John Rollinson

CS 151 SPRING 2020

MODULE 2:

Chapter 10 Checkpoints:

10.1:

The symbol “&” is called an address operator, it returns the memory address of a variable. The addresses of the variables are arbitrary. The position of the address operator & is always in front of the variable name. The following statement displays the address of the variable name, “count”:

Void main()

{

Int count = 10;

Cout << “The address of the variable count is “ << &count;

Cout << “The value of the variable count is “ << count << endl;

}

Output:

The address of the variable count is 0017FDE0

The value of the variable count is 10

10.2:

Write a statement defining a variable dPtr. The variable should be a pointer to a double.

//Defining pointer variable of double

Double \*dPtr;

10.3:

Uses of \* symbol in C++ are:

1. Multiplication operator.
2. Pointer definition.
3. Indirection operator.

10.4:

What is the output of the following program?

**#include <iostream>  
 using namespace std ;  
 int main()   
 {  
 int x = 50, y = 60, z = 70 ;  
 int \*ptr = nullptr ;  
 cout << x << " " << y << " " << z << endl ;  
 ptr = &x ;  
 \*ptr \*= 10 ;  
 ptr = &y ;  
 \*ptr \*= 5 ;  
 ptr = &z ;  
 \*ptr \*= 2 ;  
 cout << x << " " << y << " " << z << endl ;   
 return 0 ;  
 }**

Output:

50 60 70

500 300 140

10.5:

Rewrite the following loop so it uses pointer notation (with the indirection operator) instead of subscript notation.

**for (int x = O; x < 100 ; x++)   
   cout << array[x] << endl ;**

The indirection operator (\*) is used with the pointer variables. The name of an array denotes the starting address of that array. Therefore, the indirection operator can be used with the name of the array also. The specified code segment prints the first 100 values stored in the array using the subscript operator. The following code segment prints the first 100 values stored in the array using the pointer notation with the indirection operator and name of the array:

//repeat the loop for 100 times

For(int x = 0; x < 100; x++)

{

//print the value stored at the current location in the array.

Cout << \*(arr + x) << endl;

}

10.6:

Assume ptr is a pointer to an int and holds the address 12000. On a system with 4-byte integers, what address will be in ptr after the following statement?

**ptr += 10 ;**

The value of ptr after execution of ptr += 10 is 12040.

10.7:

A) **ptrInt ++ ;**  
 B) **--ptrInt ;**  
 C) **ptrInt /= 2 ;**  
 D) **ptrInt \*= 4 ;**  
 E) **ptrInt += x ;** (assume x is an int.)

A) pint++; is a valid statement, it is termed as an advanced pointer.

B) pint--; is a valid statement, Here the pointer variable processed as a decrement operator.

C) pint /= 2; is an invalid statement. Pointer variables perform only some arithmetic calculations. So, it will give an error like: “Expression must have arithmetic or unscoped enum type.”

D) pint \*= 4 is an invalid statement. Multiplication (\*) operator only works after the pointer variable is dereferenced. So a valid statement is \*pint \*= 4;

E) pint += x is a valid statement. It adds some offset address to the pointer variable.

10.8:

A) **int ivar, int \*iptr = &ivar ;**  
 B) **int ivar, \*iptr = &ivar ;**  
 C) **float fvar ;**  
 **int \*iptr = &fvar ;**  
 D) **int nums[50], \*iptr = nums ;** E) **int \*iptr = &ivar ;**  
 **int ivar ;**

A) The statement declares an integer pointer named iptr and set this iptr to the address of the integer variable ivar. The data types of both the variables ivar and iptr are the same. Therefore, these statements are not producing any errors. Hence, the definition is valid.

B) The above statement declares an integer variable named ivar, then defines an integer pointer named iptr to the address of the integer variable ivar. The data types of both the variables ivar and iptr are the same. Therefore, this statement is not producing any errors. Hence, the definition is valid.

C) The first statement declares a float variable named fvar. The second statement defines an integer pointer named iptr and set this iptr to the address of the float variable fvar. The data types of both the variables fvar and iptr are different. Therefore, the second statement is producing a compilation error. Hence, the definition is invalid.

D) The above statement declares an array named nums of 50 integer values, then defines an integer pointer named iptr and set this iptr to the starting address of the array nums. The ampersand (&) is not required to get the address of the array because, the starting point of the array specifies the address of the entire array. The data types of both the variables nums and iptr are the same. Therefore, this statement is not producing any errors. Hence, the definition is valid.

E) The first statement defines an integer pointer named iptr and set this iptr to the address of an undeclared variable ivar. This statement producing an error while assigning the address of an undeclared variable ivar to the integer pointer iptr. The second statement declares an integer variable named ivar. Hence, the definition is invalid.

Review Questions:

33.

**void change(int \*p) {   
 \*p = 20 ;  
 }**

**int i ;**

#include<iostream>

Using namespace std;

//Function prototype

Void change(int \*p);

//Main function

Int main()

{

Int I;

//function calling

Change(&i); //i will be sets to 20

} //end of main

//Function definition

Void change(int \*p)

{

\*p = 20;

}

34.

**void modify(int &x) {   
 x = 10 ;  
 }**

**int i ;**

#include<iostream>

Using namespace std;

//Function prototype

Void modify(int &x);

//Main function

Int main()

{

Int i;

//Function calling

modify(i);

} //End of main

//Function definition

Void modify(int &x)

{

X = 10;

}

Number Conversions:

* Convert the following decimal numbers to binary:

1 = 0001

9 = 1001

10 = 1010

100 = 1100100

624 = 1001110000

* Convert the following decimal numbers to hexadecimal:

10 = A

40 = 28

624 = 270

999 = 3E7

* Convert the following hexadecimal numbers to decimal:

5 = 5

A = 10

3f = 63

100 = 256

Bd92 = 48530