Comments

Due: NA

Comments are important! You think they are not, but they really are. In the real world, your comments will allow another programmer to read and understand your code much faster. Un-commented code could (and does) take much much longer to digest. I read my own programs from just a few months ago and wonder … what was I thinking?!?

In the case of academia, they are still important. Your comments help me (or the grader) figure out if your program works as expected, or what you were trying to do if things aren't working properly.

- RULE 1: Every program will have comments.
- RULE 2: If I didn't mention it in class or write it on the assignment, refer to RULE 1.

Example Program Comment Block

Every program will have a comment block nearly IDENTICAL in structure to the one below. Do NOT remove the slashes. It makes a very distinct BOX like effect. Keep it as is and only replace the pertinant information. This is the comment that will go into your main driver program. It is also the top page when turning in a printed program (unless we create a banner page).

Acceptable

```
//////////////////////////////////////////////////////////////////////////
/////
//
//
//
//////////////////////////////////////////////////////////////////////////
/////
```

OR

```
/*************************************************************************
****
*
*
*
*************************************************************************
***/
```

Program Comment Template:

```
/*****************************************************************************
****
*
*  Author:           (your name)
*  Email:            (your email address)
*  Label:            (program's label from assignment list)
*  Title:            (short title from assignment, if any)
*  Course:           (course number and prefix)
*  Semester:         (semester and year)
*
*  Description:
*        describe program here thoroughly
*
*  Usage:
*        how to use the program if necessary
*
*  Files:            (list of all source files used in this program)
*****************************************************************************
***/
```

Program Comment Example:

```
/*****************************************************************************
****
*
*  Author:           Terry Griffin
*  Email:            terry.griffin@msutexas.edu
*  Label:            A04
*  Title:            Linked List Class
*  Course:           CMPS 3013
*  Semester:         Spring 2020
*
*  Description:
*        This program implements a class that allows a linked list to be
used just like
*        an array. It overloads the "[]" (square brackets) to simulate
accessing seperate
*        array elements, but really it traverses the list to find the
specified node using
*        an index value. It also overloads the "+" and "-" signs allowing
a user to "add"
*        or "push" items onto the end of the list, as well as "pop" items
off the end of our
*        array. This class is not meant to replace the STL vector library,
its simply a project
*        to introduce the beginnings of creating complex / abstract data
types.
*
*  Usage:
```

```
*       - $ ./main filename
*       - This will read in a file containing whatever values to be read
into our list/array.
*
*  Files:
*       main.cpp    : driver program
*       list.h      : header file with list defintion
*       list.cpp    : list implementation
*************************************************************************
***/
```

Class Comment

Class Comment Template:

```
/**
 * Class Name
 *
 * Description:
 *      Description of your class and what it does
 *
 * Public Methods:
 *      - A list of
 *      - each public method
 *      - with return types
 *
 * Private Methods:
 *      - A list of
 *      - each private method
 *      - with return types
 *
 * Usage:
 *
 *      - examples of how
 *      - to use your class
 *
 */
```

Class Comment Example:

```
/**
 * Huffman
 *
 * Description:
 *      This class implements a compressions algorithm called Huffman
Coding.
 *      Huffman coding assigns codes to characters such that the length of
the
```

```
 *      code depends on the relative frequency or weight of the
corresponding
 *      character. Huffman codes are of variable-length, and prefix-free
 *
 * Public Methods:
 *                          Huffman()
 *      void                BuildFrequencyTable(string filename)
 *      string              LookupCode(char key)
 *      void                Analyze()
 *      map<char, string>   GetCodes()
 *
 * Private Methods:
 *      void                _BuildLookupTable
 *      void                _BuildTree
 *      int                 _maxDepth
 *
 * Usage:
 *
 *      Huffman H(filename):                        // Create Instance of
Huffman
 *                                                  // and build freq
table.
 *      H.GetCodes();                               // get map
<char,string> of codes
 *
 *                                                  // or
 *
 *      Huffman H;                                  // do seperately
 *      H.BuildFrequencyTable(filename);            // or use to re-build
another file
 *      H.LookupCode('s')                           // get code for 's'
 *
 */
```

Function Comment

```
    /**
     * Public/Private/Protected : function_name
     *
     * Description:
     *      Describe the functions purpose
     *
     * Params:
     *      - list params
     *      - one per line
     *      - with return type
     *      - and one line description
     *
     * Returns:
```

```
 *      - what does this function return (including the type)?
 */
```

Function Comment Example:

```
/**
 * Public : LoadList
 *
 * Description:
 *      Loads an array of integerts into a linked list.
 *
 * Params:
 *      int*    :  array of integers
 *      int     :  array size
 *
 * Returns:
 *      List*   : a pointer to a linked list of integers.
 */
```

Comments in General

- Variable declarations should have a comment after them:

```
T **Array;          // Pointer to allocate dynamic array
int Next;           // Next available location
int MaxSize;        // Max size since were using array
int HeapSize;       // Actual number of items in the array.
bool isMax;         // true = max heap false = mi
```

- Code should be commented enough to describe action in general:
- The following is taken from a “heap” class. Knowing in general how a heap works, and by having access to the other code in the file, the comments below are enough to give the reviewer an idea of what this function is doing.

```
/**
 * Function PickChild:
 *      Return index of child to swap with or -1 to not swap.
 *
 * Params:
 *      [int] index - index of parent element
 * Returns
 *      [int] index - index to swap with or -1 to not swap
 */
int PickChild(int i) {
    if (RightChild(i) >= Next) {    // No right child
```

```
            if (LeftChild(i) >= Next) { // No left child
                return -1;
            } else { // you know there is only a left child
                return LeftChild(i);
            }
        } else {        //right child exists

            if(isMax){  //This is a "maxheap"
                        // return child with "greater" value
                if (Array[RightChild(i)]->priority > Array[LeftChild(i)]-
>priority) {
                    return RightChild(i);
                } else {
                    return LeftChild(i);
                }
            }else{
                // return child with "smaller" value
                if (Array[RightChild(i)]->priority < Array[LeftChild(i)]-
>priority) {
                    return RightChild(i);
                } else {
                    return LeftChild(i);
                }
            }

        }
    }
```

Style of Comments

Good:

```
   int num1, num2, num3;     // user entered numbers
   double average;           // calculated average of the numbers
   int score1,               // score on exam 1
       score2;               // score on exam 2
```

Bad:

```
   int num1, num2, num3;          // user entered numbers
   double average;    // calculated average of the numbers
   int score1,    // score on exam 1
       score2;               // score on exam 2
```