojektuj struktorę danych reprezentującą temperaturę w stopniach Celsjusza[2] (patrz listing 19). Jeśli w konstrutorze zostanie podana temperatura mniejsza niż -273.15 °C to powinna zostać ona „obcięta" do tej wartości. Nie ma górnego ograniczenia.

Dodaj do struktury funkcję składową `auto to_string() const -> std::string`, która będzie produkować napisy reprezentujące temperaturę (tj. ilość stopni, spacja, i '`°C`').

```
struct Celsius {
    // add necessary fields here
    // add ctor here
    auto to_string() const -> std::string;
};
```

Listing 19: struktura reprezentująca temperaturę w °C

4.0.0.5 Temperatura (°F) Tak jak w zadaniu 4.0.0.4, ale dla temperatury w stopniach Fahrenheita[3].

4.0.0.6 Temperatura (K) Tak jak w zadaniu 4.0.0.4, ale dla temperatury w Kelwinach[4]. Pamiętaj, że temperatura w Kelwinach nie może spaść poniżej zera (zero Kelwinów to tzw. „zero absolutne").

4.0.0.7 Arytmetyka Do struktur danych napisanych w zadaniach 4.0.0.4, 4.0.0.5, i 4.0.0.6 dodaj operatory dodawania (`+`) i odejmowania (`-`). Przykładowa implementacja operatorów pokazana jest na listingu 20.

4.0.0.8 Porównania Do struktur danych napisanych w zadaniach 4.0.0.4, 4.0.0.5, i 4.0.0.6 dodaj operatory porównania: `==` (równe), `!=` (nierówne), `<`, `<=` (mniejsze lub równe), `>`, i `>=` (większe lub równe). Przykładowa implementacja operatorów pokazana jest na listingu 20.

4.0.0.9 Konwersje Do struktur danych napisanych w zadaniach 4.0.0.4, 4.0.0.5, i 4.0.0.6 dodaj konstruktory pozwalające na konwersję temperatur w każdy możliwy sposób. Napisz kod, który sprawdzi czy dodawanie temperatur w różnych skalach działa poprawnie. Konwersje do napisania to:

1. z °C na °F (sygnatura `Fahrenheit(Celsius const)`)
2. z °C na K (sygnatura `Kelvin(Celsius const)`)
3. z °F na °C (sygnatura `Celsius(Fahrenheit const)`)
4. z °F na K (sygnatura `Kelvin(Fahrenheit const)`)
5. z K na °C (sygnatura `Celsius(Kelvin const)`)
6. z K na °F (sygnatura `Fahrenheit(Kelvin const)`)

[2] `https://en.wikipedia.org/wiki/Celsius`
[3] `https://en.wikipedia.org/wiki/Fahrenheit`
[4] `https://en.wikipedia.org/wiki/Kelvin`

```
struct Foo {
    int field { 0 };

    auto operator+(Foo const&) const -> Foo;
    auto operator<(Foo const&) const -> Foo;
    Foo(int const);
};

Foo::Foo(int const x): field{x}
{}

auto Foo::operator+(Foo const& x) const -> Foo
{
    return Foo{ field + x.field };
}

auto Foo::operator<(Foo const& x) const -> bool
{
    return field < x.field;
}
```

Listing 20: implementacja operatora

4.0.0.10 Kąt (stopnie) Zaprojektuj strukturę danych, która będzie reprezentować miarę kąta[5] z użyciem miary stopniowej[6]: `Arc_degree`. Dodaj do struktury konstruktor, który zapewni, że wartość kąta będzie zawsze między 0°, a 360°. Dodaj do struktury operatory arytmetyczne dodawania i odejmowania, oraz operatory porównania (patrz zadanie 4.0.0.8).

4.0.0.11 Kąt (radiany) Zaprojektuj strukturę danych, która będzie reprezentować miarę kąta w radianach[7]: `Radian`. Dodaj dla niej operatory arytmetyczne i porównania, oraz zadbaj o możliwość konwersji między strukturami `Radian` i `Arc_degree`. Przetestuj swój kod.

[5] https://pl.wikipedia.org/wiki/Miara_k%C4%85ta
[6] https://pl.wikipedia.org/wiki/Stopień_(k%C4%85t)
[7] https://pl.wikipedia.org/wiki/Radian