Exercises

# C++ Basics

The following exercises will familiarize you with the basic concepts of C++: Variables and Macros.

# Variables

C++ provides multiple, simple data types (bool, char, int, float and double). You can also use pointer to those types as well as create references.

1. Declare a variable for each of the following data types: short, int and double.

short s = 50;

int i = 100;

double d = 15.5;

1. Display all the properties (value, address and size in bytes) of the variables you have defined (use the operator sizeof() to print the size of those variables in bytes).

cout << "short s: " << s << " | " << &s << " | " << sizeof(s) << endl;

cout << "int i: " << i << " | " << &i << " | " << sizeof(i) << endl;

cout << "double d: " << d << " | " << &d << " | " << sizeof(d) << endl;

1. Declare a pointer for each of the declared variables that will point the variable.

short \*shortP = &s;

int \*intP = &i;

double \*doubleP = &d;

1. Display all the properties (value, address and size in bytes) of the pointers (compare it with the properties of the variables).
2. Declare a reference (second name) for each of the variable.

short &shortReference = s;

int &intReference = i;

double &doubleReference = d;

1. Display all the properties (value, address and size in bytes) of the references (compare it with the properties of the variables).

# Macros

Macros can be used to declare constant values, flags and provide the simple use of function like code. Very quick in execution (it is applied at the start of compilation process) but also very dangerous.

1. Define the constant PI.

#define PI 3.14

1. Create a macro that will calculate the area of a circle with a given radius.

#define AREA(r) PI\*r\*r

1. Define integer variable with a value of 2.

int i = 2;

1. Use the macro in the following ways:

cout << AREA(i) << endl;

cout << AREA(i+1) << endl;

cout << AREA(i++) << endl;

cout << AREA(++i) << endl;

1. Are the values what they are expected to be? What happened with i between the macro calls?
2. Try to make the macro better. What side effect can’t be avoided?

# Loops

1. Make a program that finds all divisors of the given number
2. Make a triangle on the screen, like this:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

The user should be able to set the size of the triangle, or make a size as a defined constant.

1. Make a square on the screen, like this:

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

Try to make two versions of the program, using different type of loop.

# Program Flow Control – The Guess Game

Program behavior can be controlled using branching and looping mechanisms. In the following exercises you will familiarize yourself with those mechanisms by writing a simple Guess Game. In this game program will choose a number in a range between 0 and 100 by random. The user must guess the number in a maximum of 10 tries. After each try the program will inform the player if his guess is to low or to high.

1. Use the following tip to generate random number in range 0 – 100

#include <cstdlib>

#include <ctime>

srand (time(NULL));

unsigned short random = rand() % 101;

# Functions in C++

In this exercise you will create several functions that will familiarize you with how functions work in C++ (sending by value or by reference, name overloading and default values for parameters).

# Sending Parameters

1. Create a function that is receiving its parameter by value. The function should then display all the properties (value, address and size in bytes) of the locally copied variable, increase it by one and once again display its properties.

void value(int i)

{

cout << "int i: " << i << " | " << &i << " | " << sizeof(i) << endl;

++i;

cout << "int i: " << i << " | " << &i << " | " << sizeof(i) << endl;

}

1. Use it in main():

int main()

{

int param;

value(param);

return 0;

}

1. Display all the properties (value, address and size in bytes) of param before and after it has been sent to the function.
2. Repeat this exercise with the function that is receiving its parameter by reference.

void reference(int &i);

# Other Concepts

1. Write a function void change(int &, int &); that switches the value of received arguments. Try not to use additional memory when doing so.
2. Write a function that receives up to three arguments and returns the maximum value received (zero if no arguments were given):

typedef unsigned int u32;

u32 maximum(const u32& = 0, const u32& = 0, const u32& = 0);

Term: Function Family – multiple overloaded functions that share the same name and perform similar tasks for different arguments they receive.

1. Write a function family named int sizeBits() that can receive one of the following argument type: char, short, int or double. The function shall return the size of received data type in bits. When using the function make sure what data type is send as argument.
2. Write a function which returns the index of biggest element in given array, then use it to make a function which sort the numbers in decreasing order.

# Compound Data Types

Arrays, Structures and Unions are used for better code and structure organization. In the following exercise you will familiarize yourself with arrays and structures.

# Arrays

The following exercise will familiarize you with the arrays. In the next exercises you will also combine this knowledge with structures.

1. Define an array of 10 integers.

int numbers[10];

1. The program should ask the user to fill the array with values.

for(int i = 0; i < 10; ++i)

{

cin >> numbers[i];

}

1. Create a function that will print the given array of integers to the screen.

void printArray(int array[], const usigned int &size)

{

for (unsigned int i = 0; i < size; ++i)

{

cout << array[i] << endl;

}

}

1. Create a function that will print all the values from the given array of integers greater than 5 and the number of such variables.
2. Write a function that will search the given array for the given value. The function should return the index of the first occurrence of the searched value or -1 if there is no such value in the array.

int search(int array[], const unsigned int &size, const int &value);

1. Write a function that will return the index of the greatest element of a given array of integers (index of the first occurrence if there is more than one such element). Then use it to make a function which sort the numbers in decreasing order.

# Structures

This exercise will show you how the structures work.

1. Define a structure Date that will hold the day, month and a year.

typedef unsigned int DAY;

typedef unsigned int MONTH;

typedef unsigned int YEAR;

struct Date

{

DAY day;

MONTH month;

YEAR year;

};

1. Create a function that will set the fields for a given Date.

void setDate(Date &date,

const DAY &d,

const MONTH &m,

const YEAR &y)

{

date.day = d;

date.month = m;

date.year = y;

}

1. Write a function that will return the given Date as a string. The function sprintf is used to change and combine integers into string.

#include <cstdio>

string dateToString(const Date &date)

{

char str[11];

sprintf(str, "%d-%d-%d", date.day, date.month, date.year);

return string(str);

}

1. Create a structure Person with the following fields: string name, string surname, unsigned int ID, Date birth.

struct Person

{

string name, surname;

unsigned int ID;

Date birth;

};

1. Create a function that will display a given Person.

void displayPerson(const Person &person)

{

cout << "Name: " << person.name << endl;

cout << "Surname: " << person.surname << endl;

cout << "ID: " << person.ID << endl;

cout << "Date of birth: " << dateToString(person.birth) << endl;

}

1. Create a function that will set the fields for a given Person (use the setDate() function as an example)
2. Create a function that can compare two dates. The function should return true if the second given Date is a later than the first one.

bool dateOrder(const Date&, const Date&);

1. Write a function that can take the array of Persons and returns the oldest one.
2. Define an array of a few persons and test your function.

# Dynamic Memory Allocation

In this exercise you will get familiar with the operators used in dynamic memory allocation in C++ (new, new[], delete and delete[]).

# Allocating Arrays

1. Write a program that asks the user for a number (unsigned int size).
2. Allocate the memory for an array of [size] integers.

int \*p = new int[size];

1. Set the values in the array pointed by p.
2. Write a function that returns the average value of variables in a dynamically allocated array. The function frees the memory allocated for the given array.

double average(int array[], const unsigned int &size)

{

double sum = 0;

for (unsigned int i = 0; i < size; ++i)

{

sum += array[i];

}

delete[] array;

return sum / size;

}

1. Try to access the memory after the function average() was called. What happened?

# Memory and Structures

1. Define the following structures:

struct Circle

{

double r;

};

struct Rectangle

{

double x;

double y;

};

1. Write a function family create() that can take one or two arguments of type double. The function shall allocate memory for either Circle (when one argument used) or Rectangle (when two arguments used) and return a pointer to the created figure.

Circle\* create(const double&);

Rectangle\* create(const double&, const double&);

1. Write a function family show() that can take a pointer to either Circle or Rectangle. The function shall display the figure type and its parameters to the cout.
2. Remember to free the memory once it is not needed using operator delete.

# Memory Allocation Error Handling

Method presented below is often used by former C programmers. It will not stop the code from crushing the application! Try the code, see the presentation and fix the error.

for (int i = 1; i < 10000; ++i)

{

double\* p = new double[128000];

cout << "Allocated: " << i << "MB" << endl;

if (p == NULL)

{

cout << "Memory allocation failed!" << endl;

}

}

# Database – Singly-Linked List

In this exercise you will combine the knowledge gained during the course by creating a simple database in a form of a Singly-linked List.

The idea behind a singly-linked list is simple:

* Each record in the list is called **element** or **node**
* First element in the list is called **head** of the list (the last one is called **tail**)
* Each element contains the address of the **next** element, the remaining fields are known as **data** or payload fields

NULL

next

next

TAIL

element

element

next

HEAD

element

1. Create a structure Person with the following fields: string name, string surname, unsigned int ID, Person \*next.

struct Person

{

string name;

string surname;

unsigned int ID;

Person \*next;

};

1. Create a function newPerson that will allocate the memory for a new Person, fill its fields and returns its address.

Person\* newPerson(const string &n, const string &s, const unsigned int &id)

{

Person \*p = new Person;

p->name = n;

p->surname = s;

p->ID = id;

p->next = NULL;

return p;

}

1. Create a function that will display a given Person.
2. In main define a pointer of the type Person (head of the list), that will be the start of the list.

Person \*head = NULL;

1. Create a function that will add a given Person to the end of a list.
2. Create a function that will return the number of elements in the given list.
3. Create a function that will display the whole list.
4. Create a function that will return a pointer to the Person with the given index from the list.
5. Create a function that will remove a Person with the given index from the list.

Create a function that will clear the list