CSC2059 Data Structures and Algorithms

Practical 3: Classes and Stack Data Structure

Wednesday 13th October 2021

Launch Visual Studio. Create a new Solution Practical3 and name the first project Class1.

Project 1. Class Person

In this project we will look at a simplified version of the Person class introduced on one of the recommended textbook. Download Person.h and Person.cpp from the Canvas, place them into Practical3\Class1 folder, and then add them into project Class1 (by clicking Header Files – Add – Existing Item… to add Person.h, and by clicking Source Files – Add – Existing Item… to add Person.cpp). Study the class closely.

(1) Answer the following questions.

In the Person class there are two constructors. What does the first constructor do?

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

Rewrite the above constructor in the following form in the text box below:

```
Person::Person(string first, string family, string ID, int birth)
{
       // body of the function
}
```

In this constructor include appropriate assignment statements to initialise the data fields of an object that will achieve the same effect as the above form.

What does the 2nd constructor do?

```
Person::Person()
        : given_name(""), family_name(""), ID_number(""), birth_year(1900)
{
}
```

What does the const mean in the following function definition?

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

What does the const mean in the following function definition?

```
string Person::get_ID_number() const
{
        return ID_number;
}
```

In the project, create a testing program test1.cpp to perform the following tests.

(2) Write three C++ statements, the first declaring an object of this class with the data fields initialised by using the parameterised constructor, the 2nd applying the get_birth_year member function to this object to output (via cout <<) the birth_year, and the last printing the entire object by applying the member function print.

(3) Create an object using **new** with the data fields initialised by using the parameter-less constructor. Then update the object by applying the appropriate member functions to set the given_name, family_name, ID_number and birth_year, respectively, and finally print the entire object by applying the print function, all via the pointer to the object.

(4) Create an object using **new** with the data fields initialised by using the parameterised constructor with the same data field values as in (2). Then compare the object created here with the object created in (2) for equality. If the two objects are equal print out (via cout) "Objects are equal". There is an operator == function in the class, showing that two Person objects are equal if they have identical ID_numbers.

Before the program finishes, free the memory spaces used for the dynamically allocated objects in the above tests.

Project 2. Build and use a simple class (modified from the 2019/20 examination)

This exercise is for you to understand how to build a simple C++ class, and how to use the class to perform operations.

In Solution Practical3 create a new project Class2. Inside the project create a C++ class named xClass. The class has the following structure of data fields and member functions:

```
class xClass {
private:
	int* data;		// private, (pointer to) an integer data array
	int size;		// private, size of the data array

public:
	// (1) constructor, to set size to the given length, allocate memory
	// for the data array to the given size, and set all elements of the
	// array to random numbers between 0 and max_val-1 inclusive
	xClass(int length, int max_val);
	// (2) destructor, to free the memory allocated for the data array
	~xClass();

	// (3) function, to print all numbers in the data array;
	void print_data() const;
	// (4) function, to return the average of the numbers in the data array
	int ave_data() const;
	// (5) function, to return the number at index i of the data array
	int data_at(int i) const;
	// (6) function, to return the count of the numbers in the data array that fall
	// within the range min-max inclusive
	int range_data(int min, int max) const;

	// (7) copy constructor, used to initialise one object by another
	// e.g. xClass a = b, where object a is initialised by object b
	xClass(const xClass& b);

	// (8) operator +=, applied to two objects e.g. a += b
	// the resultant a will have a lengthened data array, holding the original
	// a's data followed by the b's data
	void operator+=(const xClass& b);
};
```

(a) First, create a header file xClass.h for the class, which includes the declaration of the class, as above.

(b) Second, create a source file xClass.cpp for the class, which includes a suitable implementation of all the member functions of the class, as defined above. These functions include: (1) the constructor, (2) the destructor, (3) the print_data function, (4) the ave_data function, (5) the data_at function, (6) the range_data function, (7) the copy constructor, and (8) the operator+= function.

(c) Next, create a source file test2.cpp to include the main function with suitable code to test your programs. For example, the following can be a test main function. Run this test program. Make sure that (1) you understand the operation of each statement, and (2) the output of each operation meets your expectation.

```
int main()
{
   xClass x(10, 100);
   x.print_data();

   cout << x.ave_data() << endl;
```

```
        cout << x.data_at(5) << endl;
        cout << x.range_data(40, 50) << endl;

        xClass y = x;
        y.print_data();

        x += y;
        x.print_data();

        return 0;
    }
```

(d) Finally, rename the above main to main1, and create a new testing main function to perform the similar tests to those in main1 by creating dynamic objects and by using pointers to the objects.

Project 3. iStack class expansion

In this project, we will add new member functions into the iStack class to expand this class.

In Solution Practical3 create a new project iStack1. Download iStackNode.h, iStackNode.cpp, iStack.h, and iStack.cpp and add these into the project iStack1 (see Project 1 to remind you how to add existing classes into a project).

(1) Inside the iStack class, add a new public function

```
void print();
```

which prints the contents of the stack, from the top to the bottom, but LEAVING THE STACK AS IT WAS.

(2) Inside the iStack class, add a new public function

```
int search(int key);
```

This function should return the position of the first occurrence of item 'key' in the stack, starting from the top of the stack. It should not change the stack in any way. The bottom position is 0. If the 'key' doesn't appear in the stack, you should return -1.

(3) In the project create a source file test3.cpp, in which perform the following operations to test the expanded iStack class:

- Declare an iStack object is
- Push 100 random numbers between 0 – 9 onto the stack
- Print the contents of the stack using the new function print
- Search the 1st occurrence of 5 (may be included in the stack) and 20 (not included in the stack), respectively, using the new function search.

(4) Repeat the above tests by using a dynamically created object of the iStack class.

Project 4. iStack class applications

In this project, we will use the iStack class public functions to build new functions from outside the class.

In Solution Practical3 create a new project iStack2; add iStackNode.h, iStackNode.cpp, iStack.h, and iStack.cpp into this project. Then create a source file test4.cpp for the following development.

(1) A stack can be used to reverse the order of a sequence of items. In test4.cpp write a main function, in which you use an iStack stack to reverse the order of a sequence of integer numbers input from the keyboard. Print out the message "Enter sequence of numbers, ending with *" to the user. When the sequence is ended (a `*' is read), your program should output the sequence of the numbers in the reverse order as they were input.

Hint – you may use a string variable to take the input and use function stoi to convert a string to an integer number.

(2) In test4.cpp, write a function

```
bool remove(iStack& is, int n);
```

which takes an iStack stack and an item number n, and removes item n from the bottom (n = 0) of the stack. Remember: you should not edit the supplied iStack classes! And don't be tempted to use an array. If there is not an item n, then output false or otherwise output true. Rename the above main to main1 and write a new main function below the remove function, in which create a stack and use it to test the effect of your remove function by using the print function of the iStack class.

If you have any questions, ask a demonstrator or a teaching staff face to face in the labs, or online by using MS Teams, during the timetabled practical session time. At any other time outside the timetabled session time, you can send in your questions by using the Ticketing System, or by using emails.