CSC2040 Data Structures, Algorithms and Programming Languages

Practical 3

Wednesday 18th October 2017

In this practical we will introduce the C++ class. In C++, typically, a class is declared in a Header file, and the class's member functions are implement in a Source file. This lets you share the class interface in other .cpp files whilst keeping the implementation separated.

Program 1

In this part of the practical we will look at the `Person` class introduced on Page 81 of the textbook. Download `Person.h` and `Person.cpp` from Queens Online. Study them closely and answer the following questions.

In `Person.h` there are two constructors. The first of these has a syntax:

```
Person(std::string first, std::string family, std::string ID, int birth) :
  given_name(first), family_name(family), ID_number(ID), birth_year(birth) { }
```

What does this do?

Make a new object of class Person and initialize the data.

What is another way to write this constructor to achieve the same effect?

```
given_name = first;
family_name = family;
ID_number = ID;
birth_year = birth;
```

Write three C++ statements one declaring an object of this class that is initialised by the above parameterised constructor, the 2nd applying the `get_birth_year()` member function to this object, and the last outputting the object.

```
Person* per = new Person("Dewei", "Liu", "001", 1996);
cout << "Birth year : " << per->get_birth_year() << endl;
cout << "This person : " << *per;
```

Write three C++ statements one declaring an object of this class that is initialised by the parameterless (i.e. default) constructor, the 2nd applying the `set_family_name(std::string family)` member function to this object with the parameter `"Smith"`, and the last outputting the object.

```
Person* per2=new Person();
per2->set_given_name("Abc");
cout << "This person : " << *per2;
```

Allocate an object of this class using **new** that is initialised by the parameterised constructor, then output the object's ID number by applying the `get_ID_number()` member function to the object, and finally delete this object.

```
cout << "ID : " << per1->get_ID_number() << endl;
```

Allocate an object of this class using **new** that is initialised by the default constructor, apply the member function `set_birth_year(int birth)` with `birth` being a suitable number to this object, and then output the object's birth year by applying the `get_birth_year()` member function. Finally delete this object.

Allocate an array of 385 objects of this class using **new,** apply the `set_given_name(std::string given)` member function to the 13th object with `given = "John"`, and then output this object's given name by applying the `get_given_name()` function. Finally delete all the objects.

```
Person* person_list = new Person[100];
person_list[13].set_given_name("John");
cout << "This 13th name is " << person_list[13].get_given_name() << endl;
delete[]person_list;
```

In the above allocation of an array of objects, which one of the two constructors is used?

Parameterless constructor

The following is an accessor function which is implemented in Person.h:

```
 std::string get_given_name() const { return
given_name; }
```

What is the word that describes this definition of a function?

What does the `const` mean in the above function declaration?

This function does not modify anything.

In this setter function

```
void set_given_name(const std::string& given)
{      given_name = given;
  }
```

How would you describe the method of passing the parameter `given`?

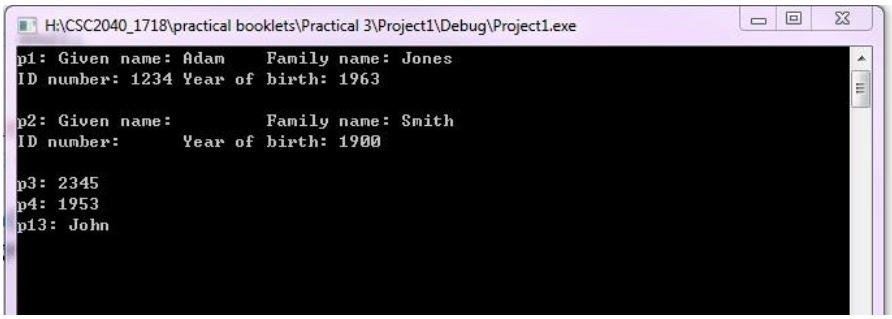
Can we change the value of the parameter `given` as below? Why?

```
void set_given_name(const std::string& given) const
{ given_name = given; given = "Unknown;    }
```

Program 2

Test the correctness of your paper work above by including all the required code into a real C++ program, and executing it. To do this, create an empty Visual Studio C++ project, and add the supplied `Person.h` and `Person.cpp` as Existing Item. Then you write a `main` function for the testing, which #include `Person.h`, and includes all the code as required above.

Save, build and execute this program and examine the output. Your output could look like the following:



Hint: Remember '\n' means take a new line (as endl). What is the symbol for a tab?

Program 3

Given two 1-D vectors (or arrays), `x`, `y`, of the same length `len`, their similarity can be measured using different methods. Two of the most common methods are: Euclidean Distance and Cosine Similarity. These are defined as follows:

Euclidean distance:

$$d(x,y) = \sqrt{(x[0]-y[0])^2+(x[1]-y[1])^2+ \ \dots \ +(x[len-1]-y[len-1])^2}$$

Cosine similarity:

$$c(x,y) = p(x,y) / (\sqrt{x[0]^2+x[1]^2+\ldots+x[len-1]^2}\sqrt{y[0]^2+y[1]^2+\ldots+y[len-1]^2})$$

where $p(x,y)$ is the inner (or dot) product between x and y:

$$p(x,y) = x[0]y[0] + x[1]y[1] + \ldots + x[len-1]y[len-1]$$

Consider the following class named as `vecsim`, which calculates the above two similarity measures for two vectors, which are passed into an object through a parameterised constructor of the object:

```
class vecsim { public:
      vecsim (double* v1, double* v2, int v_len);

      double Euclidean();
      double Cosine();
 private:
      double* vec1, *vec2;
      int vec_len;
};
```

Implement the above class. Then, develop a Test program TestVecSim.cpp in which type the following:

```
double vector1[10] = {1.5, 3., 4.5, 6., 7.5, 9., 10.5, 12., 13.5, 15.};
double vector2[10] = {3., 4.5, 6., 7.5, 9., 10.5, 12., 13.5, 15., 16.5};
 vecsim vs(vector1, vector2,
10);
 cout << "Euclidean dis = " << vs.Euclidean() <<
endl; cout << "Cosine sim = " << vs.Cosine() << endl;
```

Save, build and execute your program and examine the output.

Program 4

An extended `vecsim` class is shown below. The extended class includes an overloaded default (parameter-less) constructor, to allow for the declaration of an object without initialisers, and two overloaded distance functions, one for each distance type, to allow for the calculation of the similarity between two vectors which are passed into an object through the member function parameters.

```
class vecsim { public:
      vecsim(double* v1, double* v2, int v_len);
vecsim();

      double Euclidean();
      double Euclidean(double* v1, double* v2, int v_len);
      double Cosine();
      double Cosine(double* v1, double* v2, int v_len);

private:
      double* vec1, *vec2;
      int vec_len;
};
```

Extend your previous class to include the additional member functions. Then, in the Test program, following your previous testing statements, add the following:

```
vecsim vs2;
 cout << "Euclidean dis = " << vs2.Euclidean(vector1, vector2, 10) <<
endl; cout << "Cosine sim = " << vs2.Cosine(vector1, vector2, 10) << endl;
```

Save, build and execute your program and examine the output.

Practical Test 1

Practical Test 1 will take place during the practical class on Wednesday 25 October. You will be asked to develop C++ programs to solve a given set of problems, and then to use the Assignment Tool in Queens Online to upload your answers. The test is open book, but with restrictions. See the Course Information for the "Format for practical tests and final examination". Pen and paper can be used for working out / rough work.