CSC2059 Data Structures and Algorithms

Practical 1: Introduction to C++

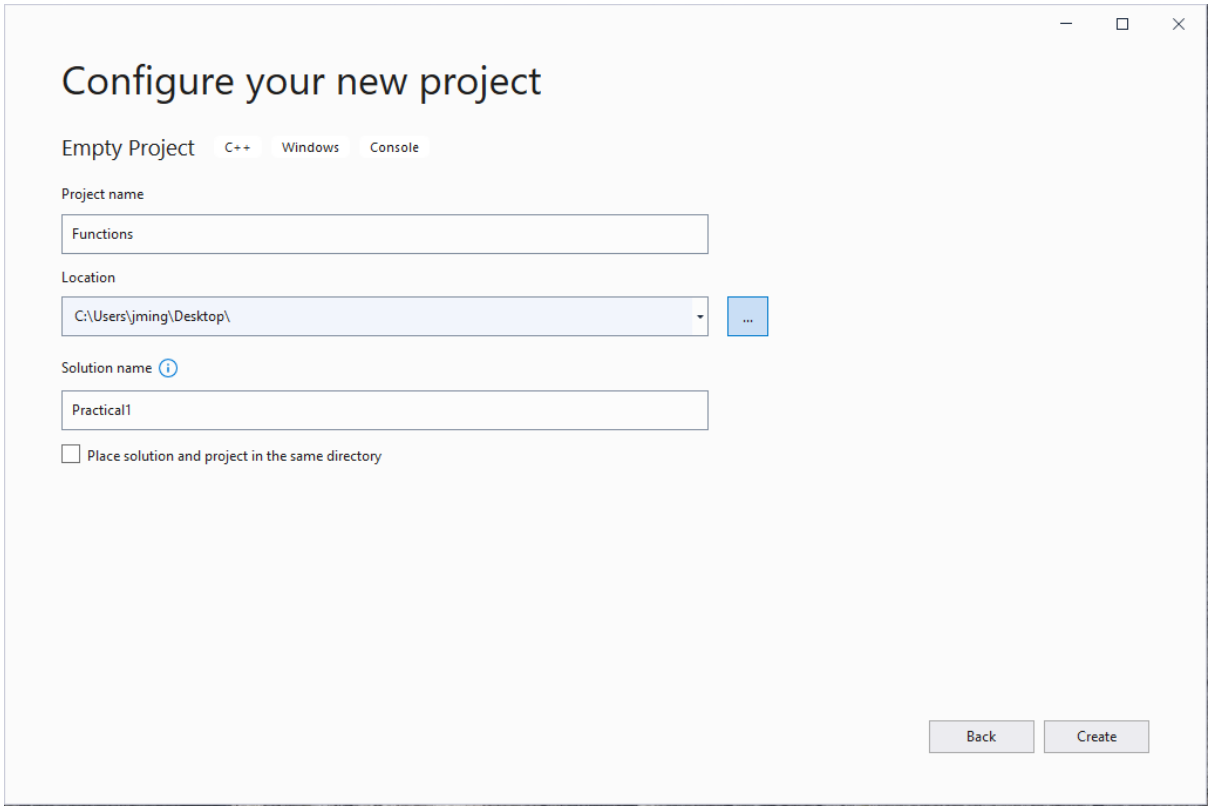
Wednesday 29th September 2021

Getting started with Microsoft Visual Studio.

Double click on the Microsoft Visual Studio icon on your Desktop.

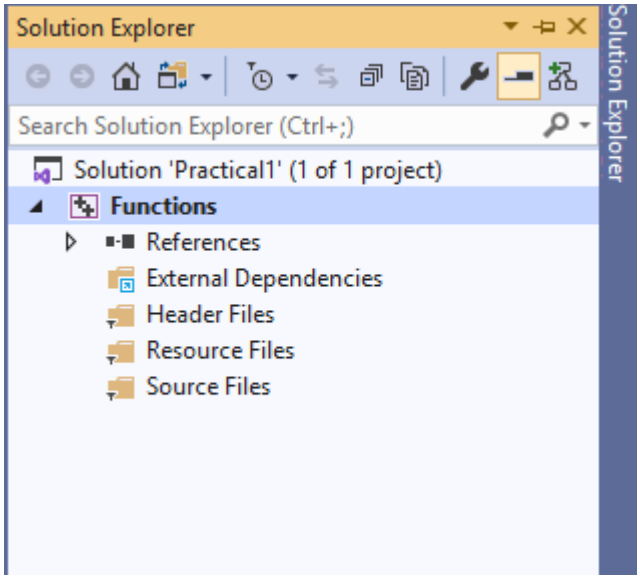
Select **Create a new project->Language->C++->Empty Project.** Click **Next.**

This takes you to the **Configure your new project** page. Use Browse to choose a proper Location to store your program, e.g. *your* Desktop (C:\Users\YourUsername\Desktop). Give your Solution a name e.g. Practical1, and your first project in the Solution a name e.g. Functions.



Click **Create**.

You will be presented with a Solution Explorer window as follows (if it is not visible choose **View->Solution Explorer**):

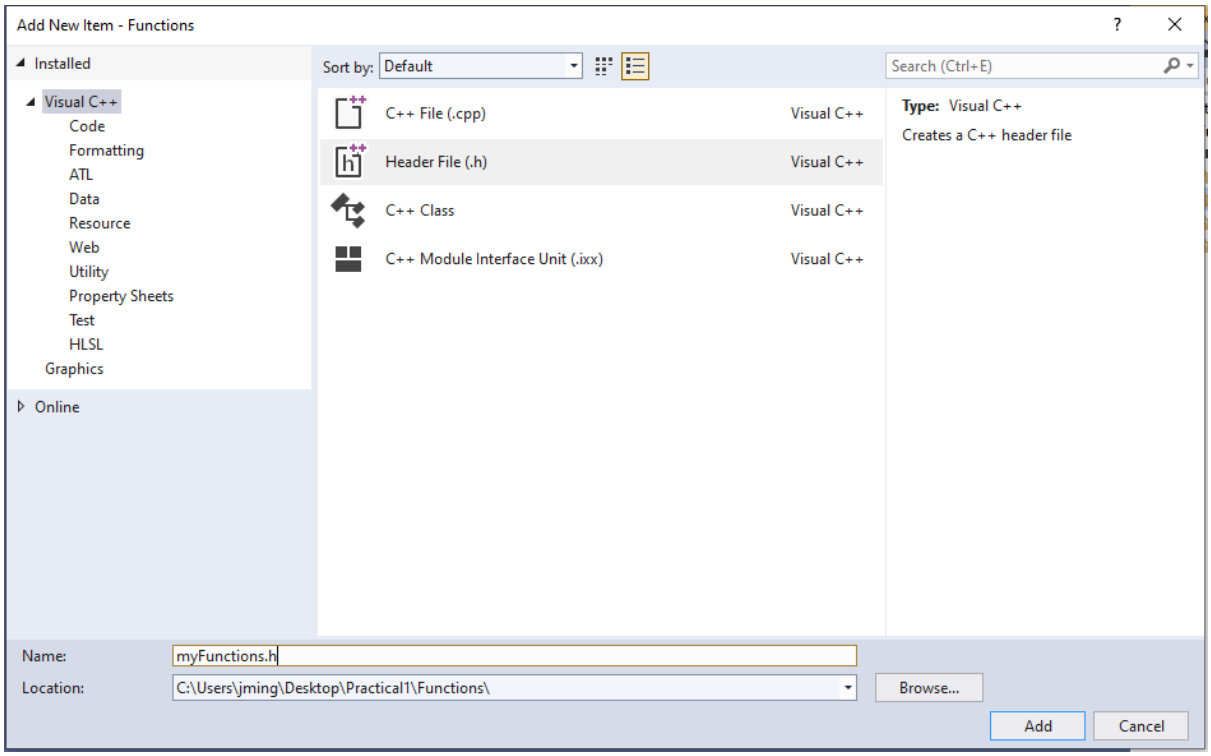


Project 1. Build library myFunctions

In project 1, you will create a simple library, named myFunctions, inside the project Functions you have just created. You write the library in the typical C++ program structure which includes two parts: (1) the header file, and (2) the source file.

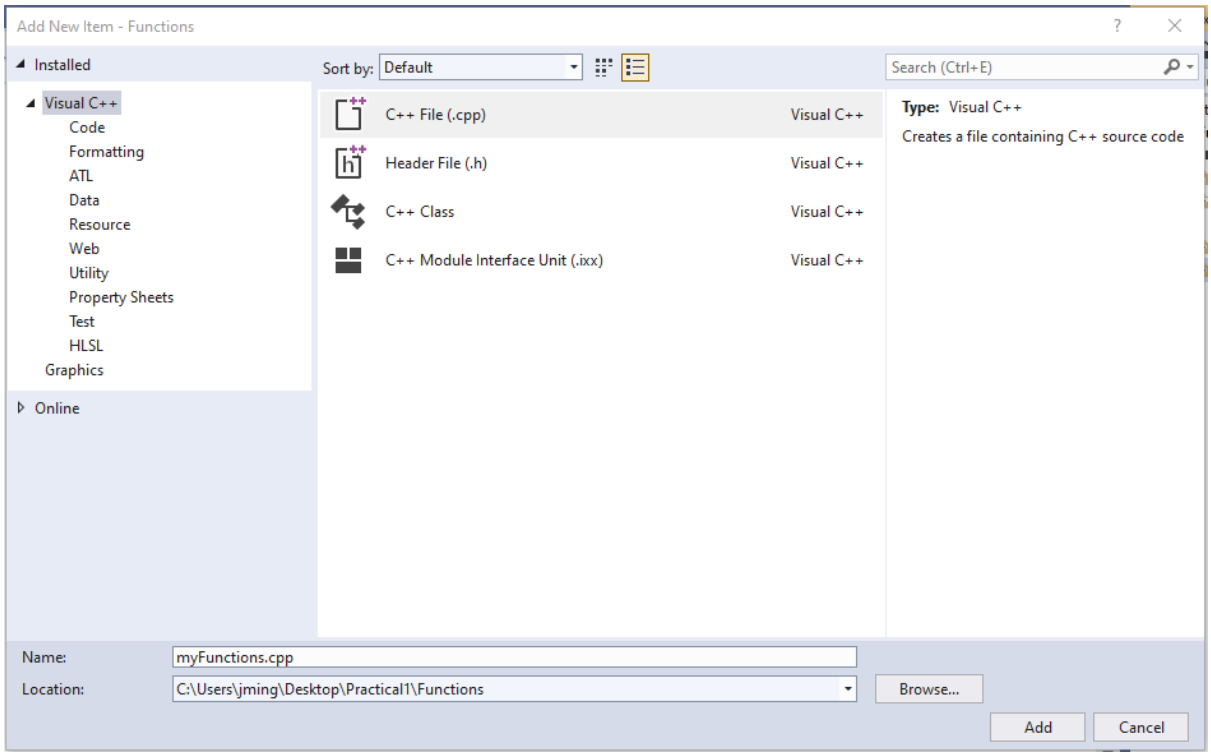
Right click on Header Files and Choose **Add->New Item**

As shown below you are creating a Header file (.h). Give it a name of the library myFunctions.h and click **Add**.



Then, right click on Source Files and Choose **Add->New Item**

As shown below you are creating a C++ file (.cpp). Give it the same name as the header file myFunctions.cpp (but use a different extension .cpp) and click **Add**.



Now, we repeat the work that we did in the lectures. First, **declare** three simple functions in the header file:

```
int iAdd(int a, int b);
double divideXbyY(double x, double y);
int divideXbyY(int x, int y);
```

Note, you must guard the above declaration by using appropriate include guards, see an example provided in the lecture notes (the need for the include guards will be explained in a later lecture).

Second, **implement** the three functions in the source file. Note, you must #include the header file myFunctions.h in the source file for implementing functions declared in the header file.

In the implementation, code defensively when appropriate. For example, the function `int divideXbyY(int x, int y)` can be written as a 'cover function' for the division operation z = x / y:

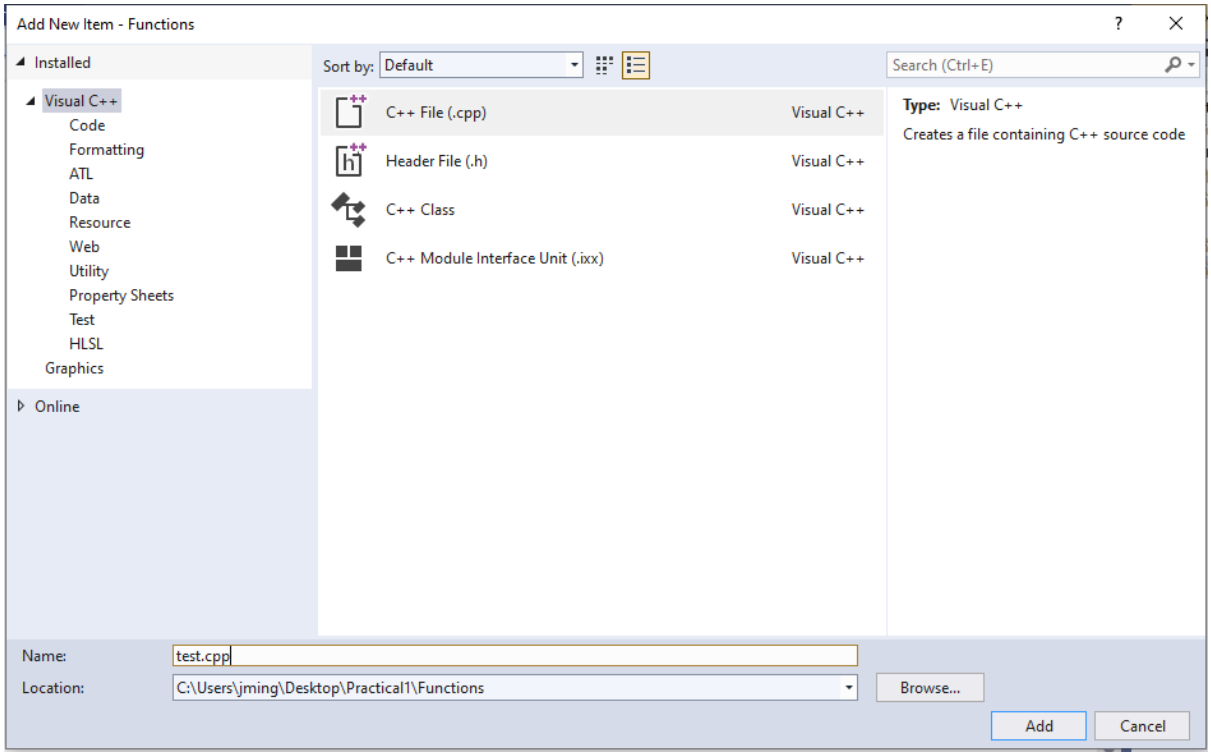
```
int divideXbyY(int x, int y)
{
	if (y == 0) {
		cout << "division by zero error" << endl;
		exit(0);
	}
	return x / y;
}
```

The cover function based operation z = `divideXbyY(x, y)` can catch the illegal condition (y = 0) while the statement z = x / y can't, or you will have to write a test like the following each time you perform a division in your code:

```
    if (y != 0)
       z = x / y;
    else {
       cout << "division by zero error" << endl;
       exit(0);
    }
```

This won't make your code clearer. You can find more about this in the lecture notes/videos.

Finally, **use** the myFunctions library built above to perform calculations. In Solution Explorer, right click on Source Files and Choose **Add->New Item,** create a C++ file (.cpp) named test.cpp, and click **Add**.



The test.cpp can be viewed as an application program, which calls the functions defined in the myFunction library. Type the following in test.cpp

```
#include "myFunctions.h" // user-defined class, included using ""
#include <iostream> // system i/o class, included using <>

using namespace std;

int main()
{
	// cout: print results to screen
	cout << iAdd(12, 13) << "\n"; // newline operator

	cout << divideXbyY(1, 2) << endl; // newline operator
	cout << divideXbyY(1.0, 2.0) << endl;
	cout << divideXbyY(16.0, 0.0) << endl;

	return 0;
}
```

To understand more about the using namespace Statement and the importance of the main function, read lecture notes or pages 4, 5 and 6 of the recommended textbook.

Save All (**File->Save All** or Ctrl + Shift + S). Compile myFunctions.cpp and test.cpp (**Build->Compile** or Ctrl + F7) to fix any syntactic errors. Then build the Solution or the Functions project (**Build->Build Solution**, or **Build->Build Functions**) to generate the executable code test.exe.

We will run the program in Debug mode, so in **Build->Configuration Manager…** set Active solution configurations to Debug, set Active solution platforms to X86, and set **Project->Functions Properties…->Linker->System->Subsystem** to **Console (/SUBSYSTEM:CONSOLE).** To run the program click **Debug->Start Without Debugging** (Ctrl F5).

Congratulations you have run your first professional C++ program.

Project 2. Expand myFunctions library

Expand the above myFunctions library by adding two new functions to calculate the base-e log(x) of argument x and square root sqrt(x) of x. Add the declarations of the two functions, below, into the header file myFunctions.h and the implementations of the two functions in the corresponding source file myFunctions.cpp.

```
double Log(double x);
double Sqrt(double x);
```

You are asked to write these two functions as **cover** functions of the direct calculations log(x) and sqrt(x), to catch the illegal conditions for the argument x. Based on mathematics, log(x) exists only for x > 0 and sqrt(x) is legal only for x >= 0.

Test the new functions in the main function in the test program test.cpp by calling them with both legal and illegal arguments x.

Project 3. Complete the exercise given in the lecture notes

Add a new function into the above user-defined function library myFunctions:

```
void sortPrint(int i, int j, int k);
```

The function takes three integer numbers from parameters, sorts these numbers in descending order, and then prints them in the sorted order on the screen. You will:

a. add the declaration (or prototype) of the function in myFunctions.h
b. add the implementation of the function in myFunctions.cpp

Test the new function in test.cpp. Rename the above main function as main1. Then write a new main function, in which you can:

a. print a message on screen to ask user to type three numbers as input
b. read the user's typed three numbers
c. use the read numbers to call sortPrint to calculate & output the result.

Project 4. Write a simple calculator

Add a new function into the library myFunctions:

```
double myCalculator(double a, char op, double b);
```

to calculate a formula in the format:

a op b

where a, b are two double-precision numbers and op is an operator which can be +, -, * or /. The function uses the given a, b and the operator (which is a char variable) to evaluate the formula and returns the result to the calling function. Rename the above main function as main2. Write a new main function to test your calculator. In the main function, you ask the user to type the formula, e.g. 3 + 2, using keyboard. Then you use the user typed numbers and operator to call the function to obtain the result of the equation. The following could be an application scenario of your program:

```
Input the formula to calculate (e.g. 3 + 2)
4 / 2
Answer = 2
```

Project 5. Loop reminder and random numbers

Loops in C++ are similar to those in Java and C#. The following shows a program to generate random numbers in C++, which you may find useful in your future programming exercises. Rename the above main function as main3. Copy the program in a new main function in your test.cpp and run it, to study the effects of R, with or without srand() etc. on the random numbers being generated.

Rewrite this test program by using the while loop, and the do…while loop, respectively, to replace the original for loop.

```
int main()
{
    // read N, R
    int N, R;
    cout << "Input N, R: the number and range of random numbers to generate" << endl;
    cin >> N >> R;

    // set the initial point for generating the random numbers using current time
    // so each run will give you a different sequence of random numbers
    // need to #include <ctime>
    srand(time(0));

    // generate N random numbers using rand() in the range [0, R-1] inclusive, and
    // calculate their sum
    int sum = 0;
    for (int n = 0; n < N; n++) {
       int rn = rand() % R;
       cout << rn << endl;
       sum += rn;
    }
    // output the average of these numbers
    cout << "average = " << sum / N << endl;

    /* implement the above loop operation by using while, and do...while loops */

    return 0;
}
```

Project 6. More loop exercises

Rename the above main function as main4. Write a new main function that allows the user 3 attempts to guess a magic number between 0-9. The magic number is generated using the C++ built-in random number generation function (see Project 5). Implement this program using the

three different loops that are available: the for loop, the do…while loop and the while loop. Allow the user three attempts to guess the magic number.

If you have any questions, ask a demonstrator or a teaching staff 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 your questions to the teaching staff by using emails.

CSC2059 Data Structures and Algorithms

Practical 2: Pointers and Dynamic Allocation

Wednesday 6th October 2021

Launch Visual Studio. Create a new Solution Practical2 and name the first project Pointer1.

Project 1. Some pointer operations

Pointers are one of the most powerful features of C++. Try to understand how to use the pointer operators * and &, and the NULL pointer, and how to program with pointers.

In project Pointer1, create a new source file test1.cpp, in which write a main function. Type the following code into the main function:

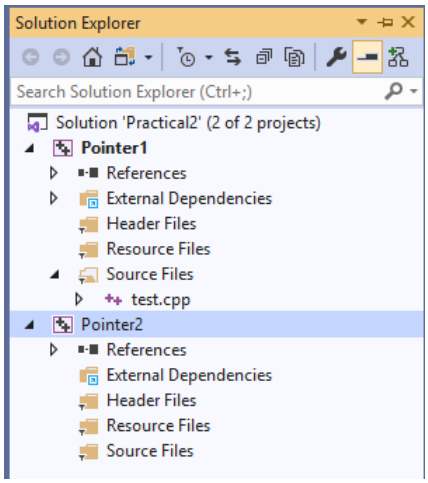
```
int i = 27;          // declare an int variable
int* pi = &i;        // declare an int pointer pi pointing to the address of i
cout << "value of i is " << i << endl;
cout << "value of pi (i.e. address of i) is " << pi << endl;
cout << "value pi is pointing to, or from dereferencing pi, is " << *pi << endl;

*pi = 35;            // change the value of i via dereferencing pi
cout << "value of i is " << i << endl;
*pi = *pi * 2;       // double the value of i via dereferencing pi
cout << "value of i is " << i << endl;

double* pd = NULL;   // declare a double pointer
                     // always initialise an unused pointer to NULL
cout << "value of pd is " << pd << endl;
cout << "value pd is pointing to is" << *pd;
                     // program crashes because you try to dereference NULL
                     // NULL is not a valid memory address
                     // comment the above line to continue the test below

// uncomment the following code
// the code tries to print the value of x, which is 10, via pointer px but is wrong.
// Correct it.
/*
int* px = NULL;
int x = 10;
*px = x;
cout << &px << endl;
*/
```

Run the program and ensure you understand the meaning of each of the operators and program statements. Answer the following questions.

- What do you notice about the value of i and *pi?
- What do you notice about the value of pi and *pi?
- What do you understand about the operation *px = x and px = &x?
- What do you understand about the value *px and &px?

Project 2. Pointer wariness

In the Solution Practical2, add a new project. Choose **File->Add->New Project-> Empty Project** and click **Next.** Set **Project name** as Pointer2 and click **Create**. Now your Solution Practical2 should have two projects as shown in the Solution Explorer on the right.



In project Pointer2, create a new source file test2.cpp, in which type the following code:

```
double* pCubeX(double x)
{
	double cube = x * x * x;
	return &cube;
}

int main()
{
	double* px = pCubeX(3.0);
	cout << *px << endl;

	return 0;
}
```

The function pCubeX calculates the cube of x and returns a pointer to the cubic value. In the main function, it is called with an appropriate value for x (3.0) and the result returned (which should be 27) is printed out.

To run the program, choose **Project** and select **Set as StartUp Project.** Is the program generating correct output? Is there a bug? Why?

Project 3. More pointer operations

In our lectures we described the implementation of the mul_div function, called by using two arguments, by using call-by-reference:

```
void mul_div(double& first, double& second)
{
	double temp = first * second;
	second = first / second;
	first = temp;
}
```

After the function finishes execution, the original calling argument for `first` will take the value of the product of the two calling arguments, i.e. first * second, and the original calling argument for `second` will take the value of the division of the two calling arguments, i.e. first / second.

The same effect can be achieved by using pointers as the parameters of the function. Write a function

```
void mul_div(double* first, double* second);
```

which will have the same effect on the two calling arguments - their values will be modified accordingly as the above after the function finishes execution.

In the Solution Practical2, add a new project Pointer3 following the same procedure as described in Project 2. In the new project, create a new source file test3.cpp, in which you first implement the

above pointer-based mul_div function, and then write a main function below this function to perform the test. Make sure you obtain the expected result.

Project 4. Search and count occurrences in a char-string array

In the Solution Practical2, add a new project Search, in which create a new source file test4.cpp. In test4.cpp, first write a function

```
int search(char* pchs, int size, char key);
```

The function takes a char-string array pointed to by pchs with size characters, and searches the occurrences of the character key in the array. It returns the count of the occurrences. You are asked to write the search algorithm by using two different methods: (1) array indexing, and (2) pointer arithmetic, to cycle through the array for the search given the pointer pchs to the array.

Next, write a main function below the search function, to take input from users for the size of the array and for the key (a-z) to be searched. In the main function, you do:

- Allocating a char array using the size provided;
- If the allocation is successful, populating the array elements with random numbers: `97 + rand() % 26;` which generate the ASCII code for lower-case alphabet characters (a-z);
- Calling the search function with the array and the key provided, to print the count found in the array.
- Deleting the array after the search operation finishes.

Project 5. Manipulate arrays in functions

In the Solution Practical2, create a new project Arrays. In the new project create a source file test5.cpp. Copy the following program into test5.cpp and run it. Study the program carefully, to learn how to write functions to manipulate pointer-led memories.

bResizeArray is a function which resizes a given int array pointed by pi from the old size (size) to a new size (sizeNew), and copy the old array numbers to the new array. The new size can be greater or smaller than the old size so the new array can be expanded (by initialising the new tail to zero) or shrunk (by erase the old tail contents). The function returns a Boolean variable, true for successful resizing or false for unsuccessful resizing. For successful resizing, the memory occupied by the old array pi is pointing to is freed; then pi is pointed to the new array, and size is updated to the new size. All these changes to the calling arguments (pi, size) are achieved by using the call-by-reference method applied to both pi and size.

```
bool bResizeArray(int*& pi, int& size, int sizeNew)
{
	// allocate new array
	int* piNew = NULL;
	if (sizeNew <= 0 || (piNew = new int[sizeNew]) == NULL)
		return false; // resize fails

	// if expanding, initialise the new tail to zero
	if (sizeNew > size) {
		for (int i = 0; i < size; i++)
			piNew[i] = pi[i];
		for (int i = size; i < sizeNew; i++)
			piNew[i] = 0;
	}

	// if shrinking, erase the tail contents
	else {
		for (int i = 0; i < sizeNew; i++)
			piNew[i] = pi[i];
	}

	// delete old array
	delete[] pi;

	// point pi to the new array, and update size to sizeNew
	pi = piNew;
	size = sizeNew;

	// resize successful
	return true;
}

int main()
{
	// old array
	int size = 10;
	int* pi = new int[size];
	for (int i = 0; i < size; i++)
		pi[i] = i;
	for (int i = 0; i < size; i++)
		cout << pi[i] << " ";
	cout << endl;

	// expanding
	if (bResizeArray(pi, size, 20)) {
		for (int i = 0; i < size; i++)
			cout << pi[i] << " ";
		cout << endl;
	}

	// shrinking
	if (bResizeArray(pi, size, 5)) {
		for (int i = 0; i < size; i++)
			cout << pi[i] << " ";
		cout << endl;
	}

	// free array memory
	delete[] pi;

	return 0;
}
```

Following the above example, write a function in test5.cpp:

```
bool bAddArrays(char*& pchs, int& size, char* pchs2, int size2);
```

The function will 'add' two arrays, the 1st pointed to by pchs with size elements, and the 2nd pointed to by pchs2 with size2 elements, by concatenating the 2nd array to the 1st array to have a size + size2 long new array, without changing the corresponding elements. After the operation finishes, pchs is pointing to the new array, size equals the new array size, and the 2nd array pointed to by pchs2 remains unchanged. The function returns true if the concatenation is successful or false if the concatenation fails.

Rename the above main function as main1. Write a new main function below bAddArrays in which design a suitable test for the function, to make sure it works as expected.

Project 6. Insert element into an array

In the Solution Practical2, create a new project Insert. In the project create a new source file test6.cpp. In test6.cpp write a function:

```
bool bInsert(int*& pi, int& size, int pos, int val);
```

This function allows you to insert a value val at position pos in an array pointed to by pi with size elements. The functions returns true when successfully inserting the value or false when the operation fails. Below are a list of instructions to help you achieve this:

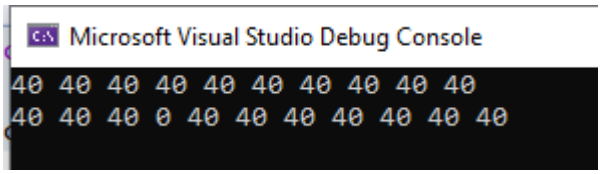
- Check the position of insertion is a legal position i.e. within the range of the array
- Increment size
- Create a new array
- Transfer everything up to pos from the old array to the new array
- Insert val into the new array at position pos
- Transfer the remainder of the old array to the new array
- Delete the old array and set the new array as the old array

Then, below the bInsert function write a main function to test if it works correctly. For example, you can use the main below to test your bInsert function:

```
int main()
{
	// the array before insertion
	int size = 0;
	int* pi = NULL;

	// insert 10 40s at the end of the array
	for (int i = 0; i < 10; i++) {
		if (!bInsert(pi, size, i, 40)) {
			cout << "Insertion fails" << endl;
			return 1;
		}
	}
	// print the array
	for (int i = 0; i < size; i++)
		cout << pi[i] << " ";
	cout << endl;
```

```
	// insert a value 0 at position 3
	if(!bInsert(pi, size, 3, 0)) {
		cout << "Insertion fails" << endl;
		return 1;
	}
	// print the array
	for (int i = 0; i < size; i++)
		cout << pi[i] << " ";
	cout << endl;

	// delete array
	delete[] pi;

	return 0;
}
```

You program should generate the following output:



Project 7. Build a formal library

The above functions were developed in separate projects for specific (test) applications. They can be placed in a formal function library and so made to be usable for a potentially wider scope of applications. In the Solution Practical2, add a new project Library. In the Library project, (1) add a new header file myLib.h, in which include the declaration of the following functions – *do not forget to include the 'include guards' in the header file* - :

```
void mul_div(double* first, double* second);
int search(char* pchs, int size, char key);
bool bResizeArray(int*& pi, int& size, int sizeNew);
bool bAddArrays(char*& pchs, int& size, char* pchs2, int size2);
bool bInsert(int*& pi, int& size, int pos, int val);
```

And (2) add a new source file myLib.cpp, in which include the implementation (i.e. code) of these functions – *do not forget to include #include "myLib.h" in the source file*.

Now you have stripped off the specific test main for each function and had a general-purpose library myLib, with two files myLib.h & myLib.cpp, which can be included (by #include "myLib.h") in any of your future programs if you need to perform an operation that is implemented by one of these functions.

For example, create a new source file test7.cpp in the project, in which you #include "myLib.h". Then write a main function in test7.cpp in which you can perform the operations as you have done in test3.cpp – test6.cpp or beyond.

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.

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.

CSC2059 Data Structures and Algorithms

Practical 4: Template Functions and Classes

Wednesday 20th October 2021

Launch Visual Studio. Create a new Solution Practical4 and name the first project Template1.

Project 1. Template functions

The following shows a C++ implementation of a function (see Project6 in Practical 2), for inserting a value val into an integer array pointed to by pi of size elements, at position pos:

```
bool bInsert(int*& pi, int& size, int pos, int val)
{
	if (pos < 0 || pos > size) {
		cout << "pos is out of range" << endl;
		return false;
	}

	// new array size after insertion
	size++;
	// new array
	int* piNew = new int[size];
	if (piNew == NULL)
		return false;

	// copt pi to piNew & insert val
	for (int i = 0; i < pos; i++)
		piNew[i] = pi[i];
	piNew[pos] = val;
	for (int i = pos + 1; i < size; i++)
		piNew[i] = pi[i - 1];

	// delete old array
	delete[] pi;
	// point pi to the new array
	pi = piNew;

	return true;
}
```

You are asked to convert the above function into a template function so that it can be used to insert an element for other arrays of suitable data types. To do this you may:

(1) In the project Template1 create a source file test1.cpp, in which write your template function bInsert.

(2) Below the function, write a main function to perform suitable tests. For example,

1. Define size= 5. Allocate a string array pointed to by pstrs with size elements. Assume initialising the array with the following string elements. Print the values in this array after the initialisation.

A `string` array with size = 5 elements

abc
def
jkl
mno
pqr

Then, call the above function with the array to insert a string: “ghi”, at position 2. Note that when you use “ghi” directly in place of val Visual Studio may give you an error message of type mismatch, as VS takes “ghi” as a constant and val as a variable. You can convert “ghi” to a variable by using an explicit type conversion (string)”ghi”, or by using string(“ghi”) instead of “ghi”. After the insertion operation print the values in the new array to verify that the function is correct.

2. Class is a data type defined by the user and can certainly be represented by the generic data type T. Download the provided Person class (Person.h, Person.cpp) and add this class into the Template1 project. Define size = 10 and allocate a Person array pointed to by pps with size person records. Print the values in this array using the member function print of this class.

   Then, call the above bInsert function with the array to insert a new Person object with first/family/ID/birth = “Jack”/“Smith”/“123456”/1990 at a suitable position (e.g. 4). After the insertion operation print the values in the new array to verify that the result is correct.

Project 2. Template class

In C++, it is possible to overrun (or underrun) an array boundary at run time without generating a runtime error message. The following shows a class that implements an int type *safe* array that provides runtime boundary checking by overloading the [ ] operator. Study the class and the attached test program carefully:

- The class uses a constructor to allocate the memory data for the array and a destructor to free the memory after the array object is destroyed;
- The class overloads the operator [ ] for accessing the individual elements of the array with boundary checking;
- The operator function returns the address (i.e. the reference) of the element that is being accessed, so that the [ ] operator can be used on both the left side and the right side of an assignment statement (see the attached main program for examples).

Safe array class & testing program

```
class sarray {
public:
	sarray(int size);
	~sarray();

	int& operator[](int i);

private:
	int size;
	int* data;
};
```

```
sarray::sarray(int size)
	: size(size)
{
	if (size > 0) data = new int[size];
	else {
		cout << "illegal array size = " << size << endl;
		exit(1);
	}
}

sarray::~sarray()
{
	delete[] data;
}

int& sarray::operator[](int i)
{
	if (i < 0 || i >= size) {
		cout << "index " << i << " is out of bounds." << endl;
		exit(1);
	}
	return data[i];
}

int main()
{
	// create a 10-element safe array
	sarray array(10);

	// in-bound access, [] is used on the left side or an assignment
	array[5] = 23;
	// in-bound access, [] is used on the right side of an operation
	cout << array[5] << endl;

	// out-of-bound accesses
	array[13] = 392;
	cout << array[-1] << endl;

	return 0;
}
```

In the above program, comment out the lines highlighted in yellow, and run the program in release mode (Solution Configurations->Release). You may see that C++ can 'tolerate' some out-of-bounds errors. Do not forget to uncomment the lines after this test.

The above class, however, only allows you to create safe arrays for integers. In this project, you are asked to implement a template class, to create a generic safe-array class that can be used for creating safe arrays for any suitable data types.

In the Solution Practical4 add a new project Template2, in which you implement the template class for safe arrays in a header file sarray.h and conduct suitable tests of the class you implement in a source file test2.cpp.

To test your new class you should create safe arrays for at least three data types: double, char, and string. You should test your code by accessing the arrays you create within bounds and out-of-bounds.

Project 3. Template Stack class

In the Solution Practical4 add a new project Template3. Download the provided template classes StackNode.h and Stack.h and add them into the new project.

(1) First, add a new operator == function inside the Stack class to perform logic comparison between two Stack objects e.g. a == b. By definition, two stacks are identical if and only if they have the same size and same sequence of data items (assuming that different data items are comparable for equality). The comparison should not change either of the stacks in any way.

Create a test3.cpp in the project, with a main function to test the operator function by creating two identical stacks a, b and two different stacks a, b, respectively, for comparison:

```
if (a == b)
	cout << "Two stacks are identical" << endl;
else
	cout << "Two stacks are different" << endl;
```

(2) Next, extend Project 3 of Practical 3 from the iStack class to the template Stack class, to add member functions print and search into the class. The tasks were detailed in (1) and (2) in Project 3, Practical 3, for the iStack class; now you will implement the same functions for the template Stack class.

(3) Finally, extend the tasks described in (3) and (4) in Project 3, Practical 3 from testing the iStack class to testing the template Stack class. You will test the new member functions for the template Stack class by using a suitable data type of your choice.

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.

CSC2059 Data Structures and Algorithms

Practical 5: Lists

Wednesday 3rd November 2021

Launch Visual Studio. Create a new Solution Practical5 and name the first project List1.

Project 1. Linked List class expansion

In this project, we will add new member functions into the List class to expand this class. Download the provided template classes ListNode.h and List.h, and add these classes into the project List1.

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

```
void print();
```

which prints the data items of the list nodes in a list object, from the first node to the last node.

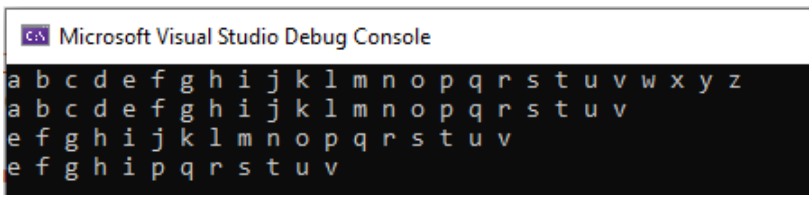
(2) Inside the List class, add a new public function

```
bool remove(int p1, int p2);
```

This function is an extension of the member function `int remove(int p);` which removes a node in a list at position p. The new function will remove a range of nodes from position p1 to position p2 (inclusive). It returns true for successful removal and false for illegal parameters p1 and/or p2.

(3) Create a test1.cpp in the project and use the following main function to test the above new member functions. The expected outputs are shown below the test function.

```
int main()
{
	// create an alphabet list
	List<char> chlist;
	for (int i = 0; i < 26; i++)
		chlist.insert_end(97 + i);

	// print chlist
	chlist.print();

	// remove nodes from 22-25
	if (chlist.remove(22, 25))
		chlist.print();

	// remove nodes from 0-3
	if(chlist.remove(0, 3))
		chlist.print();

	// remove nodes from 5-10
	if(chlist.remove(5, 10))
		chlist.print();

	return 0;
}
```



Project 2. Linked List for storage of data with an unknown size

In Solution Practical5 add a new project List2, in which create a source file test2.cpp. Download the provided corpus.txt file and place it into the List2 folder along with the test2.cpp file. Add the template classes ListNode.h and List.h into the project.

The following shows an incomplete program, showing the use of a linked list (an object of our List class) for loading data of an unknown size – the corpus file mentioned in our lecture - into the program. The program uses C++ file I/O classes for opening the corpus file for reading the data into a linked list, each list node storing a word. To use the class you need to #include <fstream>.

As you can notice, the corpus contains many word items which include some non-alphabetic character(s), e.g., \', \-, \., \,, \", \&, \%, \; …. There is also a sentence code (e.g., 011a0101) at the beginning of each sentence which is not a word. You should ignore all these items and only store the **proper** words, in which each character is an alphabetic letter, in the list.

You are asked to complete this program, in test2.cpp, by completing the missing parts indicated by points 1-4 in the program. After execution, your program should print out the number of the proper words in this corpus file.

Hit: You will find it useful to write a function for testing for proper words (and non-proper words):

```
bool isWord(string item);
```

The incomplete program with missing parts indicated by 1-4

```
int main()
{
	// allocate a string-type list pointed to by pc, used to store all
	// words in the corpus
	List<string>* pc = new List<string>;

	// open the corpus.txt file for input
	ifstream fi("corpus.txt");
	string item;

	// read each item in the file separated by space until the end of the file
	while (fi >> item) {
		// 1. test if the read item is a word - with only letters and
		// no other characters

		// 2. if yes insert the word into the list pointed by pc
	}
	// close the file
	fi.close();

	// 3. print the size of the list
	// 4. free the memory used to store the corpus

	return 0;
}
```

Project 3. Timing operations for array list and linked list

In Solution Practical5 add a new project List3, in which add a source file test3.cpp. Copy the corpus.txt file into the List3 folder with test3.cpp for convenience of its reading into the program. Add the template classes ListNode.h and List.h into the project.

(a) In our lecture video and code, we showed an example of comparing the timings of the insertion operation between the array list and linked list, to compare their performance. Following the lecture example, in this project we first time the deletion operations for the array list and linked list. The following shows two functions for array list insertion and deletion operations. This is followed by a main function, which outlines a plan to time the deletion operations for an array list of a pre-defined Max_ListSize, and a linked list, assuming both lists holding the same data of corpus.txt.

You are asked to convert the testing plan into testing programs in test3.cpp. Then run the tests for variable conditions to compare the performance of the two list data structures for deletion operations.

```
// maxmimum size of array list e.g. 2^20
constexpr int Max_ListSize = 262144;

// insert an item at position p, into an array list with size items pointed by List
template<typename T>
T* insert(T* list, int& size, int p, T item)
{
	if (p < 0 || p > size) {
		cout << "Unable to insert at " << p << endl << endl;
		return NULL;
	}
	else if (size >= Max_ListSize) {
		cout << "List full" << endl << endl;
		return NULL;
	}

	for (int i = size - 1; i >= p; i--)
		list[i + 1] = list[i];
	list[p] = item;
	size++;

	return list;
}


// remove an item at position p, from an array list with size items pointed by List
template <typename T>
T remove(T* list, int& size, int p)
{
	if (p >= 0 && p < size) {
		T item = list[p];
		for (int i = p + 1; i < size; i++)
			list[i - 1] = list[i];
		size--;
		return item;
	}
	else {
		cout << "Unable remove at position " << p << endl;
		exit(1);
	}
}
```

```
int main()
{
	/* array list */
	// 1. generate a string-type array list, alist, of Max_ListSize
	//    initialise size of alist, size, to 0

	// 2. open the corpus file: ifstream fi("corpus.txt");
	//    read each item in the file & insert each proper word at the end of alist
	//    close the file: fi.close();

	// 3. time deletion operations for alist
	//    e.g. remove 100 words at the front, middle or end of alist

	/* linked list */
	// 1. declare a string-type List object llist

	// 2. open the corpus file: fi.open("corpus.txt");
	//    read each item in the file & insert each proper word at the end of llist
	//    close the file: fi.close();

	// 3. time deletion operations for llist
	//    e.g. remove 100 words at the front, middle or end of llist

	return 0;
}
```

(b) Next, we time the operations of traversing a linked list. Rename the above main function to main1, and copy the following main function into test3.cpp. Part 1 of the main is to be completed by you by copying your code from task (a) above.

Part 2 and Part 3 show two methods to traverse the list, from the first node to the last node, to find the longest proper word in the list (you will use the isWord function defined in Project 2). The first method (Part 2) uses the member function **get(int p)** to get the item value at each position p, for p = 0 to listSize – 1. The second method (Part 3) first sets the pointer pCurrentNode to pFirstNode, then steps through the list by calling the member function **get_next()**, each call returns the item value pointed to by pCurrentNode and advances pCurrentNode to pCurrentNode->pNextNode. At the end of each part, the code prints out the longest word found in the corpus.

Time the operations of Part 2 and Part 3 to compare how much time will be needed by each method to find the longest word.

Warning - Part 2 may take a long time to finish but Part 3 is fast!

Make sure you understand what causes the big speed difference between these two methods.

When your run the Part 2 code, you should comment the Part 3 code, and vice versa.

```
int main()
{
	// 1. use a linked list - llist, to store corpus.txt
	/* To be completed by you */

	// 2. traverse llist using get(int p) to each position p in the list to
	// find the longest word
	string word;
	unsigned int wordLen = 0;
	for (int p = 0; p < llist.size(); p++) {
		string item = llist.get(p);
		if (isWord(item) && item.size() > wordLen) {
			wordLen = item.size();
			word = item;
		}
	}
	cout << "The longest word is: " << word << endl;

	// 3. traverse llist using set_first & then get_next to each position in the
	// list to find the longest word
	string word;
	unsigned int wordLen = 0;
	llist.setp_first();
	for (int p = 0; p < llist.size(); p++) {
		string item = llist.get_next();
		if (isWord(item) && item.size() > wordLen) {
			wordLen = item.size();
			word = item;
		}
	}
	cout << "The longest word is: " << word << endl;

	return 0;
}
```

A mock test for Practical Test 1

Practical Test 1

Practical Test 1 will take place from the timetabled Practical session time on Wednesday 10th November. The test will cover the taught material from Lecture 1 to 5, and practical material from Practical 1 to 5. More details about the test arrangement can be found on CSC2059 Canvas.

Take the following as an exercise. Using this Practical 5 as a mock test, practice how your test will be timed and how to upload your submission at the end of the test to Canvas.

On the Canvas Page for CSC2059, click Assignments and then click CSC2059 Practical 5 (the mock test). Click **Take the quiz** to start the mock test. Carefully read the instructions in the test.

At the end of the test document, you will see instructions for how to submit your work to Canvas for marking. For Practical 5, the instructions are:

Submission Notes

1) Save your projects.
2) Make sure that you have the following FOUR source files ready for submission:

   test1.cpp, List.h
   test2.cpp
   test3.cpp

3) ZIP the above 4 files, and DO NOT INCLUDE ANY OTHER FILES.
4) Name the zip file to XXXXXXX.zip, where XXXXXXX is your student number.
5) Check that your .zip file contains the above 4 source files (double click on the .zip file. You do not need to extract the files again).
6) In case something goes wrong with your submission, first make a backup of your zip file on to a USB drive, and/or email it to yourself.
7) On the Canvas Page for CSC2059, click Assignments and then click CSC2059 Practical 5. Go to Question 2 'Upload your work here'. Click Choose a file and then locate the zip. Click Open to upload it. Finally click **Submit quiz** to see your file on Canvas.

The mock test submission won't marked.

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 your questions by using the Ticketing System, or by using emails.

CSC2059 Data Structures and Algorithms

Practical 6: Binary Trees and Recursion

Wednesday 17th November 2021

Launch Visual Studio. Create a new Solution Practical6 and name the first project Recursion.

Project 1. Recursion

The sum of the reciprocals of square numbers produces a convergent series (the Basel problem), with a limit equalling $\pi^2/6$ (≈ 1.64493):

$$\frac{1}{1}+\frac{1}{4}+\frac{1}{9}+\frac{1}{16}+\frac{1}{25}+\frac{1}{36}+\cdots=\frac{\pi^2}{6}.$$

The following shows a function to calculate the sum of the first n reciprocals by using a loop:

```
double rsquare(int n)
{
	double sum = 0.;
	for (double i = 1; i <= n; i++)
		sum += 1. / (i * i);
	return sum;
}
```

You are asked to implement the calculation by using a recursion. In the Recursion project create a test program test1.cpp. Write your recursive function in test1.cpp and test it in the main function by calling it with variable values for n.

In the following, we will try to implement all the operations on tress by using recursions.

Project 2. Extending the TreeNode class

In Solution Practical6 add a new project Tree1. Include the provided binary-tree class TreeNode.h into the project.

(1) Add a member function to the TreeNode class (in TreeNode.h):

```
bool search(T key);
```

which searches the binary tree for the given item 'key', and returns 'true' if the key was found in the tree, and 'false' if it wasn't. Assume the initial tree has at least one item.

In the project, create a test program test2.cpp. In the program construct a binary tree of week days as shown in Fig.1 below:

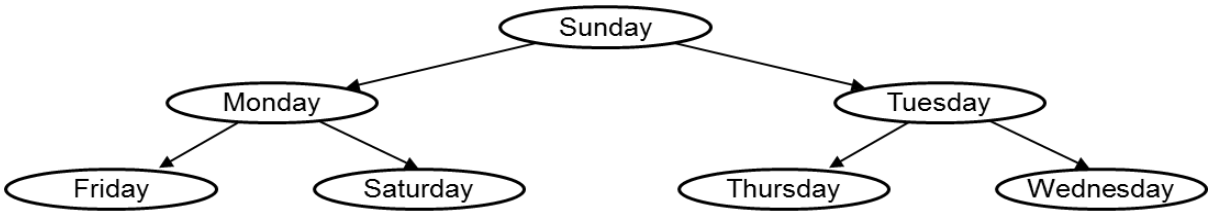


Fig.1 A week-day binary tree

Test your function by searching the above tree for a number of cases:

- search for all seven days
- search for several "days" which aren't in the tree. Try to break it!

For each search, print out its details: what you were searching for, and whether or not you found it.

(2) Add a member function to the TreeNode class:

```
int depth();
```

which returns the depth of the binary tree. Remember that the tree may not be balanced. The depth of an empty tree is 0.

In test2.cpp test your function using the trees you constructed in (1), and some others (e.g., below).

Project 3. Binary search trees

In Solution Practical6 add a new project Tree2. In the project, create a test program test3.cpp.

(1) In test3.cpp write a function:

```
TreeNode<int>* bst();
```

Inside this function you construct a balanced binary search tree (BST) using dynamic allocation, for the random number sequence: 6, 23, 12, 69, 17, 29, 7. The function returns the pointer to the tree. You should use the "search with insertion" method described in the lecture notes to construct this tree.

(2) In test3.cpp write a function:

```
int sigma(TreeNode<int>* tree);
```

The function is called with a binary integer tree. It takes the tree and calculates the sum of all the (int) node values inside the tree; it returns the sum after finishing execution.

(3) To further your experience with tree recursions, in test3.cpp write a function:

```
int maxnum(TreeNode<int>* tree);
```

The function takes a binary integer tree and finds the maximum number inside the tree. It returns the number found after finishing execution.

Finally, in test3.cpp write a main function to test all the three functions. You first call the bst function to construct and obtain the BST. Then you use this BST to test the other two functions sigma and maxnum by printing out the results returned from the functions.

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 your questions by using the Ticketing System, or by using emails.

CSC2059 Data Structures and Algorithms

Practical 7: Introduction to writing, timing and analysing algorithms

Wednesday 24th November 2021

Launch Visual Studio. Create a new Solution Practical7 and name the first project Time1. In project Time1 create a source file test1.cpp.

Project 1. Timing array processing loops

(1) The method used to time parts of your code

To time algorithms in this practical, we will have to slow them down. This is because we can only time to the nearest millisecond, and most of the programs here will run in rather less than a millisecond! We will slow the programs down by including a Sleep operation in the innermost loop. Therefor to time programs you need to do several things:

a) First, you will need to include the library <windows.h>, to enable you to pause your program for one time unit by calling (in the innermost loop):

```
Sleep(1);
```

b) Second, to measure and print the actual time it takes to execute some code, you need to include <time.h> and read the system clock at the start, read it at the end, and subtract the two, giving an answer in milliseconds. You can convert this to seconds by dividing by 1000. For example:

```
clock_t begin = clock();

/* … your code to be timed goes here… */

clock_t  end = clock();

double elapsed = double(end - begin);
cout << "Time taken with N = " << N << " is " << elapsed << " ms"
       <<" = " << elapsed / 1000.0 << " s" << endl;
```

Exercise: in test1.cpp write a function `testTiming(int N)` which executes and times "Sleep(1)" N times. Call the function in the main function with variable N values read from keyboard.

(2) Timing an array processing loop

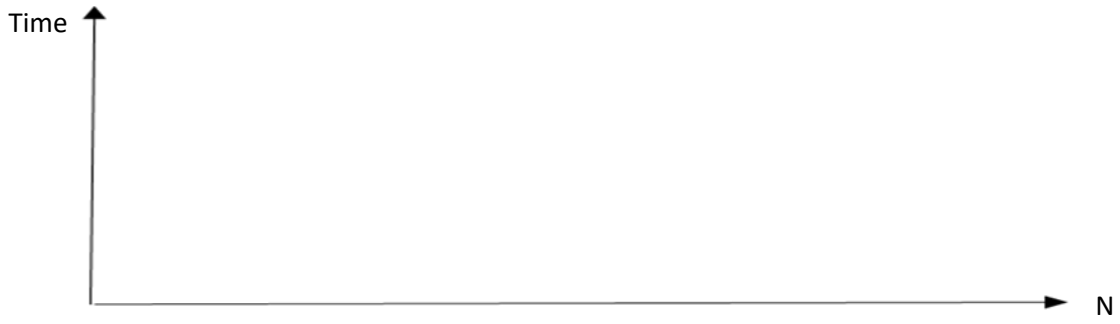
First, reads a value N from the keyboard, and then creates an array of integers of this size. Now initialise the array to contain N random numbers, of range 0 ~ 100000. Test this bit first before going any further. You might want to print out the array's values. Try typing in different values of N each time you run it.

Now that you have your array set up, write a function, `double findAverage(int* pi, int N)`, taking the array (pointed to by pi) and N as parameters, to **find the average** of the numbers in the array (the result will be of type double). Check that your answer is correct for a small value of N.

Note: inside your loop to calculate the average, make sure to include a Sleep(1) operation, to slow it down.

Finally, time your function to find the average. Do this for different values of N (say, 100, 200, 300, up to 1000 or 2000). Plot the times as a graph below. Is this what you expected?

What is the complexity of the findAverage operation, O( ), in terms of N?



(3) How many above average

Write another function, `int find_countAverage(int* pi, int N)`, that first finds the average of an array, and then finds out how many values in the array are above average (it returns the count). Again, in each loop make sure to include a Sleep(1) operation to slow it down. Before you run it, predict the shape of the graph you expect to get.

What is the complexity of the find_countAverage operation, O( ), in terms of N?

(4) Finding the value which is furthest from any other value

For the same array, write a function, `int findFurthest(int* pi, int N)`, which finds which number in the array is the most remote: i.e. it is furthest from any other number in the array. Think of the numbers as being islands spaced out.

For example, if your array contained:

12 20 6 31 40 1 42 19

then the answer should be 31, because the closest number to 31 is 9 away (40). The closest to any of the other numbers is less than this.

Once you have it working (and tested for a small value of N), put a 'Sleep(1)' inside the innermost loop (or inside each loop, if you have separate non-nested loops). Time your algorithm for a range of input values of N. Again, plot these as a graph below.

Questions: Is this what you expected? What is the time proportional to (in terms of N)? What is the complexity of your algorithm? O ( ) Why?

Project 2. Timing linked list traversals

In Solution Practical7 add a new project Time2. In the project, create a test program test2.cpp, and include the provided ListNode.h and List.h classes. In the provided List.h, we have added two Sleep(1) operations for the convenience of the following timing experiment. The experiment conducts a complexity analysis of the two linked list operations, discussed in Practical 5, Project 3, Part (b).

In test2.cpp, first create a linked list (an object of the List class) of N random numbers, in the range 0 ~ 100, with N being taken from keyboard.

Then, write a function, count1, taking the list and an integer key as parameters, to find the number of occurrences of the key number in the list. The function uses get(int p) to cycle through the list to access the number at each positon p for comparing with the key (see Part 2, Page 5, Practical 5).

Next, write another function, count2, which accomplishes the same task as above but uses set_first and then get_next to cycle through the list to compare each number in the list against the key (see Part 3, Page 5, Practical 5).

Finally, time the two functions for a range of input values of N (say, 10, 20, 30, up to 100 or 200). Plot both functions' times in the same graph below.

Questions: For each of the functions, count1, count2,

- What is the time proportional to (in terms of N)?
- What is the complexity, O ( )? Why?

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 your questions by using the Ticketing System, or by using emails.

CSC2059 Data Structures and Algorithms

Practical 8: Searching

Wednesday 1st December 2021

Launch Visual Studio. Create a new Solution Practical8 and name the first project Search1. In project Search1 create a source file test1.cpp.

Project 1. Some simple searches

(1) In test1.cpp write a function

```
int find_first_not_in_range(int* pi, int N, int L, int U);
```

The function takes an array pointed by pi of size N and searches for the first number that doesn't fall in the range [L, U] with specified lower and upper bounds L, U. When successful the function returns the index of the number searched for; otherwise it returns -1.

Write a main function in which create an int array of size N filled with random numbers within the range 0-100. Test your function with variable values for N (e.g., 10, 100, …), and variable bounds [L, U] (e.g., [20, 50], [10, 60], …). At the end of the test print out the following statement:

```
cout << "The complexity of find_first_not_in_range is " << "O(XXX)" << endl;
```

where XXX is what you think the complexity is in terms of the order of N.

(2) In test1.cpp write another function

```
void print_duplicate (int* pi1, int N1, int* pi2, int N2);
```

The function takes two arrays, pi1 and pi2, of size N1 and N2, respectively. It finds the elements in pi1 that are also found in pi2 and prints these elements out as they are being found. Rename the above main to main1 and write a new main function, in which you take N1, N2 from keyboard and use these to create two arrays with random numbers, to test the above function. At the end of the test print out the following statement:

```
cout << "The complexity of print_duplicate is " << "O(XXX)" << endl;
```

where XXX is what you think the complexity is in terms of the order of the size of the problem.

Project 2. Binary search for linked list

In Solution Practical8 add a new project Search2, in which add the provided ListNode.h and List.h classes, and also add a source file test2.cpp.

(1) In test2.cpp, write a function

```
int binarySearch(List<int>* plist, int first, int last, int key);
```

The function takes a pointer to an integer linked list, plist, and performs binary search for the given key in the list, from the position first to the position last. The function returns the position of the key in the list when it is found (position 0 for the first node of the list), or -1 if the key is not found.

In test2.cpp write a main function to test the binary search. You take the list size N from keyboard, create a list of N sorted int values by adding 0 to N – 1 into the list by using the insert_end function,

and then call the above function to search over the complete list for keys that are within or outside the list.

(2) In test2.cpp, write code to perform linear search for the same keys for the list by using

(a) the get(int p) function

(b) the setp_first() and get_next() functions

to cycle through the list to read its node values. Compare the times between these two methods, and with the above binary search method.

Project 3. Hash tables

(1) In Solution Practical8 add a new project Search3, in which add the provided HashTable.h, PhoneDir.h, PhoneDir.cpp and test3.cpp.

The hash table class (HashTable.h) is a template class created by us. We use this example to demonstrate to you the potential near O(1) performance for retrieving information from a hash table. Assume that we look for a person's phone number by giving the person's name as a searching key, from a hash table storing phone directories (simulated by the PhoneDir class).

Run test3.cpp, which allows you to search a hash table created to store persons' name-phone number records. Increase the hash table size, which indicates how many records may be in the table, from 11 to some larger numbers (e.g., 6137, 91193, 391929, …). You do not see the time taken to find the record is lengthened as the size increases (whether the person you search for is in the table or out of the table). Of course this performance may be unrealistic given that there may be hash key collisions between different names. How to design good hash functions and choose good content keys to reduce the probability of collision are some of the topics of current research.

(2) In test3.cpp, write a function

```
size_t hash_index(string date, string hotelname, string address, string customerref,
size_t tableSize);
```

which will be used to calculate the hash indices to implement the database for bookings made by customers on a hotel booking travel web site, where tableSize is the hash table size. To keep the average running time as small as possible, only a maximum number of ten letters from the hotel address are used. All four fields are combined in the given order to form the search key. The hash index is expected to distribute well so you should use a good hash function (e.g. the one introduced in the lecture notes).

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 your questions by using the Ticketing System, or by using emails.

CSC2059 Data Structures and Algorithms

Practical 9: Sorting

Wednesday 8th December 2021

Launch Visual Studio. Create a new Solution Practical9 and name the first project Sort. In project Sort1 create a source file test.cpp.

Project Sort. Testing sorting algorithms

(1) Testing if an array is sorted

In your main method, create an array of N sorted integers, something like:

```
for (int i = 0; i < N; i++)
        a[i] = 3 * i + 1;
```

and an array of N unsorted integers, something like:

```
for (int i = 0; i < N; i++)
        a[i] = rand() % 1000;
```

Now write a function

```
bool isSorted(int* a, int N);
```

which examines any supplied array, of length N, and returns true if the array is sorted in ascending order, and returns false if it isn't.

Test your function for a range of arrays of different length, some sorted and some unsorted.

(2) Sorting an array using Bubble Sort

In test1.cpp, write a new function

```
void bubbleSort(int* a, int N);
```

which will sort the supplied array of size N into ascending order, using the Bubble Sort algorithm discussed in lectures.

Test your method first on a small known array, and then on an array of random numbers. Validate your function using the isSorted function above. Then time the operation for several values of N (10000, 20000, …, 50000) and record the times.

(3) Sorting an array using Quick Sort

In test1.cpp write a new function

```
void quickSort(int* a, int first, int last);
```

which sorts the supplied array, from the first element to the last element, into ascending order, using the Quick Sort algorithm discussed in lectures. Your function should return the result in the original array a. Again, test your method on an array of random numbers, and validate your function using isSorted. (Note: don't run it on the already-sorted result of bubbleSort! You can back up the array a using an array b, before a is sorted by bubbleSort, and then sort b using quickSort for comparing the times between the two algorithms).

To check that you understand the algorithm, print out the array inside the function after the partitioning phase, but before the sorting phase (i.e. before the recursive calls to quickSort). Also print out the value of 'i' (the start position of the upper half after partitioning). Check carefully that you understand exactly why each value ended up exactly where it is.

For example, do this exercise on paper. Trace the program from the code (don't run it):

int b[] = {41, 8, 33, 16, 14, 56, 50, 5};

After 1st partitioning (don't run the program on this array yet! Do it by hand):

b will be: …… i (start of right 'half') will be …….

When sorting the left 'half': after partitioning the left half:

it will be: …… Its i will be …….

When sorting the right 'half' of the original partitioned b: after partitioning the right half:

it will be: …… Its i will be …….

Now, run your program to see if your answers are correct. It should output the values of the array segments after each partitioning phase. See if you made any mistakes in your predictions – and learn from them!

Finally, time the operation for N = 10000, 20000, …, 50000 and record the times.

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 your questions by using the Ticketing System, or by using emails.

Practical Test 2

Practical Test 2 will take place on Wednesday 15th December. The test will mainly cover the taught material from Lecture 6 to 9, and practical material from Practical 6 to 9. Details of the test arrangement will be made available on CSC2059 Canvas.