CS 122 Lab 5

Dynamic Arrays — Paul Cao

There is a close association between pointers and arrays. We have learned how array elements were accessed. In C++, an array variable is actually a pointer variable that points to the first indexed variable of the array. Array elements can be accessed using pointer notation as well as array notation. In this lab, we will practice how to use dynamic arrays to achieve efficient coding. This work is due before midnight on 2/24/2010.

Pre-lab activity

You don't have to turn in any work for this pre-lab.

There is a close association between pointers and arrays. In C++, an array variable is actually a pointer variable that points to the first indexed variable of the array. Array elements can be accessed using pointer notation as well as array notation. However, until we have learned dynamic arrays, we have to declare arrays with a fixed size before we can use it. This constraint makes our code less flexible. For example, if we want to ask the user to enter some numbers for us and we will split the numbers into two categories: positive and negative. Without dynamic arrays, we have to use partially filled arrays to perform the task. With the help of pointers and dynamic operators like new and delete, we can perform this task in a much more flexible manner. The following code splits an array of integers into two arrays: one for the positives and the other for negatives. Please read the code and pay attention about how dynamic arrays are used.

```
// split.cpp - A program that splits an array of integers into
//             two arrays: one for the positives and the other
//             for the negatives

#include <iostream>
using namespace std;

typedef int* IntPtr;

int main()
{
    int pos = 0, neg = 0;
    int pos_size=0,neg_size=0;
    int arraySize;

    cout << "Enter the number of integers: ";
    cin >> arraySize;

    // Allocate memory for array sample
    IntPtr sample = new int[arraySize];

    // Read the integers into array sample
    cout << "Enter " << arraySize << " integers: ";
    for( int i = 0 ; i < arraySize ; i++){
      cin >> sample[i];
      if(sample[i]>=0)
            pos_size++;
      else
```

```
            neg_size++;
    }
    // Echo user's input
    cout << "Array sample before splitting it: " << endl;
    for( int i = 0 ; i < arraySize ; i++)
      cout << sample[i] << " ";
    cout << "\n\n" << endl;

    // Allocate memory for arrays positives and negatives
    IntPtr positives = new int[pos_size];
    IntPtr negatives = new int[neg_size];

    // Split the array into two arrays
    for( int i = 0 ; i < arraySize ; i++)
      if( sample[i] >= 0)
            positives[pos++] = sample[i];
      else
            negatives[neg++] = sample[i];

    // Display array positives
    cout << "Array Positives: " << endl;
    for( int i = 0 ; i < pos_size ; i++)
      cout << positives[i] << "      ";
    cout << "\n" << endl;

    // Display array negatives
    cout << "Array Negatives: " << endl;
    for( int i = 0 ; i < neg_size; i++)
      cout << negatives[i] << "      ";
    cout << "\n" << endl;

    // Cleanup: Delete memory allocated for the three arrays
    delete [] sample;
    delete [] positives;
    delete [] negatives;

    // Done. Exit
    return 0;
}
```

In this code, dynamic arrays were constructed using the new operator and array elements are accessed like regular arrays. It fact, sample[i] is the same as *(sample+i) thus it is up to you to decide what you want to use in your code.

In-lab activity

Write a code that can add Hexadecimal numbers. Hexadecimal numbers are integers written in base 16. The 16 digits are 0 through 9 plus a for 10, b for 11, c for 12, d for 13, e for 14, and f for 15. For example, the hexadecimal numeral d is the same as the base 10 number 13 and the hexadecimal numeral 1d is the same as base 10 numeral 29. Write a C++ program to perform the addition of two hexadecimal numbers.

The user will provide how long each number is to your code. You can assume that the two numbers have the same length but you can't limit how long the numbers can be. Thus you should use dynamic arrays to store hexadecimal numerals as array of characters. Include a loop to repeat this calculation for new numbers until the user says he or she wants to end the program.

Note:

- You shouldn't use the hexadecimal processing tools in C++ for this lab.
- Since it is possible that the addition of two n-digit numbers results in a n+1 digit sum, your sum dynamic array should be properly designed.
- Don't forget to properly delete the dynamic arrays when they are no longer used.
- Since you need to align your output properly, setw function may be handy.
- If you need to do basic character I/O, be careful about the \n in the input stream. cin.ignore function may be helpful. If you decide to use cin.getline function for input, you should leave space for the \0 terminator for the dynamic char array.

Sample output

```
Welcome to C++ hex adder!
How long are the numbers we are about to add: 4
Number 1 pls: 1111
Number 2 pls: 2222
         1111
        +2222
--------------------
         3333
Do you want to add another two numbers? yes or no: yes
Welcome to C++ hex adder!
How long are the numbers we are about to add: 10
Number 1 pls: 12345fedcb
Number 2 pls: fffffffff
         12345FEDCB
        +FFFFFFFFF
--------------------
        112345FEDCA
Do you want to add another two numbers? yes or no: yes
Welcome to C++ hex adder!
How long are the numbers we are about to add: 2
Number 1 pls: 79
Number 2 pls: 01
         79
        +01
--------------------
         7A
Do you want to add another two numbers? yes or no: no
Bye, lab 5!
```