

COMP1200c - Assignment 1
Due 11:59 pm – Wednesday – January 25
Submit `assign1a.c` and `assign1b.c` via Blackboard

PART A.

The following steps will guide you in entering the commands necessary to complete this assignment. These basic steps will be used in all future assignments. This assignment exposes you to several C statements that you will learn about in this course. For now, type as given. Try reading the programs to see if you can understand what they are doing.

0. Read Chapter 2 in the text. See the files in Resources on Blackboard.

1. Start jGrasp.

2. We now want to create a new file and add the program below.

- 1.) Select **File** from **Main Menu**, opens **File Menu**
- 2.) Select **New** from the **File Menu**
- 3.) A popup menu will open with several language choices, select **C File**
- 4.) We want to give the file a name:
 - i.) Select **File** from **Main Menu**
 - ii.) Select **Save As** from the **File Menu**, a **Save As** window will open
 - iii.) If in a lab, select the H: drive. Create a COMP1200 folder to help you organize your files.
 - iv.) In the **Filename** text box, enter **assign1a.c**, then select **Save**

Now you can begin typing in the program.

Where there are CAPS type the requested information. Enter the rest of the program below **exactly as is**, observing the column restrictions and including the comment and blank lines. The program prints a two line message welcoming you to COMP-1200.

```
/* PUT YOUR NAME HERE
COMP-1200
PUT THE SEMESTER AND YEAR HERE
Assignment 1a
program: hello
-----
This a simple program which prints a message to
welcome you to COMP-1200.
-----
*/
#include <stdio.h>
#include <math.h>

int main(void)
{
    printf("Welcome to COMP %d \n", 60 * (int)(sqrt(400.0)) );
    printf("\n"War Eagle"\n");

    return 0;
}
```

Yes, you are to type the
information between `/* */`
and
include the blank lines.

3. You are now ready to **compile and link** the program. First, click on the title bar of **assign1a.c** to make sure it is selected. On the **Main Menu**, select the **Compiler**, then select **Compile & Link**. In the message area at the bottom of the screen, you will see the results of the compile.

Look at the **Results** pane (at the bottom) to see if you have any errors. The most common error for this simple program is forgetting the terminating semicolon or not having matching parentheses or braces. Other errors can be introduced by typing errors. Make sure the program is entered exactly as shown above. When finished **compile and link** the program again. If you still have errors, proof read your program carefully.

4. You are now ready to run the program. From the **Main Menu**, select **Run** then **Run**. The output of the program will be displayed in the lower window.

5. Now let's introduce some errors. Doing the following will not produce something to turn in, but it will give you an idea of how to deal with errors. Everyone will make errors. They can be a teaching tool.

- ☐ 16th line: change **int** to **irnt**
- ☐ 18th line: change (**int**) to just **int** (remove parentheses)
- Note: **int** was previously **blue** but now **black**. The editor is **smart** and can tell that they are now invalid C keywords (this doesn't help us with the 2nd error - it's not that smart!).
- Now compile and link the program again. Note the errors in the **Results** pane.

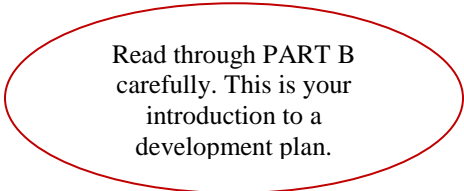
Errors are indicated with the line number of the error between two colons and a short description of the error. If you **left-click** the mouse on the error the cursor will move to the line where the error occurs in the edit pane (sometimes on longer programs you must scroll up one line to make the error appear). Try it. Fix this first error and compile again. Then follow the same procedure for the second error.

PART B:

1. Now it is time for you to design, code and run a larger program. The program we will write will prompt the user for a student's grades, compute and print the GPA, and print a notice if the student is on the Dean's list or the Failure list. A student will be on the Dean's list if his or her GPA is greater than or equal to 3.50. He or she will be on the Failure list if his or her GPA is less than 1.00. We will assume that all classes have the same number of credit hours (4).

To write this program, we will follow the five-step process for problem solving given in your book:

1. State the problem clearly.
2. Describe the input and output information.
3. Work the problem by hand for a simple set of data.
4. Develop a solution that is general in nature.
5. Test the solution with a variety of data sets.



Read through PART B carefully. This is your introduction to a development plan.

2. The first step of the five-step process should come easy in this course, since the problems that are assigned are generally stated clearly to begin with. In general, deciding exactly what the problem is and what you want the computer to do will be relatively difficult.

The problem we are going to solve is to compute a student's GPA and determine if the GPA places the student on the Dean's list or the Failure list.

3. The second step is to describe the input needed and the output given by the program. Like the first step, this step will be easier in this course than in the real world.

For our problem, the I/O description is the following:

Input - the student's grades

Output - the student's GPA and whether the student is on the Dean's list or Failure list.

4. The third step is to work the problem by hand for a simple set of data. Let's try this for a case where the student's grades are A, B, C and C.

Grade	Grade Points	Class
A	4	1
B	3	2
C	2	3
C	2	4

The student was in 4 classes and had a grade point total of 3, so their GPA is $11/4$ or 2.75.

Notice that hand calculation shows that because each class has the same number of credit hours, you do not have to consider credit hours when computing GPA. (Let N be the number of credit hours for each class. For this example, total grade points would be $4N + 3N + 2N + 2N = 11N$. Total hours attempted would be $4N$. So the GPA would be $11N / 4N = 2.75$.) Another way of looking at this is to view each class as being one credit hour.

Let's work one more example by hand, where the student's grades are A, A, and B.

Grade	Grade Points	Class
A	4	1
A	4	2
B	3	3

The student was in 3 classes and had a grade point total of 11, so the GPA is $11/3$ or 3.67. The program will detect that 3.67 is greater than 3.50 and indicate that the student is on the Dean's list.

One more thing might have jumped out at you by now. Not all students take the same number of classes. When our program reads in the student's grades it must have a way of knowing how many classes a student took. There are different ways we could handle this, but let us do it by prompting the user for the number of classes a student took. We will have to do this before prompting the user to input the student's grades. Note this changes the problem and description given in step 11. Our new I/O description will now be:

Input - number of classes taken and the student's grades in the different classes

Output - the student's GPA and whether the student is on the Dean's list or Failure list.

It is quite acceptable to change the I/O description - sometimes you have no other choice. In a professional setting, if other people had already been given or approved your I/O description, you would need to inform these people and be sure that your change did not adversely impact their work.

5. Now that we have a feel for how to solve the problem, we can do the fourth step: develop an algorithm. We will use top-down design meaning that we will start at the top of the problem and break it down into smaller problems that can be solved separately. The first technique of top-down design is problem decomposition - identification of the pieces of the problem that need to be solved sequentially.

DECOMPOSITION

- 1.) Read in the number of classes taken by the student
- 2.) Read student grades and compute grade point total
- 3.) Compute the GPA
- 4.) Print the GPA
- 5.) Print if the student is on the Dean's list
- 6.) Print if the student is on the Failure list

This problem could have been broken down differently. Also, a good program checks for bad input from the user. For simplicity, we will skip checking the input for errors, but as your programming skills improve, you should start considering where programs might go wrong and how you could design the program to recover from an error.

The second technique of top-down design is stepwise refinement - doing each step of the problem decomposition in enough detail that it can be easily converted to computer instructions. Flowcharts, pseudocode, or several other methods of specifying detail can be used to do the stepwise refinement. We will use pseudocode.

STEPWISE REFINEMENT

1.
 - a) **Prompt** the user for the number of classes taken
 - b) **Read** in the *number of classes taken* by the student
2.
 - a) **Prompt** the user to enter the student's grades
 - b) **Set** the *grade point total* to zero
 - c) **Set** the *course counter* to zero
 - d) **While** *course counter* less than *number of classes taken* **do**
Read in a *course grade*
If *grade* = **A** **then**
add 4 to the *grade point total*
Else If *grade* = **B** **then**
add 3 to the *grade point total*
Else If *grade* = **C** **then**
add 2 to the *grade point total*
Else if *grade* = **D** **then**
add 1 to the *grade point total*

Add 1 to the *course counter*
3. **Set** *GPA* to *grade point total* divided by *number of classes taken*
4. **Print** the student's *GPA*

5. **If** *GPA* is greater than or equal 3.5, **then**
 Print that the student is on the Dean's list
6. **If** *GPA* is less than 1.0 **then**
 Print that the student is on the Failure list

Note: items in **bold** are analogous to C statements or standard I/O library functions; items in *italics* are analogous to C variables (refer to program below to convince yourself)

6. Now that the algorithm has been designed, it is time to turn it into C code. Your C lectures have not yet covered all of the computer statements needed to solve this problem. But if you have followed the design process this far, you will be able to follow most of the code below. This is the second program you must type in and run for this assignment.

Now open the edit pane as described in **step A.2** above and enter the program found on the last page of these instructions.

7. Now repeat steps A.3 and A.4 to build and run your program. Use the new filename **assign1b.c** when you get to the Save As window. **This program is correct as it appears. However, you may need to correct any errors caused by typing errors.**

8. Now it is time for the fifth step in the five-step process for problem solving. The fifth step was to test your program with a variety of data sets. Start by testing with the two cases we worked out by hand in step 4. Your program results should match your hand-worked results. If they do not, correct the program and try again. Make up more test cases and test the program until you are sure it is correct. You want to try to test the program on all possible execution patterns.

9. When your program is saved, your work is stored in a directory created for you on the system. When you are logged onto the system, your directory appears to be a disk drive. The name of the drive is H:

10. You can get a printout of your program (make sure the program you want to print is in the edit pane) by selecting **File** from the **Main Menu**, and then **Print**. This will open the **Print Window**, select **OK** and your source file should print to the printer indicated in the printer test box of the window) be aware that the pages you print are counted. The cost for printing in the lab is \$0.06/per page.

11. Read the section **Standards for Documentation of C Programs**. Compare the guidelines given there to the program we developed in this lab.

12. Submit your **assign1a.c** and **assign1b.c** via **Blackboard**.

13. If you are working in a lab, **BE SURE TO LOG OUT** of Microsoft Windows **BEFORE YOU LEAVE THE LAB**. This will log you out of the system. Failure to do so will mean someone else can sit down at the machine and **delete all the files in your directory or print lots of pages which will be charged to you**.

WARNINGS:

- **DO NOT LEAVE A DISK IN THE PC**
- **DO NOT TURN OFF THE PC**
- **EXIT WINDOWS BEFORE LEAVING**

```
/* PUT YOUR NAME HERE
COMP-1200
PUT THE SEMESTER AND YEAR HERE
Assignment 1b
Program Student
```

```
-----
Given the number of classes a student has taken and
the grades that they earned, this program will compute
a student's GPA and indicate whether the student is on
the Dean's list or the Failure list.
```

```
GPA - the student's GPA
GPT - the student's grade point total
Num - the number of classes the student took
Count - the course counter
Grade - the student's grade in one course
-----
```

Yes, you are to type the
information between `/* */`
and
include the blank lines.

```
*/
#include <stdio.h>
int main()
{
    double GPA, GPT;
    int    Num, Count;
    char   Grade;

    /* Read in the number of classes taken by the student */

    printf("Enter the number of classes the student took.\n");
    scanf("%d", &Num);

    /* Read in student grades and compute grade point total */

    printf("Enter student grades, one per line.\n");
    GPT = 0.0;
    Count = 0;
    while (Count < Num)
    {
        scanf("\n%c", &Grade);
        if ((Grade == 'A') || (Grade == 'a'))
            GPT = GPT + 4.0;
        else if ((Grade == 'B') || (Grade == 'b'))
            GPT = GPT + 3.0;
        else if ((Grade == 'C') || (Grade == 'c'))
            GPT = GPT + 2.0;
        else if ((Grade == 'D') || (Grade == 'd'))
            GPT = GPT + 1.0;

        Count = Count + 1;
    }

    GPA = GPT / (double)( Num );           // Compute the GPA

    printf(" GPA is %5.2f\n", GPA);         // Print the GPA

    /* Print if the student is on the Dean's list */
    if ( GPA >= 3.50 ) printf(" Dean's list\n");

    /* Print if the student is on the Failure list */
    if ( GPA < 1.00 ) printf(" Failure list\n");

    return 0;
}
```

Refer to
Step 5 in PART A
and correct your errors.