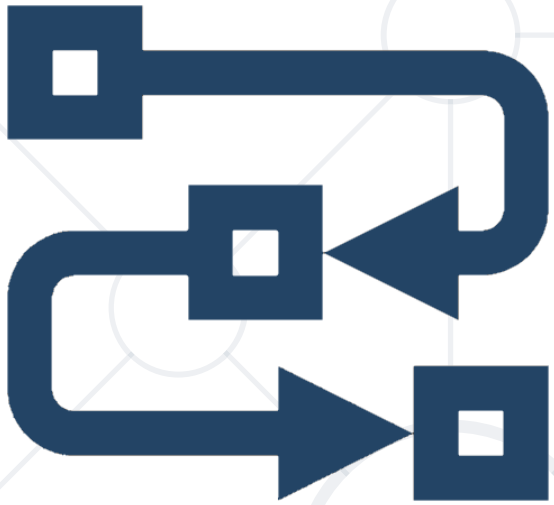
Functions

Defining and Using Functions



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

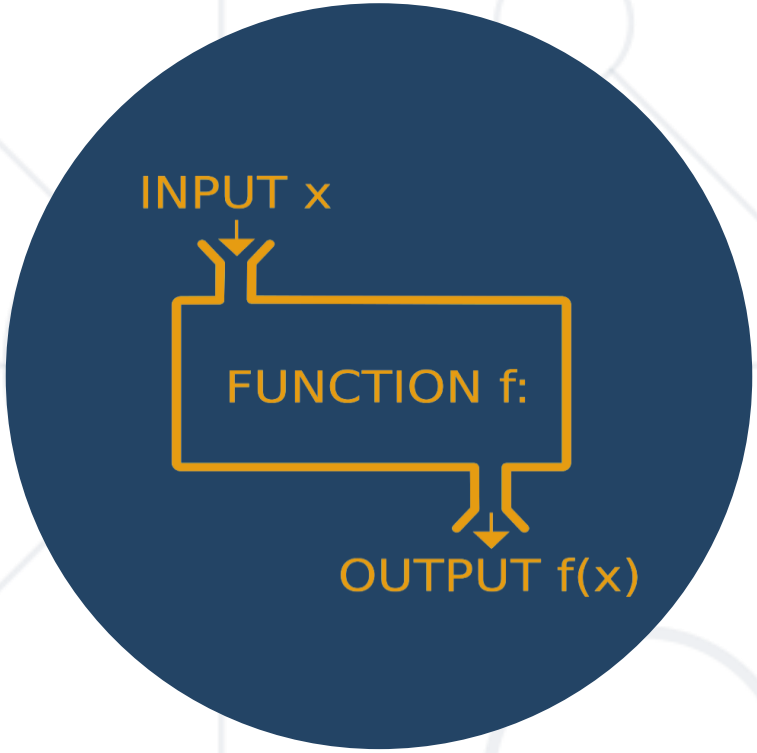
Have a Question?

Software University

sli.do

#cpp-fundamentals

Table of Contents



Functions

What is a Function?

Software University

- **Named block of code**, that performs a specific task
- Can take parameters and return a value
- Sample function **definition**:

```
void printHelloWorld()
{
  cout << "Hello World!" << endl;
}
```

Function named `printHelloWorld`

- Also known as methods (when in classes)
- **`main()` is a function**

Function **body** always surrounded by **{ }**

Why Use Functions?

- More **manageable programming**
  - Splits large problems into small pieces
  - Better organization of the program
  - Improves code readability
  - Improves code understandability
- Avoiding **repeating code**
  - Improves code maintainability
- Code **reusability**
  - Using existing methods several times

Declaring and Calling Functions

Declaring Functions

- **Declaration – function's name, return type and parameters**
  - Can be separate from definition (which includes the code block)
- Parameters: **empty, single or several separated by ,**

Type

Function name

Parameters

```
void printNumber(int number)
{
   cout << number << endl;
}
```

Function body

Calling Functions

- Using functions is almost like using variables, however:
  - You write `()` after them, which could contain parameters
- Most functions return a value – you can use it in an expression
  - `void` functions don't have values

```
void helloWorld()
{
    cout << "Hello World!" << endl;
}
int main()
{
    helloWorld();
    return 0;
}
```

Declaring vs. Defining Functions

Declaring vs Defining Functions

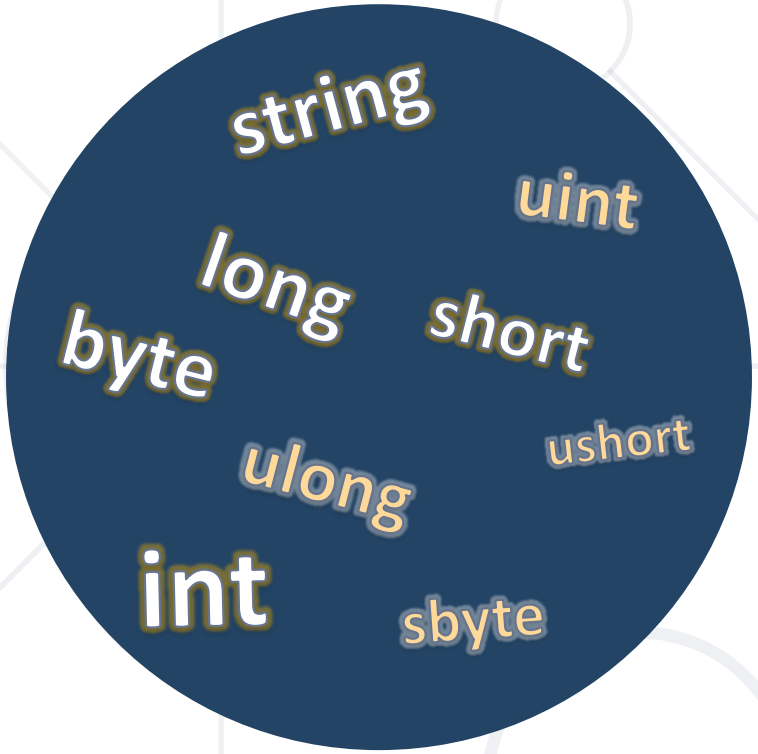
- **Declaration** – tells the compiler there is certain a function
  - **Can be anywhere**
  - **Can appear multiple times**
  - **Same visibility rules as for variables**
- **Definition** – function's execution

```
#include<iostream>
using namespace std;

void helloWorld();

int main()
{
    helloWorld();
    return 0;
}

void helloWorld()
{
cout << "Hello World!" << endl;
}
```



Functions with Parameters

Function Parameters

- Function **parameters** can be of **any data type**
- Parameters are just variables **used in the function's block**

```
void printNumbers(int start, int end)
{
  for (int i = start; i <= end; i++)
  {
    cout << i << endl;
  }
}
```

Multiple parameters separated by comma

- Call the function with **certain values** (**arguments**)

```
int main()
{
  printNumbers(5, 10);
  return 0;
}
```

Passing arguments when called

Parameters and Default Values

- Parameters with default values can be omitted by the caller

```
#include <iostream>
using namespace std;
void countNumbers(int a = 1, int b = 10)
{
    for (int i = a; i <= b; i++)
    {
      cout << i << endl;
    }
}
int main()
{
    countNumbers();
    return 0;
}
```

Returning Values from Functions

Returning Values from Functions

- The `return` keyword immediately **stops** the function's execution – **early exit**
- Returns the **specified value**
  - Non-`void` functions must have a `return` followed by a value

```
int getMax(int a, int b)
{
    if (a > b)
    {
        return a;
    }
    return b;
}
```

Using the Return Values

- Return value can be:
  - **Assigned** to a variable:

```
int max = getMax(5, 10);
```

  - **Used** in expression:

```
double total = getPrice() * quantity * 1.20;
```

Overloading Functions

Overloaded Functions

- Using the same function **name** and **return type** but with different parameter list

```
int getMax(int a, int b)
{
    if (a > b)
    {
        return a;
    }
    return b;
}

int getMax(int a, int b, int c)
{
    return getMax(a, getMax(b, c));
}
```

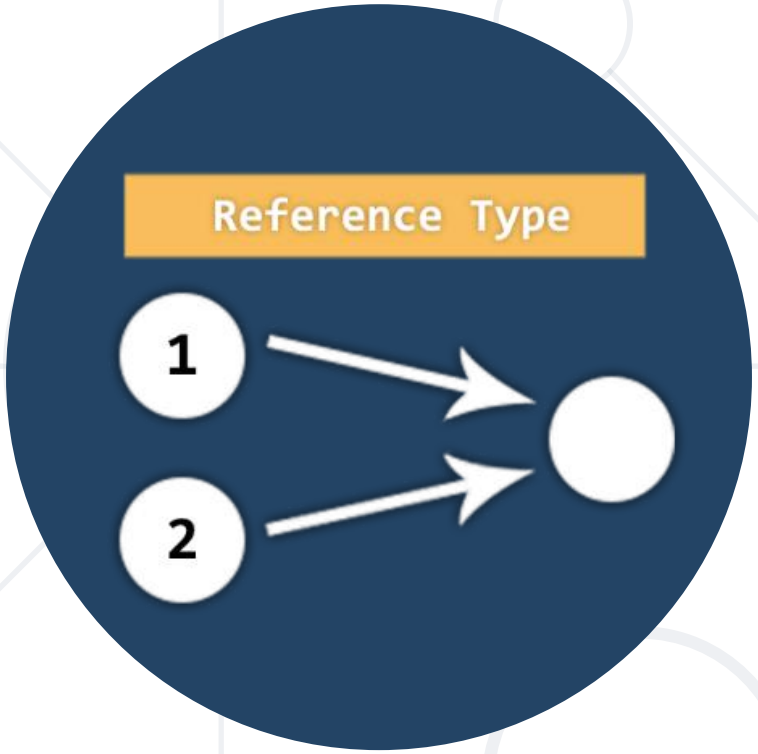
Static Variables Inside Functions

static Variables Inside Functions

Software University

- **static variables** live through entire program, initialized once
- **static variables** can be used inside functions to track state

```
void countNumbers(int a = 1, int b = 10)
{
    static int num = 0;
    for (int i = a; i <= b; i++)
    {
        cout << i << endl;
        num++;
    }
    cout << "Static int -> " << num << endl;
}
```



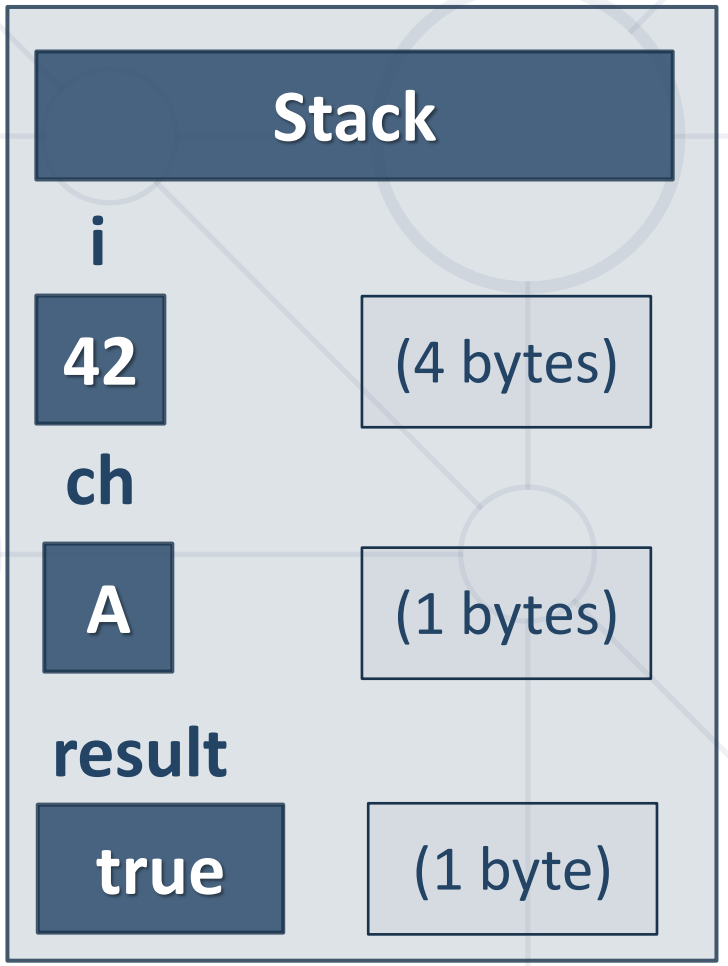
Value vs. Reference Types

Memory Stack and Heap

Value Types

- **Value type** variables hold directly their value
  - **`int`, `float`, `double`, `bool`, `char`…**
- Each variable has its own **copy** of the **value**

```
int i = 42;
char ch = 'A';
bool result = true;
```



Reference Types

- **Reference type** variables hold a reference (pointer / memory address) of the value itself
- Two reference type variables can **reference** the **same variable**
  - Operations on both variables access/modify **the same data**

Value vs Reference Types

pass by reference

cup =

fillCup( )

pass by value

cup =

fillCup( )

Passing By Value vs. Passing By Reference

- Parameters are normally **copies of their originals**
  - **Passing by value**
- To access the caller's variables directly, use **references**
  - Syntax: `DataType& param`
  - **Passing by reference**

```
int square(int num)
{
    num = num * num;
    return num;
}
void swap(int& a, int& b)
{
    int oldA = a; a = b; b = oldA;
}
int main()
{
    int x = 5;
    cout << square(x) << endl; // 25
    cout << x << endl; // 5
    int y = 42;
    swap(x, y);
    cout << x << endl; // 42
    return 0;
}
```

Summary

- Break large programs into simple **functions** that solve small sub-problems
- Functions consist of **declaration** and **body**
- Functions are called by their **name** + **()**
- Functions can accept **parameters**
- Functions can `return` a value or nothing (`void`)

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

top EMPLOYER България Bulgaria 2024 FOR A BETTER WORLD OF WORK

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Questions?

SoftUni

Software University | SoftUni AI | SoftUni Digital | SoftUni Creative | SoftUni Foundation | СофтУни БУДИТЕЛ | Finance Academy

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

Software University

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg