Assignment 1 (Course Pretest)

C/C++ Programming II

C2A1 General Information

This assignment is a course pretest and covers only concepts you should already be familiar with. If you do not understand these concepts or cannot obtain at least 16 of the possible 20 points (80%) with a reasonable amount of time and effort you probably do not have the knowledge necessary to complete this course successfully. In that case I strongly recommend that you either switch to "C/C++ Programming I" or drop this course and do some catchup studying on your own.

You may drop this course with a refund by the drop deadline, which is usually 5 or 6 days after the course starts – BUT CHECK THE EXTENSION CATALOG TO BE SURE. You may have an additional week to switch to "C/C++ Programming I" if you prefer.

Course Assignment/Exercise Notation Conventions: Each weekly "assignment" consists of several "exercises". Throughout this course I commonly refer to these using an abbreviated notation, where a form like **C2A1E3** would refer to exercise 3 in assignment 1 of the "C/C++ Programming II" course and **C2A1** would refer to the entirety of assignment 1 of that course.

Development Tools and Operating Systems: You may use any development tools and operating systems you want. I recommend Microsoft's "Visual Studio Community" for Windows, "Xcode" for Max OS X, and **"**Code::Blocks" for Linux. Information on obtaining, installing, and using these IDE's is provided in the appropriate version of the course document titled "Using the Compiler's IDE...", a link to which is provided on the "Assignments" page of the course Web site. I'm sorry but I don't have information on other IDE's or operating systems.

Common Restrictions (all course assignments):

1. Do not use inappropriate magic numbers. Avoid them by using macros in C and constant variables in C++.
2. Do not use non-constant external (global) variables.
3. Do not use the `#include` directive to include implementation files (.c or .cpp) in other files.
4. Do not prompt the user for or display anything not called out in the exercise requirements.

Errors/Warnings: If you get run-time errors or compiler errors/warnings about issues in the instructor-supplied "Driver" files, the problem is due to something in your code.

Exercise Submission Procedure: Get an exercise to work first on your computer, then submit it to the "Assignment Checker" and wait for the results to be returned. If there are any errors or warnings make the appropriate corrections and resubmit, repeating as necessary until all issues are corrected. Additional details are provided in each exercise and in the course document titled "Preparing and Submitting Your Assignments".

Lines of Code – This Assignment Only:
You don't need to match or beat this line count!
Next to the names of most implementation files you are required to write I've indicated the number of lines of code I used in the body of the required function in my solution, not including blank lines, comments, or lines only containing braces. I used no coding tricks or non-standard techniques in my solutions. This line count is provided only as a guide in case you might be making your solution more difficult than necessary.
You don't need to match or beat this line count!

C2A1 General Information, continued

In case you don't already know...

Where Does a Program Look for Files When Attempting to Open Them?
Where Does a Program Create New Files?
Where Should You Put Instructor-Supplied Data Files?

What is a "Working Directory"?
A program's "Working Directory" is the directory it uses for any files it opens or creates if their names are specified without a path, and you must place any instructor-supplied data file(s) (.txt or .bin extensions) your program needs in that directory. Its default location differs between IDEs and operating systems and it's important to know where it is and how to change it. For further information please refer to the **Determining/Changing the "Working Directory"** topic in the version of the course document titled "Using the Compiler's IDE..." that is applicable to the IDE you are using.

Opening Files – Testing for Failure/Success

Always check the success/failure status of opening a file before using it or opening another file.

Supplying Information to a Program via its "Command Line"

It is often more appropriate to supply information to a program via "command line arguments" than by user prompts. Such arguments can be provided regardless of how a program is being run, whether it be from within an IDE, a system command window, a GUI icon, or a batch file. For this course I strongly recommend using an IDE for running all programs.

If you are not familiar with using command line arguments first review note 8.3 for information on how to process them within any program, then review the appropriate version of the course document titled "Using the Compiler's IDE...", which illustrates implementing an arbitrary command line in several ways including implementing command arguments containing spaces.

It is important to note that command line redirection information (note 4.2), if any, is only visible to the operating system and will not be among the command line arguments available to the program being run.

Get a Consolidated Assignment 1 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment, send an empty email to the assignment checker with the subject line **C2A1_*ID***, where ***ID*** is your 9-character UCSD student ID. Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

C2A1E0 *(4 points total – 0.4 points per question – No program required)*

Assume language standards compliance and appropriate header file inclusion unless stated otherwise. Testing erroneous or implementation dependent code by running it can be misleading. These are not trick questions and each has only one correct answer. Major applicable course book notes are listed.

1. The data types of:
   47+6.5, 5*e*5**L**, +'5', and ***sizeof****(0.0)* are:
   (Note 2.10)
   A. **int**, **long**, **char**, size_t
   B. **float**, **long**, **signed char**, **unsigned int**
   C. **int**, **long double**, **int**, **double**
   D. **double**, **long double**, **int** or **unsigned**, size_t
   E. *implementation dependent*

2. Predict the output:
   ```
   const long ix;
   for (ix = 7; ix >= 0; --ix)
       cout << ix << ' ';
   ```
   (Note 2.14)
   A. 7 6 5 4 3 2 1 0
   B. 7 6 5 4 3 2 1
   C. 7 7 7 7 7 7 7
   D. *Results are implementation dependent.*
   E. *It won't compile.*

3. For ***int*** *x = 1;* predict the value in *x* after:
   x = x--;
   (Note 3.4)
   A. 1
   B. 0
   C. 3
   D. -1
   E. undefined

4. Predict the output:
   ```
   if (5 < 4)
       if (6 > 5)
       {
           putchar('1');
       }
   else if (4 > 3)
       putchar('3');
   else
       putchar('2');
   putchar('6');
   ```
   (Note 3.15)
   A. 2
   B. 6
   C. 36
   D. 6 *or* 36, *depending upon the implementation*
   E. *Nothing is printed.*

5. Predict the output:
   cout << (12 > 5 ? "Hello " : "World")
   (Note 3.16)
   A. Hello
   B. Hello World
   C. World
   D. World Hello
   E. *implementation dependent*

6. If function calls with no prototypes are permitted, what data types get passed to *MyFunction* by:
   *MyFunction((****short****)23, (****char****)34, 87****L****, 6.8****f****)*
   (Note 5.5)
   A. **int** *or* **unsigned**, **int** *or* **unsigned**, **long**, **double**
   B. **int** *or* **unsigned**, **int** *or* **unsigned**, **long double**, **float**
   C. **short**, **char**, **int**, **float**
   D. **short**, **char**, **long**, **float**
   E. none of the above or *implementation dependent*

7. What is the most serious problem?
   ```
   int *ip;
   for (*ip = 0; *ip < 5;  *ip++)
       ;
   ```
   (Note 5.11)
   A. Nothing is wrong.
   B. It dereferences an uninitialized pointer.
   C. It does nothing useful.
   D. It contains a magic number.
   E. *implementation dependent*

8. If **char**s are 8 bits, **int**s are 40 bits, and
   int **ptr = (****int*** **)40;*
   predict the value of
   --ptr
   (Note 6.14)
   A. 39
   B. 0
   C. 35
   D. 32
   E. *none of the above or implementation dependent*

C2A1E0, continued

9. Predict what will happen:

```
char ch;
while ( (ch = cin.get()) != EOF )
      cout.put(ch);
```

(Note 4.3)

A. A false EOF might be detected or the real EOF might be missed
B. It won't compile
C. cin.get() reads one **int** at a time from input, then its value is printed
D. EOF is not defined in C++
E. Nothing unwanted happens. It simply reads and prints characters until EOF is reached.

10. What is wrong?

```
struct Svalues {char x; int y;} s1 = { 25, 30 };
class Cvalues {char x; int y;} c1 = { 25, 30 };
```

(Note 9.13)

A. Nothing is wrong.
B. Members of the class and the structure have the same names.
C. Public structure members are being accessed by an initializer list.
D. Private class members are being accessed by an initializer list.
E. Public class members are being accessed by an initializer list.

Submitting your solution

Using the format below place your answers in a plain text file named **C2A1E0_Quiz.txt** and send it to the Assignment Checker with the subject line **C2A1E0_*ID***, where ***ID*** is your 9-character UCSD student ID.

-- Place an appropriate "Title Block" here --
1. A
2. C
etc.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E1 *(2 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E1_Macros.h**. Also add instructor-supplied source code file **C2A1E1_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

File **C2A1E1_Macros.h** (I used 6 lines of code but you don't have to match or beat that.) must contain an appropriate "include guard" and the following three macro definitions:

1. a function-like macro named `Product` that has two parameters and whose value is the product of any two arguments of any arithmetic types passed to it.
2. a function-like macro named `Negate` that has one parameter and whose value is the negated value of any arithmetic argument of any type passed to it. For example, if the argument's value is -5, 5 will be produced, or if the argument's value is 5, -5 will be produced. DO NOT use multiplication, division, or subtraction.
3. a function-like macro named `Elements` that has a single parameter and whose value is the count of the number of elements in any 1-dimensional array of any type whose array designator is passed to it.

This file must not contain any code other than that stated above, that is, no `#include` directives, additional macro definitions, function definitions, variable declarations, etc.

Submitting your solution

Send only files **C2A1E1_Macros.h** and **C2A1E1_main-Driver.c** to the Assignment Checker with the subject line **C2A1E1_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E2 *(2 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E2_main.c**.

File **C2A1E2_main.c** (I used 5 lines of code but you don't have to match or beat that.)
must contain the definition of function `main`. It must display a count of the number of command line arguments that were present when the program was started followed by those arguments in their original order starting with `argv[0]`. The count and each argument must be displayed alone on separate lines.

Test your program with various command line arguments, including some containing spaces.

Submitting your solution

Send only file **C2A1E2_main.c** to the Assignment Checker with the subject line **C2A1E2_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E3 *(2 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E3_FindFirstInt.c**. Also add instructor-supplied source code file **C2A1E3_main-Driver.c**. Do not write a main function! main already exists in the instructor-supplied file and it will use the code you write.

File **C2A1E3_FindFirstInt.c** (I used 5 lines of code but you don't have to match or beat that.)
must contain the definition of a function named FindFirstInt that returns type "pointer to **int**" and has a three parameters named ptr, count, and value, left-to-right. They are of type "pointer to constant **int**", **size_t**, and **int**, respectively. FindFirstInt must find the first occurrence of the value represented by value in the array represented by ptr, which has count elements. If the value is found a pointer to that element is returned. Otherwise, a null pointer is returned.

Submitting your solution

Send only files **C2A1E3_FindFirstInt.c** and **C2A1E3_main-Driver.c** to the Assignment Checker with the subject line **C2A1E3_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E4 *(2 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E4_StrToUpper.c**. Also add instructor-supplied source code file **C2A1E4_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

File **C2A1E4_ StrToUpper.c** (I used 4 lines of code but you don't have to match or beat that.)
must contain the definition of a function named `StrToUpper` that returns type **size_t** and has two parameters, where the first is of type "pointer to **char**" and the second is of type "pointer to constant **char**". `StrToUpper` must copy the string represented by its second parameter into the memory represented by its first parameter, with any lowercase characters converted to uppercase. The length of the string, not including its null terminator character, is returned.

Restrictions:

1. Use the `toupper` standard library function to convert from lowercase to uppercase.
2. You may not call any function other than `toupper`.
3. You may only use one variable other than the two parameter variables and it must be of type "pointer to constant **char**".

Submitting your solution

Send only files **C2A1E4_StrToUpper.c** and **C2A1E4_main-Driver.c** to the Assignment Checker with the subject line **C2A1E4_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirement.

C2A1E5 *(2 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E5_ResizeAlloc.c**. Also add instructor-supplied source code file **C2A1E5_main-Driver.c**. Do not write a main function! main already exists in the instructor-supplied file and it will use the code you write.

File **C2A1E5_ResizeAlloc.c** (I used 8 lines of code but you don't have to match or beat that.) must contain the definition of a function named ResizeAlloc that returns type "pointer to **void**" and has three parameters. The first is named pOld and is of type "pointer to **void**" while the second and third are named newSize and oldSize**,** respectively, and are both of type **size_t**.

ResizeAlloc either dynamically allocates an entirely new block of memory containing newSize bytes or, in effect, resizes an existing block in pOld containing oldSize bytes to contain newSize bytes. When resizing occurs all existing data that will fit into newSize bytes will be preserved. ResizeAlloc may not call calloc or realloc or any function or macro that you know does call them.

I recommend implementing the following simple algorithm unless you can devise a better one:

```
If newSize is zero
    return a null pointer.
Else
    Dynamically allocate a new block containing newSize uninitialized bytes.
    If the allocation fails
        return a null pointer.
    Else If pOld is a null pointer
        return a pointer to the new block.
    Else
        If newSize is greater than oldSize
            copy oldSize bytes from pOld to the new block.
        Else
            copy newSize bytes from pOld to the new block.
        Free pOld.
        Return a pointer to the new block.
```

Submitting your solution

Send only files **C2A1E5_ResizeAlloc.c** and **C2A1E5_main-Driver.c** to the Assignment Checker with the subject line **C2A1E5_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E6 *(2 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one named **C2A1E6_AppendFile.cpp**. Also add instructor-supplied source code file **C2A1E6_main-Driver.cpp**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

Place the three instructor-supplied data files **Append_GettysburgAddress.txt**, **Append_3000Nulls.bin**, and **Append_ExpectedResults.bin** in the program's "working directory". DO NOT add these files to your IDE project, send them to the assignment checker, or modify them.

File **C2A1E6_AppendFile.cpp** (I used 21 lines of code but you don't have to match or beat that.)
must contain the definition of a function named `AppendFile` that returns type **int** and has two parameters of type "pointer to constant **char**", where the first is named `inFile` and the second is named `outFile`. Each represents a string that specifies the name of a file, such as **file.c**, **test.txt**, **MyFile**, etc. `AppendFile` must append the contents of the file specified by `inFile` onto the file specified by `outFile`, creating the output file if it doesn't already exist.

`AppendFile` must first open these two files in the binary mode using the minimum access privileges necessary. If an open fails the function must immediately output an error message to `cerr`, close any open files, and return **-1**. Please keep in mind that attempting to close a file that isn't open makes absolutely no sense.

`AppendFile` must:
1. work correctly for both text and binary files;
2. open each file only once;
3. not use any file positioning functions (e.g., **seekg**, **seekp**, **tellg**, **tellp**, etc.);
4. not attempt to read the entire contents of any file into the program at once since in the general case a file can contain more bytes than the largest possible array can hold;
5. close all open files and return **0** after appending completes.

Submitting your solution

Send only files **C2A1E6_AppendFile.cpp** and **C2A1E6_main-Driver.cpp** to the Assignment Checker with the subject line **C2A1E6_*ID***, where ***ID*** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

C2A1E7 *(4 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones named **C2A1E7_Employee.h** and **C2A1E7_Employee.cpp**. Also add instructor-supplied source code file **C2A1E7_main-Driver.cpp**. Do not write a main function! main already exists in the instructor-supplied file and it will use the code you write.

File **C2A1E7_Employee.h** must be protected by an "include guard" and must contain the following in whatever order you deem appropriate:

1. The entire definition of function Employee::Get that returns type **double** and has a single parameter of type "pointer to **double**". It stores the value of member salary in the address pointed to by its parameter and then returns the value of member salary. This function definition must be placed outside the definition of the Employee class itself.
2. The definition of data type class Employee that contains **only** the following:

   Items A-D are **private** data members:

   A. type **"**pointer to **char**" member named name;
   B. type **int** member named age;
   C. type **float** member named raise;
   D. type **double** member named salary;

   Items E-H are **public** member functions, each named Set and each returning type **void**:

   E. the prototype only: has a single parameter of type "pointer to constant **char**";
   F. the entire definition: has a single parameter of type **int** and sets member age to the value of that parameter. This function may optionally be called without an argument, in which case the value of its parameter will be **25**;
   G. the entire definition: has a single parameter of type "reference to constant **float**" and sets member raise to the value of that parameter;
   H. the entire definition: has a single parameter of type "pointer to constant **double**" and sets member salary to the value pointed to by that parameter;

   Items I-L are **public** constant member functions, each named Get:

   I. the entire definition: returns type "pointer to **char**" and has a single parameter of type "pointer to pointer to **char**". It stores the value of member name in the address pointed to by its parameter and then returns the value of member name.
   J. the entire definition: returns type **int** and has a single parameter of type "reference to **int**". It stores the value of member age in its parameter and then returns the value of member age.
   K. the entire definition: returns type "reference to **float"** and has a single parameter of type "reference to **float**". It stores the value of member raise in its parameter and then returns a reference to member raise.
   L. the prototype only - **inline function**: returns type **double** and has a single parameter of type "pointer to **double**".

File **C2A1E7_Employee.cpp**
must contain the definition of function Employee::Set that returns type **void** and has a single parameter of type "pointer to constant **char**", which points to the first character of a C-style string. The function will dynamically allocate exactly the amount of memory necessary to hold that string (including its null terminator), set member name to point to that memory, and copy the string into that memory.

Submitting your solution

Send only files **C2A1E7_Employee.h**, **C2A1E7_Employee.cpp**, and **C2A1E7_main-Driver.cpp** to the Assignment Checker with the subject line **C2A1E7_*ID***, where ***ID*** is your 9-character UCSD student ID.
See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.