# PODSTAWY C++

## ELEMENTY JĘZYKA

CODERS SCHOOL

MATEUSZ ADAMSKI

ŁUKASZ ZIOBROŃ

# AGENDA

1. Funkcje
2. Instrukcje warunkowe
3. Pętle
4. Zakresy
5. Stałe - `const`

# REPOZYTORIUM

coders-school/cpp-fundamentals

# PODSTAWY C++

## FUNKCJE

CODERS SCHOOL

# FUNKCJE

Funkcja jest to fragment programu, któremu nadano nazwę i który możemy wykonać poprzez podanie jego nazwy oraz ewentualnych argumentów.

Funkcja == podprogram == procedura

Przykładowo, w trakcie jazdy na rowerze naszą główną funkcją jest przemieszczanie się z punktu a do b. Jednak wykonujemy także kilka podprogramów, jak zmiana biegów, hamowanie, rozpędzanie, skręcanie. Podobnie w programie możemy wydzielić konkretne zachowania i przenieść je do funkcji, które nazwiemy tak, by sugerowały co robią. Ważne, aby funkcja robiła tylko jedną rzecz. Jedna funkcja zmienia biegi, druga hamuje, trzecia skręca.

# SYGNATURY FUNKCJI (DEKLARACJE)

`void fun(int)` - funkcja ma nazwę fun, nic nie zwraca a przyjmuje jeden argument typu int.

## ODGADNIJCIE SYGNATURY PO OPISIE

Funkcja o nazwie foo, która nic nie zwraca a przyjmuje jeden argument typu double.

```
void foo(double)
```

Funkcja o nazwie bar, która zwraca typ double a przyjmuje 2 argumenty. Pierwszy to float, a drugi to const int (const oznacza, że wartość ta nie może zostać zmodyfikowana).

```
double bar(float, const int)
```

# WYWOŁANIA FUNKCJI

`foo(5.0)` -> wywołujemy funkcję `foo` z argumentem `double`, który jest równy `5.0`

`double result = bar(5.4f, 10)` -> wywołujemy funkcję `bar` z argumentem `float (5.4f)` oraz `int (10)` a jej wynik przypisujemy do zmiennej typu `double` o nazwie `result`.

# ZADANIE

Dopisz brakującą funkcję `multiply`. Ma ona pomnożyć dwie liczby podane jako jej parametry. Pobierz zadanie

```
#include <iostream>

// Write missing function here

int main() {
    std::cout << "4 * 5: " << multiply(4, 5) << "\n";
    std::cout << "10 * 5: " << multiply(10, 5) << "\n";
    std::cout << "-5 * 5: " << multiply(-5, 5) << "\n";

    return 0;
}
```

# PODSTAWY C++

## INSTRUKCJE WARUNKOWE

CODERS SCHOOL

# INSTRUKCJA `if`

Instrukcja warunkowa to nic innego jak zadanie programowi pytania np.:

- Czy otrzymałeś już wszystkie dane?
- Czy życie bossa spadło do 0?
- Czy osiągnięcie zostało zdobyte przez gracza?
- Czy liczba jest większa od maksymalnie dopuszczanej?

# KONSTRUKCJA `if`

```
if (condition) {
    // do sth
}
```

# ŁĄCZENIE WARUNKÓW

A co w przypadku, gdy wiele informacji musi być spełnionych? Możemy połączyć warunki operatorem **lub** (`||`, `or`) bądź **i** (`&&`, `and`)

```
if (are_potatoes_eaten && is_meat_eaten && is_salad_eaten)
```

Wszystkie 3 warunki muszą zostać spełnione

```
if (player_has_20_dex || player_has_18_int || player_has_22_str)
```

W tym przypadku wystarczy spełnić jeden z 3 warunków. Mogą zostać spełnione wszystkie, ale wystarczy by został spełniony jeden dowolny.

# INSTRUKCJA `else`

Jeżeli program może różnie zareagować na spełnienie jakiś warunków możemy zastosować konstrukcje `if else`

```
if (number < 2) {
    critical_miss();
} else if (number < 18) {
    hit();
} else {
    critical_hit();
}
```

# INSTRUKCJA switch/case

```
char option = getInput();
switch (option) {
case 'l':
    goLeft();
    break;
case 'r':
    goRight();
    break;
default:
    exit();
}
```

- `case` oznacza konkretny przypadek
- `break` informuje, że wychodzimy z instrukcji warunkowej `switch` i kontynuujemy dalej program. Jego brak spowoduje, że wykonają się instrukcje z kolejnego `case`.
- `default` jest to miejsce gdzie program dotrze, gdy żaden inny warunek nie zostanie spełniony
- Zmienna sterująca instrukcją `switch/case` musi być typu liczbowego całkowitego czyli np. `int`, `char`, `long` etc. Może być także typu wyliczeniowegp `enum`, który także jest widziany przez program jako liczba całkowita

# ZADANIE

Dopisz funkcję `max`. Ma ona zwracać maksymalną z trzech podanych wartości. Pobierz zadanie

```
#include <iostream>

// Write your function here

int main() {
    std::cout << "max (1, 2, 3): " << max(1, 2, 3) << "\n";
    std::cout << "max (2, 3, 1): " << max(2, 3, 1) << "\n";
    std::cout << "max (3, 2, 1): " << max(3, 2, 1) << "\n";

    return 0;
}
```

# PODSTAWY C++

## PĘTLE

CODERS SCHOOL

# PĘTLE

Pętla służy do powtarzania instrukcji, które chcemy by się wykonały więcej niż raz bez konieczności ich wielokrotnego pisania w kodzie.

Podstawowe pętle: `while`, `for`

# PĘTLA `while`

`while` używamy, gdy chcemy coś wykonać dopóki nie zostanie spełniony jakiś warunek. Przeważnie nie mamy pojęcia, kiedy to nastąpi (nie znamy liczby kroków) np:

- Przeglądamy koszule w Internecie dopóki nie znajdziemy pasującej do nas
- Powtarzamy walkę z tym samym bossem aż go nie pokonamy
- Jemy zupę, aż talerz nie będzie pusty
- Przeszukujemy kontakty w telefonie aż nie znajdziemy interesującej nas osoby

# KONSTRUKCJA PĘTLI `while`

```
while (condition) {
    // Do sth
}
```

# PRZYKŁAD

```
while (a == b) {
    std::cin >> a;
    std::cin >> b;
}
```

# PĘTLA `for`

`for` używamy, gdy chcemy coś wykonać określoną liczbę razy. Przeważnie znamy liczbę kroków np:

- Wypełniamy ankietę składającą się z 10 pytań -> liczba kroków 10
- Przemieszczamy się z punktu A do B -> liczba kroków = dystans / długość kroku
- Piszemy egzamin składający się z 4 zadań -> liczba kroków (jak umiemy to 4, jak nie to jeszcze wykonujemy podprogram `ściągaj`)
- Zapinamy koszule (o ile nie wyrwiemy żadnego guzika)

# KONSTRUKCJA PĘTLI `for`

```
for (variable = initial_value; condition; variable_change) {
    // Do sth
}
```

# PRZYKŁAD

```
for (size_t i = 0; i < 10; i += 2) {
    std::cout << "i: " << i << '\n';
}
```

Każdą pętlę `for` można zamienić na `while` i odwrotnie. Wybieramy wygodniejszy dla nas zapis, zazwyczaj w zależności od znajomości liczby kroków.

Istnieje jeszcze jeden rodzaj pętli. Jaki?

# PĘTLA `do/while`

```
do {
    // Do sth
} while(condition)
```

Kod w pętlach `while` lub `for` może się nie wykonać ani razu, gdy warunek nie będzie nigdy spełniony.

Kod w pętli `do/while` wykona się co najmniej raz.

# ZADANIE

Dopisz funkcję `printString`. Ma ona wypisywać tekst podany jako pierwszy argument tyle razy, jaka jest wartość liczby podanej jako drugi argument. Pobierz zadanie

```
#include <iostream>

// Write your function here

int main() {
    printString("Hello", 5);
    std::cout << "\n";

    printString("AbC", 2);
    std::cout << "\n";

    printString("HiHi ", 6);
    std::cout << "\n";

    return 0;
}
```

# PODSTAWY C++

## ZASIĘG ZMIENNYCH

CODERS SCHOOL

# ZMIENNE LOKALNE

Zmienne lokalne są to zmienne, które są widziane w obrębie jakiegoś zakresu.

```
{
    int local_variable = 5;
    // ...
}
local_variable = 10;   // error -> local_variable doesn't exists
```

Zakres zawsze tworzą nawiasy klamrowe m.in:

- same nawiasy - `{ /* ... */ }`
- ciała funkcji - `void fun() { /* ... */ }`
- instrukcje warunkowe - `if (condition) { /* ... */ }`
- pętle - `while (condition) { /* ... */ }`

# ZMIENNE GLOBALNE

Zmienna globalna, jest widoczna dla wszystkich zakresów. Zawsze możemy się do niej odwołać.

```
int global_value = 5;

void foo() {
    std::cout << global_value;
}

int main() {
    std::cout << global_value;
}
```

Tworzenie zmiennych globalnych zazwyczaj jest złą praktyką.

# CO WYPISZE SIĘ NA EKRANIE?

```
int number = 1;

int main() {
    int number = 2;
    {
        int number = 3;
        std::cout << number;
        std::cout << ::number;
    }
    std::cout << number;
    std::cout << ::number;
}
```

3121

# PRZESŁANIANIE NAZW

- możemy mieć wiele zmiennych o takiej samej nazwie, jeśli będą w różnych zakresach
  - aby unikać niejednoznaczności nie jest to raczej polecane
- nazwa z lokalnego zakresu zawsze przesłania tę z szerszego zakresu (np. globalnego)
- można odwoływać się do nazw z globalnego zakresu stosując `::` (operator zakresu)

# PODSTAWY C++

## STAŁE

CODERS SCHOOL

Stałych w odróżnieniu od zmiennych nie można modyfikować. Można im nadać wartość tylko podczas inicjalizacji obiektu i później nie można jej zmienić.

Stałe oznacza się słowe kluczowym `const` przed nazwą typu.

```
const int a = 42;
a = 43; // compilation error
```

# CODERS SCHOOL