**Lab Experience Thirteen**

**Objectives:**

1. Introduction to pointers.
2. Pointer referencing of array elements.

**Background**

Variables consist of four components in a computer system and they are:

1. A variable’s name is the way we normally refer to it in a program. Example: **int x;**
2. The variable’s address is the memory location associated with its name.
3. The variable’s type indicates the kind of value to be stored in its memory location, which in turn determines its size, or the number of bytes needed for the variable.
4. The variable’s value is the contents of its memory location.

**What is a pointer?**

A pointer is a variable that can **only**contain an address of a memory location.

**How do you declare pointers?**

A pointer variable is preceded by the dereferencing operator, which is **\***, as follows

**int \*ptr;**

**ptr** is an integer pointer variable. It can only contain addresses of integer data types. I.e. type int. **Note: A common mistake is to assume the variable already contains an address, remember if you don't assign a value to a variable before using it, the contents of the variable is garbage.**

**How do you assign addresses to pointers?**

To assign an address to a pointer you utilize the address of operator, which is **&**, for any variable except arrays and other pointer variables. Why? The array name is a pointer variable and already contains the starting address of the array. Example:

**int \*ptr; // pointer of type integer can only contain addresses**

**// of integers**

**int x = 3; // a regular variable**

**int a[100]; // a integer array**

**double \*dptr; // type doublepointer, can only contain addresses of**

**// doubles**

**double y = 2.5; // primitive variable**

**double b[100]; // a double array**

**ptr = &x; // store the address of x into ptr;**

**dptr = &y; // store the address of y into dptr;**

**ptr = a; // store the starting address of the array a into ptr;**

**dptr = b; // store the starting address of the array b into dptr;**

**ptr = dptr; // Error why? Number of bytes assigned to variables of type**

**// double and int are not equal.**

**De-referencing Pointers**

De-referencing pointers means to access the contents of the memory location whose address is contained in the pointer variable. This is accomplished by the dereferencing operator. Example:

**int \*ptr; // pointer of type int can only contain addresses of integers**

**int x = 3; // a regular variable**

**int a[100]; // a integer array**

**double \*dptr; // pointer of type double can only contain addresses of doubles**

**double y = 2.5; // primitive variable**

**double b[100]; // a double array**

**ptr = &x; // store the address of x into ptr;**

**cout<< \*ptr**

**<<endl; // will display the contents of the variable x. Why?**

**// Because the address of x was assigned to the variable ptr.**

**\*ptr = 5; // changes the contents of x to the value of 5.**

**int \*dummyptr; // declare another pointer**

**cout<< \*dummyptr<<endl; // will generate an error. Why? nothing was assigned**

**// to dummyptr**

When the compiler processes the following declarations

**charch;**

**intintVal;**

it allocates(or sets aside) memory for the variables **ch** and **intVal**. If the memory location set aside for **ch** is **0x08**, and the compiler allocates **ch** and **intVal** in adjacent memory locations, then we might picture memory as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| .... | **0x08** | **0x09** | **0x0A** | **0x0B** | **0x0C** | .... |
|  |  |  |  |  |  |  |
|  | **ch** | **intVal** | | | | |

Such a picture is called a memory map, because it represents a mapping between a program’s variable names and its memory addresses, which typically are represented in hexadecimal (base-16) notation. Note that the memory address associated with **intVal** is**0x09**, even though**intVal**actually consists of locations**0x09**through**0x0C**, asindicated by the shaded part of the picture.

A variable’s name can be thought of as symbolic replacement for its address, because an access to a variable is really an access to its memory location. To illustrate, assigning a value to a variable,

**ch = ‘A’**

simply changes the value of the variable’s memory location. If ASCII code is in use, we can picture the result of such an assignment as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| .... | 0x08 | 0x09 | 0x0A | 0x0B | 0x0C | .... |
|  | **65** |  |  |  |  |  |

**Constant Pointers and Arrays**

Constant pointers are pointers whose contents cannot change. An array name is a constant pointer, which allows greater flexibility in programming.

Given the following declaration:

intxArray[10]; // declares 10 storage locations

Since xArray is a pointer, pointer dereferencing can be used to access each memory location.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **xArray** |  |  |  |  |  |  |  |  |  |  |

What would be contents of **xArray** after execution of the statements below?

**\*xArray = 5;**

**\*(xArray + 4) = 10;**

Answer:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **xArray** | 5 |  |  |  | 10 |  |  |  |  |  |

This means \*xArray is the same as xArray[0] and \*(xArray + 4) is the same as xArray[4]. The reason pointers is used is to increase the speed of execution since the translation step has been removed.

**Pointer Arithmetic**

Arithmetic operations may be performed on pointers. The operations are:

1. ++ and –
2. A constant value may be added or subtracted from a pointer variable. The previous address stored in the pointer is destroyed in favor of a new address.
3. Pointers can be subtracted from other pointers.

Example:

intxArray[10]; // declares 10 storage locations

int \*xptr = xArray; // store the address of xArray into xptr

// notice the absence of & in front of xArray

// Why? Because xArray contains an address

// and pointer variables can only contain addresses

// Thus it is similar to the following assignment:

// int x = 5; int y = 6; y = x; The contents of x is transferred to y

for(inti = 0; i< 10; i++){

\*xptr = 5 \* i;

xptr++;

}

Answer:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **xArray** | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |

**Dynamic Memory**

Every array that has been declared is considered a static allocation of memory. This means the compiler places a request to the operating system for memory before execution of the program begins. It is possible to allocate memory during program execution. This process is called dynamic memory allocation. To allocate memory during execution the **new** operator is used and a request is made to the OS for more memory. The OS can either grant the request or deny the request. If the request is denied a NULL reference is stored into the pointer.

An example of dynamic memory allocation is shown below:

**int \*ptr;**

**ptr = new int [10]; // allocates an array of 10 storage locations**

**if(ptr == NULL){**

**cout<< “\n\n Error allocating memory --- terminating\n\n”;**

**exit(1106); // error out with code 1106**

**}**

**for(inti = 0; i< 10; i++)**

**ptr[i] = 5 \* i; // can use subscripts to reference each memory location.**

**// or can use pointer offset \*(ptr + i)**

**// But you don’t want to change ptr, otherwise a memory leak will occur.**

**delete [] ptr; // when done, return memory back to the OS**

**Lab Exercises Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Directions:**

Start Microsoft word and record the questions and answers to all of the exercises in the lab 13 word document   
Answer the following questions based on material presented in lecture and found in chapters 1-9 of the textbook.

**Lab Exercises:**

1. Look at the following code.

double value = 29.7;

double \*ptr = &value;

Write a cout statement that uses the ptr variable to display the contents of the value variable.

**cout<< \*ptr ;**

1. Write code that dynamically allocates an array of 20 integers, then uses a loop to allow the user to enter values for each element of the array.

**int \*ptr ;**

**ptr = new int [20] ; // request OS to give 20 storage locations**

**if (ptr == NULL) {**

**cout<< "\n\n Error allocating memory --- terminating\n\n";**

**exit (1106) ;**

**} // end if**

**for (int i = 0; i< 20; i++){**

**cout<< "Please enter integer values to be stored in the array: " ;**

**cin>> ptr[i] ;**

**} // end for**

1. Look at the following function definition.

voidgetNumber(int&n){

cout<< "Enter a number: ";

cin>> n;

}// end getNumber

In this function, the parameter n is a reference variable. Rewrite the function so that n is a pointer.

**voidgetNumber(int \*n){**

**cout<< "Enter a number: ";**

**cinn>>\*n;**

**}// end getNumber**

1. Each of the following deﬁnitions and program segments has errors. Rewrite them without errors.

* 1. int x, \*ptr; **int x, \*ptr ;**

&x = ptr; **ptr = &x ;**

* 1. int x, \*ptr; **int x, \*ptr;**

\*ptr = &x; **\*ptr = x;**

* 1. int x, \*ptr; **int x, \*ptr;**

ptr = &x; **ptr = &x;**

ptr = 100; // Store 100 in x **\*ptr = 100;**

cout<< x <<endl; **cout<< x <<endl;**

* 1. int numbers[] = {10, 20, 30, 40, 50};

cout<< "The third element in the array is ";

cout<< \*numbers + 3 <<endl;

**int numbers[] = {10, 20, 30, 40, 50};**

**cout<< "The third element in the array is ";**

**cout<< \*(numbers + 2) ;**

* 1. int values[20], \*iptr;

iptr = values;

iptr \*= 2;

**int values[20], \*iptr;**

**iptr = values;**

**\*iptr \*= 2;**

* 1. float level;

int fptr = &level;

**float level;**

**float \*fptr = &level;**

1. The following function uses reference variables as parameters. Rewrite the function so it uses pointers instead of reference variables, and then demonstrate the function in a complete program.

int pointerFun(int &x, int &y)

{

int temp = x;

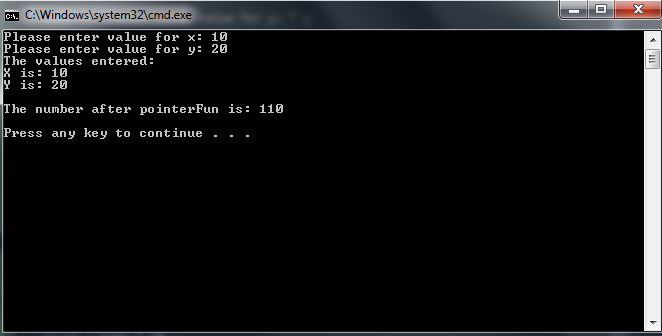
x = y \* 10;

y = temp \* 10;

return x + y;

}

Copy and paste your program into the word document. Capture the output window demonstrating your function works directly below the program.



**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//**

**// Programmer: Chris Dang Class: CSCI 1106 Fall 2014**

**//**

**// Description: Runs program created for exercise 5.**

**//**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include <iostream>**

**using namespace std;**

**int pointerFun (int \*numX, int \*numY) ;**

**int main () {**

**int x, y, newNum ;**

**int \*numX, \*numY ;**

**cout << "Please enter value for x: " ;**

**cin >> x ;**

**cout << "Please enter value for y: " ;**

**cin >> y ;**

**cout << "The values entered:\n" <<**

**"X is: " << x << endl <<**

**"Y is: " << y << endl << endl ;**

**numX = &x ;**

**numY = &y ;**

**newNum = pointerFun( numX , numY ) ;**

**cout << "The number after pointerFun is: " << newNum << endl << endl;**

**return 0 ;**

**} // end main**

**int pointerFun (int \*numX, int \*numY) {**

**int temp = \*numX ;**

**\*numY = temp \* 10 ;**

**return \*numX + \*numY ;**

**} // end pointerFun**

1. Write a program that dynamically allocates an array large enough to hold a user-deﬁned number of test scores (max size of 10 is sufficient for testing). Once all the scores (scores should be randomly generated numbers from 1 to 20) are entered, the array should be passed to a function that sorts them in ascending order. Another function should becalled that calculates the average score. You should have a function that displays the contents of the array. The program should display the array before it is sorted, sorted list ofscores and averages with appropriate headings. **Use pointer notation rather than array notation.** Use the bubble sort to sort your array in ascending order.

Copy and paste your program into the word document. Capture the output window demonstrating your function works directly below the program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// |

// Programmer: Chris Dang Class: CSCI 1106 Fall 2014

//

// Description: Program will ask user input for number of test scores and then

// dynamically allocate memory as their are test scores and then display the

// scores, sort the scores, and then take the average of the scores which will

// then be displayed.

// |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream>

#include <iomanip> // need this for displaying averages

#include <ctime> // need this for random number generator

using namespace std;

const int LOW\_TEST\_SCORE = 1 ; // Lowest score on test is 1

const int HIGH\_TEST\_SCORE = 20 ; // Highest score on test is 20

void displayArray(int a[], int numOfTests) ; // Displays the scores

void loadArray(int \*ptr, int numOfTests) ; // Loads the array with scores

void bubbleSort(int \*ptr, int numOfTests) ; // Sorts the scores

double calculateAvg (int \*ptr, int numOfTests) ;// Calculates average of scores

int main () {

//~~~~~~~~~~~~~Seed for Random Number Generator ~~~~~~~~~~~~~~

srand(time(NULL));

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int numOfTests ;

double avgScore ;

int \*ptr ;

//Get number of test scores///////////////////////////////

cout << "Please enter number of number of tests scores: " ;

cin >> numOfTests ;

while (numOfTests < 1) { //Validates input so that tests are > 1

cout << "Number of tests must be 1 or more.\n" ;

cout << "Please enter number of number of tests scores: " ;

cin >> numOfTests ;

}

//////////////////////////////////////////////////////////

//\*\*\*\*Ask OS for memory locations\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ptr = new int [numOfTests] ;

if (ptr == NULL) {

cout<< "\n\n Error allocating memory --- terminating\n\n" ;

exit (1106) ;

} // end if

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

loadArray(ptr, numOfTests) ;

cout << "\nThe test scores are:\n" ;

displayArray(ptr, numOfTests) ;

bubbleSort (ptr, numOfTests) ;

cout << "\nTest scores after sorting are:\n" ;

displayArray(ptr, numOfTests) ;

avgScore = calculateAvg( ptr, numOfTests) ;

cout << fixed << showpoint << setprecision(2) ;

cout << "\nThe average test score is: " << avgScore << endl << endl ;

delete [] ptr ; // kill process to stop memory leak

return 0 ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function: displayArray

//

// Description: displays a 1D pointer array

//

// Pre: The parameters are the pointer array that has dynamically allocated

// memory and the integer number of tests

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void displayArray(int \*ptr, int numOfTests) {

for(int i = 0 ; i < numOfTests; i++) {

cout << setw(4) << \*(ptr + i) ;

} // end for

cout << endl ;

} // end displayArray

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function: loadArray

//

// Description: loads a 1D pointer array with random numbers

//

// Pre: The parameters are the pointer array that has dynamically allocated

// memory and the integer number of tests

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loadArray(int \*ptr, int numOfTests) {

for (int i = 0; i < numOfTests; i++) {

\*(ptr + i) = LOW\_TEST\_SCORE + rand() % HIGH\_TEST\_SCORE ;

}// end for

} // end loadArray

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function: bubbleSort

//

// Description: sorts a 1D pointer array

//

// Pre: The parameters are the pointer array that has dynamically allocated

// memory and the integer number of tests

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void bubbleSort(int \*ptr, int numOfTests) {

bool swap ;

int temp ;

do {

swap = false ;

for (int i = 0; i < (numOfTests - 1); i++) {

// compares memory location i to location i + 1

if ( \*(ptr + i) > \*(ptr + i + 1)) {

temp = \*(ptr + i) ;

\*(ptr + i) = \*(ptr + i + 1) ;

\*(ptr + i + 1) = temp ;

swap = true ;

} // end if

} // end for

} while (swap) ;

}// end bubbleSort

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function: calculateAvg

//

// Description: calculates the average of a pointer array

//

// Pre: The parameters are the pointer array that has dynamically allocated

// memory and the integer number of tests

//

// Post: Returns the averge of the numbers from the pointer array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double calculateAvg (int \*ptr, int numOfTests) {

int total = 0 ;

double average = 0 ;

for (int i = 0; i < numOfTests; i++) {

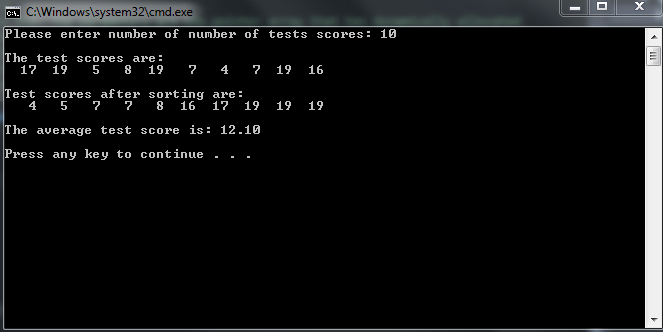
total = total + \*(ptr + i) ;

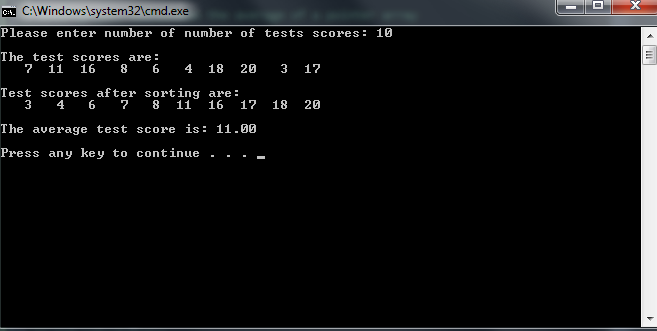
}// end for

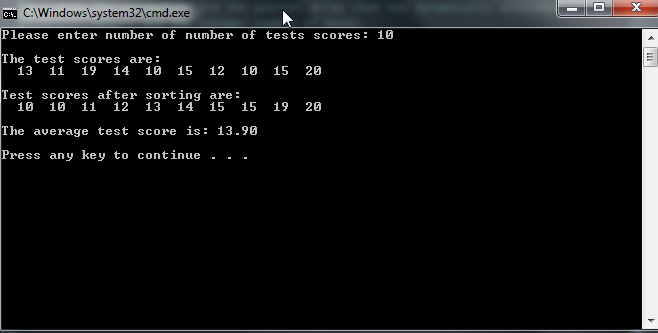
average = total / static\_cast<double>(numOfTests) ;

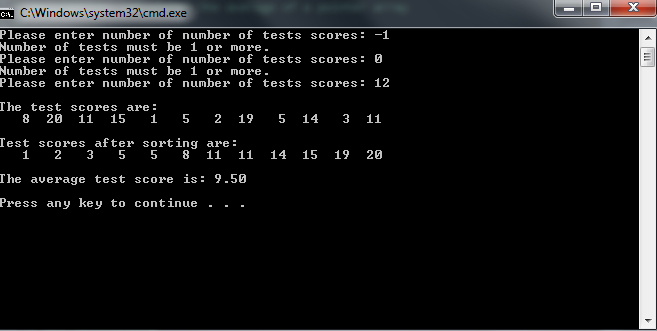
return average ;

} // end calculateAvg









**Due Dates:**According to the due date posted for the drop box folder.

**What to hand in:**

1. Hand in a print out of your word document.
2. Hand in a print outs of your program.
3. Compress the .cpp files and the word processed document into a single compressed file called **{yourname}Lab13.zip** e.g. timwrennlab13.zip.
4. Place the compressed file into the lab 13drop box folder.