

**DUE DATES**

To receive full credit, each programming assignment must be handed in **on the due date no later midnight**.

Programs may be turned in after the due date for partial credit. You will lose a letter grade for each week the programming assignment is late.

There are actually 2 deadlines for each programming assignment an initial deadline (marked as for example '1 I' on the schedule) and a final deadline (maked as '1 F'). You are required to upload your first version of the assignment to your educator account by the initial deadline. Each day, during my office hours from 9am to 11am, I will check for all programming assignment that have been submitted. I will grade and return the assignment, with a score of either acceptable or unacceptable.

An acceptable score means that your program compiles, runs, is correct, solves all of the requirements asked for in the assignment and is written using good programming style.

An unacceptable score means that one or more of these criteria were not meet. If you program is unacceptable I will give a brief description of the problem. You may fix the problem and resubmit your programming assignment, up to the final deadline (and after, though you loose 1 letter grade per week late as stated above).

**PROGRAMMING STYLE**

Each program you turn in **must** contain an initial block of comments with this information:

- 1) your name
- 2) the class
- 3) the assignment number
- 4) the compiler you are using
- 5) the main function file should name **all** of your source files needed to run the program
- 6) a detailed statement of the purpose of the program

Assume the reader has not seen the program assignment and needs a general understanding of what your program does and how it works. Try to answer these questions in your statement: What values does the program input? What values does it output? What kind of processing does the program do to obtain the output values? Explain any processing which would not be obvious to a reader not familiar with the assignment.

The program should contain sufficient additional comments to clarify what you're doing. A reader of the program should be able to determine from the comments alone a general idea of what is happening without actually having to decipher any code.

Avoid redundant comments which don't add to the reader's comprehension. Good comments explain **why** you're doing something or **what it means** in the context of the program.

Here's an example of what I consider to be helpful comments in code searching for the largest value in an array. Also notice how the variable names are chosen to describe the values the names represent.

```

hiVal = arr[0];           // Assume first element's value is largest
hiSub = 0;               // Remember position of largest value
n = 1;                   // Start searching with the second element
while (n < MAX)          // Look at each element
{
    if (arr[n] > hiVal)   // If this element value is larger than previous largest,
    {
        hiVal = arr[n];   // save it as largest,
        hiSub = n;        // and remember its position
    }
    n++;                  // Look at the next element
} // end of loop to find the largest element

```

Modularize your program. The main function should have minimal references to implementation-dependent details. Unless a particular assignment states otherwise, **only file variables may be declared globally; all other variables used by a function must be declared locally or passed as parameters**. Every program module should have a block of comments explaining what the module does.

When creating a class try to make it general and flexible for maximum reusability. Always provide class access functions to return private variable values so that clients can code their own functions for operations the class implementer didn't anticipate.

Choose reasonably descriptive identifiers to name variables, constants, functions, classes, etc. Single-character identifiers are unacceptable except for variables like subscripts and loop-control variables (and a more descriptive identifier is recommended for these variables whenever it is appropriate).

Please use at least minimal indentation standards such as indenting statements within a loop, aligning the clauses of IF statements, etc. This will not only make me happy but will also make your life easier in reading and debugging your own programs. Following the coding styles used in the Malik book programming examples would be one way of choosing a good coding style. In any case, whatever style you choose be consistent in applying it throughout your code.

Use blank lines liberally to separate sections of your program, and use extra spacing within a statement to enhance readability.

## **INPUT FILES**

If an assignment specifies an input file, that file will typically be provided for you and be available on my web page. It doesn't matter where you put the file when you test your program; I will expect to change your filename path.

## **HAND IN**

We will be using the Educator online system to hand in and hand back programming assignments, as well as other information. For each assignment, upload a machine-readable copy of all source files needed to run your program. Since I will compile and run each program myself, it is not necessary to give me other files, like workspace or executable files. Please make sure that your name appears in comments in **every** file of your program. I would like people to use a convention for uploading assignments. Always give the name of your file the same as the name of the assignment, and append a version number. For example, if this is the first time you are turning in programming assignment #1, you should call the file prog1-v1.cpp. If you have submitted a version 1 and it is returned as unacceptable and you consequently fix and want to submit another version, upload the file with the new name prog1-v2.cpp. Continue increasing the version number as required.

## **PROGRAM EVALUATION**

As indicated previously, you will receive either an acceptable or unacceptable evaluation for each version of your assignment. You may fix and resubmit unacceptable programs. Some programs have possible extra credit, if you complete the extra credit correctly I may give you an excellent rating, which would translate into a little bit extra for each excellent on your final grade determination.

## **PLAGIARIZING**

Programs which are so similar that they are almost certainly copies of one another will receive a grade of unacceptable. Unfortunately this penalizes the original author as well as those who cheat. To protect yourself from this penalty, protect your own work. Don't leave copies of your program on the hard disk of a lab PC and don't leave diskettes or hard copies of your program lying around. Definitely do not give anyone else a copy of your program.

## **Tips for successful programming:**

- 1) Spend some time on the design of your algorithm and consider possible alternatives before you do any coding.
- 2) Design your program in modules, no matter how you actually code it.
- 3) Code and test one module at a time.
- 4) Always echo your input to the screen to make sure your data is being read properly.
- 5) Don't wait until the last minute to start working on a programming assignment.