C++ Programming Tutorials

Introduction

TODAY'S laboratory is to get you familiar some of the basic features of C++. You should develop the programmes on your Raspberry Pi Zero W using MobaXterm[1] to connect to your Pi over the USB RNDIS-/Gadget interface and use the Geany IDE to edit, compile and execute programmes in a Unix/Linux environment.

Please note that there are errors present in these examples so that even if you cut and paste them from this document you will need to correct the errors before the program will compile and run correctly. This type of programming exercise is there to improve your understanding of the C++ Syntax and object oriented programming concepts.

Other sections just contain code fragments that you should include in a program you develop by yourself.

[1] MobaXterm uses **SSH** as a possible way of communication with your Raspberry Pi. Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line, login, and remote command execution, but any network service can be secured with SSH.

SSH provides a secure channel over an unsecured network by using a client–server architecture, connecting an SSH client application with an SSH server. The protocol specification distinguishes between two major versions, referred to as SSH-1 and SSH-2. The standard TCP port for SSH is 22. SSH is generally used to access Unix-like operating systems, but it can also be used on Microsoft Windows. Windows 10 uses OpenSSH as its default SSH client and SSH server.

SSH was designed as a replacement for Telnet and for unsecured remote shell protocols such as the Berkeley rsh and the related rlogin and rexec protocols. Those protocols send sensitive information, notably passwords, in plaintext, rendering them susceptible to interception and disclosure using packet analysis. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet.

- From Wikipedia

Program - Out of Range

Note in this example you will need to add the main function

```
vector<int> v(10); // a vector of 10 ints,
           // each initialized to the default value, 0,
           // referred to as v[0] .. v[9]
for (int i = 0; i<v.size(); ++i) v[i] = i;  // set values
for (int i = 0; i<=10; ++i)     // print 10 values (???)

cout << "v[" << i << "] == " << v[i] << endl;
```

Program - Call by Reference

```
// call-by-reference (pass a reference to the argument)
int f(int& a) { a = a+1; return a }
int main()
```

```
{
  int xx = 0;
  cout << f(xx) << '\n'; // writes 1
               // f() changed the value of xx
  cout << xx << '\n';  // writes 1
  int yy = 7;
  cout << f(yy) << '\n';  // writes 8
               // f() changes the value of yy
  cout << yy << '\n';  // writes 8
}
```

Program – Call by Value/Reference/Const Reference

```
void f(int a, int& r, const int& cr) { ++a; ++r; ++cr; } //error:
    cr is const
void g(int a, int& r, const int& cr) { ++a; ++r; int x = cr; ++x;
    } // ok

int main(){
  int x = 0;
  int y = 0;
  int z = 0;
  g(x,y,z);     // x==0; y==1; z==0
  g(1,2,3);     //error: reference argument r needs a variable to
      refer to
  g(1,y,3);     // ok: since cr is const we can pass "a temporary"
}
// const references are very useful for passing large objects
```

Program - 2D Array

Write user defined functions for square matrix[2] to calculate

- Left diagonal sum
- Right diagonal sum

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{1}$$

[2] In mathematics, a **matrix** (plural matrices) is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object
-From Wikipedia

Program - Triangular Matrix

Write a user defined functions named `upper()` and `lower()` which takes a two-dimensional array A, with size n rows and m columns

as argument and prints the upper half and lower half of the array. For example;

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix} \tag{2}$$

Examples

These examples will focus on the object oriented paradigm[3] of the C++ programming language.

[3] Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of this or self). In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

Many of the most widely used programming languages (such as C++, Java, Python, etc.) are multi-paradigm and they support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include: Java, C++, C-Sharp, Python, R, PHP, Visual Basic.NET, JavaScript, Ruby, Perl, SIMSCRIPT, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Smalltalk. *-From Wikipedia*

Classes - Date

You will need to create the main function from the last three lines and include other necessary part for the code to properly compile and execute. Comment out the lines that don't compile.

```
// simple Date (control access)
class Date {
  int y,m,d;        // year, month, day
public:
  Date(int y, int m, int d); // constructor: check for valid date
      and initialize

  // access functions:
  void add_day(int n);  // increase the Date by n days
  int month() { return m; }
  int day() { return d; }
  int year() { return y; }
};
// ...
Date my_birthday {1950, 12, 30};   // ok
cout << my_birthday.month() << endl;  // we can read
my_birthday.m = 14;          // error: Date::m is private
```

Classes - Rectangle

You will need to fix the code to properly compile and execute. Comment out the lines that don't compile.[4]

```
// classes example
#include <iostream>
class Rectangle {
  int width, height;
public:
  void set_values (int,int);
  int area() {return width*height;}
};
```

[4] public, private and protected are access modifiers and keywords which are used in a class.

Anything declared in public can be used by any object within the class or outside the class,variables in private can only be used by the objects within the class and could not be changed through direct access(as it can change through functions like friend function).

Anything defined under protected section can be used by the class and their just derived class. - *From Stack overflow*

```
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: "<< rect.area();
return 0;
}
```

Classes - Two Rectangles

You will need to fix the code to properly compile and execute. Comment out the lines that don't compile.

```
// example: one class, two objects
#include <iostream>

class Rectangle {
  int width, height;
public:
  void set_values (int,int);
  int area () {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: "<< rect.area() << endl;
  cout << "rectb area: "<< rectb.area() << endl;
  return 0;
}
```

Program - Class Constructors

What would happen in the previous example if we called the member function area before having called `set_values`?

An undetermined result, since the members `width` and `height` had never been assigned a value.In order to avoid that, a class can include

a special function called its constructor, which is automatically called whenever a new object of this class is created, allowing the class to initialize member variables or allocate storage.

This constructor function is declared just like a regular member function, but with a name that matches the class name and without any return type; not even `void`[5]. The previous `Rectangle class` can easily be improved by implementing a constructor:

```
// example: class constructor
#include <iostream>

class Rectangle {
  int width, height;
public:
  Rectangle (int,int);
  int area () {return(width*height);}
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: "<< rect.area() << endl;
  cout << "rectb area: "<< rectb.area() << endl;
  return 0;
}
```

[5] When used as a function return type, the void keyword specifies that the function doesn't return a value. When used for a function's parameter list, void specifies that the function takes no parameters. When used in the declaration of a pointer, void specifies that the pointer is "universal."

If a pointer's type is void*, the pointer can point to any variable that's not declared with the const or volatile keyword. A void* pointer can't be dereferenced unless it's cast to another type. A void* pointer can be converted into any other type of data pointer.

In C++, a void pointer can point to a free function (a function that's not a member of a class), or to a static member function, but not to a non-static member function.

You can't declare a variable of type void.

-From Microsoft Documents

Program - Overloaded Constructors

Like any other function, a constructor can also be overloaded with different versions taking different parameters: with a different number of parameters and/or parameters of different types. The compiler will automatically call the one whose parameters match the arguments:

```
// overloading class constructors
#include <iostream>

class Rectangle {
  int width, height;
public:
  Rectangle ();
  Rectangle (int,int);
  int area (void) {return(width*height);}
};
```

```
Rectangle::Rectangle () {
  width = 5;
  height = 5;
}

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb;
  cout << "rect area: "<< rect.area() << endl;
  cout << "rectb area: "<< rectb.area() << endl;
  return 0;
  }
```

Uniform Initialisation

The way of calling constructors by enclosing their arguments in parentheses, as shown above, is known as functional form. But constructors can also be called with other syntaxes. First, constructors with a single parameter can be called using the variable initialization syntax (an equal sign followed by the argument):

```
class_name object_name = initialization_value;
```

More recently, C++ introduced the possibility of constructors to be called using uniform initialization, which essentially is the same as the functional form, but using curly braces instead of parentheses (()):

```
class_name object_name { value, value, value, ... }
```

Optionally, this last syntax can include an equal sign before the braces. Here is an example with four ways to construct objects of a class whose constructor takes a single parameter:

```
// classes and uniform initialization
#include <iostream>

class Circle {
  double radius;
public:
  Circle(double r) { radius = r; }
  double circum() {return 2*radius*3.14159265;}
};

int main () {
```

```
  Circle foo (10.0); // functional form
  Circle bar = 20.0; // assignment init.
  Circle baz {30.0}; // uniform init.
  Circle qux = {40.0}; // POD-like

  cout << "foo's circumference: "<< foo.circum() << '\n';
  return 0;
}
```

Program - Pointers to Classes

Objects can also be pointed to by pointers[6]: Once declared, a class becomes a valid type, so it can be used as the type pointed to by a pointer. For example:

```
Rectangle * prect;
```

is a pointer to an object of class Rectangle. Similarly as with plain data structures, the members of an object can be accessed directly from a pointer by using the arrow operator (->). Here is an example with some possible combinations:

```
// pointer to classes example
#include <iostream>

class Rectangle {
  int width, height;
public:
  Rectangle(int x, int y) : width(x), height(y) {}
  int area(void) { return width * height; }
};

int main() {
  Rectangle obj (3, 4);
  Rectangle * foo, * bar, * baz;
  foo = &obj;
  bar = new Rectangle (5, 6);
  baz = new Rectangle[2] { {2,5}, {3,6} };
  cout << "obj's area: "<< obj.area() << '\n';
  cout << "*foo's area: "<< foo->area() << '\n';
  cout << "*bar's area: "<< bar->area() << '\n';
  cout << "baz[0]'s area:"<< baz[0].area() << '\n';
  cout << "baz[1]'s area:"<< baz[1].area() << '\n';
  delete bar;
  delete[] baz;
  return 0;
}
```

[6] In computer science, a pointer is an object in many programming languages that stores a memory address. This can be that of another value located in computer memory, or in some cases, that of memory-mapped computer hardware. A pointer references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer. As an analogy, a page number in a book's index could be considered a pointer to the corresponding page; dereferencing such a pointer would be done by flipping to the page with the given page number and reading the text found on that page. The actual format and content of a pointer variable is dependent on the underlying computer architecture.
-From Wikipedia

Date Example - I

Using the declarations presented in last week's lecture create a class called Date create the `is_valid` member function to check whether the date is valid when a `Date` object is created by calling it from the constructor.

Test it with various valid and invalid dates. Remember that a leap year is a multiple of 4 with the exception of centennial years not divisible by 400 (i.e. 1900 didn't have a leap year but 2000 did)

Date Example - II

Create the member function `add_day` which increases the date of the `Date` object by the number of days passed as the parameter to the `add_day` routine.

Date Example - III

Write a class having two private variables and one member function which will return the area of the rectangle.

Date Example - IV

Write a program and input two integers in main and pass them to default constructor of the class. Show the result of the addition of two numbers.

Date Example - V

Create a C++ program to read time in seconds and convert in time format (HH:MM:SS) using class programming

Date Example - VI

Create a C++ program to read time in HH:MM:SS format and convert into total seconds using class programming

Inheritance - I

Write a C++ program to add two numbers using single inheritance. Accept these two numbers from the user in the base class and display the sum of these two numbers in derived class.

Inheritance - II

Write a C++ program to design a base class `Person` (name, address, phone_no). Derive a class `Employee` (eno, ename) from `Person`. De-

rive a class `Manager`(designation, department name, basic-salary) from `Employee`. Write a menu driven program to:

- Accept all details of 'n' managers.
- Display manager having highest salary.

Inheritance - III

Write a C++ program to define a base class `Item` (item-no, name, price). Derive a class `Discounted_Item` (discount-percent). A customer purchases 'n' items. Display the item-wise bill and total amount using appropriate format.

Inheritance - IV

Write class declarations and member function definitions for a C++ base class to represent an `Employee` (emp-code, name).

Derive two classes as `Fulltime` (daily rate, number of days, salary) and `Parttime` (number of working hours, hourly rate, salary).

Write a menu driven program to:

- Accept the details of 'n' employees.
- Display the details of 'n' employees.
- Search a given Employee by emp-code.

Inheritance - V

In a bank, different customers have savings account. Some customers may have taken a loan from the bank. So the bank always maintains information about bank depositors and borrowers.

Design a Base class Customer (name, phone-number). Derive a class Depositor(accno, balance) from Customer. Again, derive a class Borrower (loan-no, loan-amt) from Depositor.

Write necessary member functions to read and display the details of 'n' customers.

This document has been based on the materials "*Laboratory 3 - More C++ Exercises, Laboratory 4 - C++ Classes, Laboratory 5 - Inheritance*" by Prof. Jeremy Smith.