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:

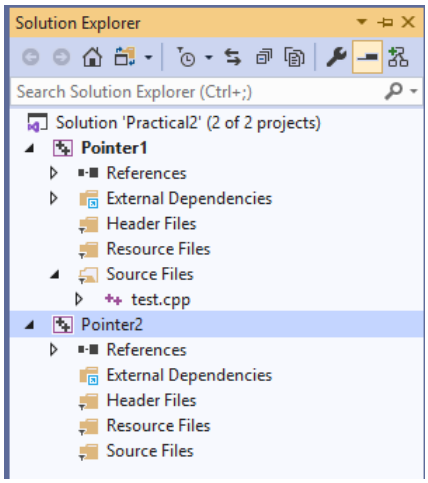
```
int i = 27;          // declare an int variable
int* pi = &i;        // declare an int pointer pi pointing to the address of i
cout << "value of i is " << i << endl;
cout << "value of pi (i.e. address of i) is " << pi << endl;
cout << "value pi is pointing to, or from dereferencing pi, is " << *pi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

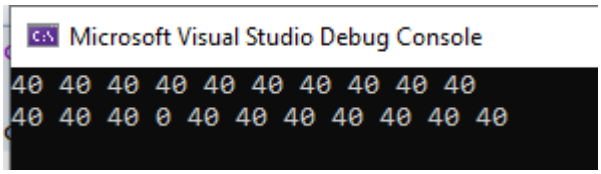
- Check the position of insertion is a legal position i.e. within the range of the array
- Increment size
- Create a new array
- Transfer everything up to pos from the old array to the new array
- Insert val into the new array at position pos
- Transfer the remainder of the old array to the new array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