Lab objective

The goal of this lab is to understand the use of pointer variables in C++. One important use of pointer variables is their use in conjunction with nameless variables. In other words, variables that are dynamically allocated at run time using the “new” operator.

**Step 1)** To begin with, let's create the following C++ program. Then compile and run it. Everything should be compiling and running at this time.

```cpp
#include <iostream>
#include <bitset>
using namespace std;

// Forward declaration
void pointers_1(void);
void pointers_2(void);
void pointers_3(void);
void pointers_4(void);
void pointers_5(void);
void pointers_6(void);

text

void main()
{
    //pointers_1();
    //pointers_2();
    //pointers_3();
    //pointers_4();
    //pointers_5();
    //pointers_6();
    system("pause");
}

OUTPUT: Display the Screen Shot of the above program and include it in Lab3 submission.

**Step 2:** Now let's add the following function to the bottom of your program. Then look at the comments in the function, and below each comment line write the corresponding C++ code:

```cpp
void pointers_1(void)
{
    // declare I as integer
    // declare p as a pointer to an integer
    // assign p to the address of i

    // print i
    // print the address of i
```
// print p
// print the content of p
// print the address of p
}

Now run the program and examine the results.

**OUTPUT:** Display the Screen Shot of the above program and include it in Lab3 submission.
If you had difficulty with the above, review the following:

```cpp
void pointers_1(void)
{
    int i = 3;  // i is an integer
    int *p;     // p is a pointer to an integer
    p = &i;     // p is pointing to the address of i
    cout << "i =  " << i << endl;   // 3
    cout << "i =  " << std::bitset<32>(i) << endl;  // Print in binary
    cout << "&i = " << &i << endl;   // Show the memory address of variable i
    cout << "p  = " << p << endl;   // What is p
    cout << "*p = " << *p << endl;  // Content of p

    cout << "What about &p = " << &p << endl;
}
```

**Step 3:** Now let’s add the following function to the bottom of your program. Then look at the comments in the function, and below each comment line write the corresponding C++ code:

```cpp
void pointers_2(void)
{
    int i = 3;
    int *p;
    p = &i;

    int **q;
    q = &p;

    // print i
    // print p
    // print q
    // print the content of q
    // print the content of content of q

    // change the value of i to 33 without touching i (hint use q)

    // print i
}
```
If you had difficulty with the above, review the following:

```c
void pointers_2(void)
{
    int i = 3;
    int j = 5;

    int *p;  // pointer to an integer
    p = &i;

    // Let's go one level deeper (make a pointer to a pointer!)
    int **q;  /* why ?*/
    q = &p;

    // Creating a pointer to a pointer will allow us
    // to change both p and i
    cout << "Let's see what we have:" << endl;
    cout << "  i = " << i << endl;  // 3
    cout << "  p = " << p << endl;  // What is p
    cout << "  q = " << q << endl;  // what is q
    cout << "  *q = " << *q << endl;
    cout << "  **q = " << **q << endl;

    cout << "----Let's reach in and change i -------------------" << endl;
    **q = 33;  // change 3 to 33
    cout << "  **q = " << **q << endl;
    cout << "  i = " << i << endl;  // 33

    cout << "----Now Let's reach in and change p----------------" << endl;
    *q = &j;
    **q = 55;
    cout << "  i = " << i << endl;  // 33
    cout << "  j = " << j << endl;  // 33
}
```
Step 4: Now let’s add the following function to the bottom of your program. Then look at the comments in the function, and below each comment line write the corresponding C++ code:

Review the topic on dynamic memory allocation in your text book. (read about new and delete)

```cpp
void pointers_3(void)
{
    int x = 5;
    int *p;
    p = new int(10); // Dynamically allocate an integer

    // print x
    // print p
    // print the content of p

    // delete p;
    // now print p again
    // now print the content of p
}
```

If you had difficulty with the above, review the following:

```cpp
void pointers_3(void)
{
    //Static integer variable :
    int x = 5;

    //Dynamic integer variable :
    int *p;    // pointer to a variable
    p = new int(10); // Dynamically allocate an integer

    cout << "x = " << x << endl;
    cout << " p = " << p << endl;
    cout << "*p = " << *p << endl;

    // what happens if we assign p to the address of x?  
    // is it legal?  
    // if so, what happens to the memory that had the value of 10?  (Lost memory?!) 
    //p = &x;
    //cout << "*p = " << *p << endl;

    // what about the following? 
    char c = 'A'; 
    // can we assign p to &c 
    // is it legal? if not, why not?  
    //p = &c;
    delete p;
    cout << " p = " << p << endl;
    cout << "*p = " << *p << endl;
}
```
Step 5: Now let’s add the following function to the bottom of your program. Study and then execute the code.

```cpp
// Dynamic vs. Static array allocation and use!
void pointers_4(void)
{
    cout << "Dynamic Array....." << endl;
    int *myDynamicArray;

    myDynamicArray = new int[10];     // create a 10 integers
    for (int i = 0; i < 10; i++)
        myDynamicArray[i] = i;

    for (int i = 0; i < 10; i++)
        cout << myDynamicArray[i] << endl;

    cout << "note use of Dynamic array and static array are similar!" << endl;
    cout << "what does that tell you?" << endl;

    cout << "Static Array....." << endl;
    int myStaticArray[10];
    for (int i = 0; i < 10; i++)
        myStaticArray[i] = i;

    for (int i = 0; i < 10; i++)
        cout << myStaticArray[i] << endl;

    cout << "That tells you that staticArrays are essentially constant pointers to array of values" << endl;
    cout << "However, we can free the dynamic array using 'delete'" << endl;
    delete myDynamicArray;

    //delete myStaticArray;

    cout << "note that the myStaticArray Name and the &myStaticArray[0] point to the same location!" << endl;
    cout << myStaticArray << endl;
    cout << &myStaticArray[0] << endl;

    cout << "also *myStaticArray and myStaticArray[0] have the same value" << endl;
    cout << *myStaticArray << endl;
    cout << myStaticArray[0] << endl;
}
```
Step 6: Now let’s add the following function to the bottom of your program.

Study the following code segment about creating a dynamically allocated array.

```cpp
// 2D Arrays as a single dimension array!
void pointers_5(void) {
    int rows = 3;
    int cols = 5;
    int counter = 0;

    cout << "Dynamic Array.....a 2D array as a long 1D array!" << endl;
    int *myDynamicArray;

    myDynamicArray = new int[rows*cols]; // create 15 cells (1 dimension)

    for (int i = 0; i < rows * cols; i++) {
        myDynamicArray[i] = i;
    }

    for (int i = 0; i < rows * cols; i++) {
        cout << myDynamicArray[i] << " ";
    }

    cout << endl;
    cout << "Now lets print the array as a 2D array" << endl;

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++) {
            myDynamicArray[(i * cols) + j] = (i * cols) + j;
        }

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
            cout << myDynamicArray[i * cols + j] << " ";
        cout << endl;
    }

    // to free the array
    delete myDynamicArray;
}
```
Step 7: Now let’s add the following function to the bottom of your program. Then look at the comments in the function, and below each comment line write the corresponding C++ code:

Study the following code segment about dynamically creating a 2-D array. By first creating an array of pointers, then pointing each element of the array to a 1-D array of integers.

```
// 2D Array as an array of pointers to arrays!
void pointers_6(void)
{
    int rows = 3;
    int cols = 5;
    int counter = 0;

    cout << "Dynamic Array..... using array of pointers to arrays" << endl;
    int **myDynamicArray;

    typedef int* intPtr;    // Define a new type

    myDynamicArray = new intPtr[rows];  // create an array of pointers to int

    for (int i = 0; i < rows; i++)
        myDynamicArray[i] = new int[cols]; // now create an array of size cols and
                                            // point the pointer to it.

    cout << "Now let’s print the array as a 2D array" << endl;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            myDynamicArray[i][j] = (i * cols) + j;
        }
    }

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
            cout << myDynamicArray[i][j] << " ";
        cout << endl;
    }

    cout << "To free the array we must free all of its sub arrays" << endl;
    for (int i = 0; i < rows; i++) {
        delete myDynamicArray[i];
    }
}
```