Assignment 1 (Course pre-test)

C/C++ Programming II

General Information

Getting Started

Before starting your first assignment you should install and configure your development tools (editors, compilers, etc.) if you have not already done so. Although you are free to use absolutely any tools you wish as long as you can complete the assignments as required, for most students I recommend the free Microsoft "Visual Studio Express 2013 for Windows Desktop" application (http://www.microsoft.com/en-us/download/details.aspx?id=40787) for program development. In my opinion it is one of the most programmer-friendly and reliable integrated development environments (IDEs) available. Regardless of which tools you decide to use, but especially if you decide to use Visual Studio, please refer to the course document titled "Using the Compiler's IDE" for step-by-step configuration details.

About This Assignment Only

The purpose of this assignment, which should mostly be a review of things already learned, is to help students decide if they have sufficient background knowledge and perseverance to take "C/C++ Programming II". It is not intended to exhaustively test programming ability. This assignment:

1. tests knowledge of some basic "C/C++ Programming I" concepts, including some of the "gotchas" and portability issues that permeate C and C++;
2. familiarizes students with the assignment submission requirements and expectations for this course.

You are not expected to be an expert and you may use any reference materials you wish. However, if you are unable to do reasonably well on this assignment it is unlikely you will be able to successfully complete this course and you should consider taking/re-taking "C/C++ Programming I" first. Please contact me with any questions or if you would like to switch to "C/C++ Programming I".

Exercise 1 *(7 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A1E1_main.c**. Write a program in that file that contains all of the items listed below, in order. Please keep your code simple but correct. Note that nothing is required that actually runs/tests your code. However, if you have time and wish to test it feel free to add the test code in your `main` function, although no additional credit will be given for doing this.

1. …Outside of any function, write the definition of a function-like macro named `Sum` that has two parameters, where the first is named `addend1` and the second is named `addend2`. `Sum` must return the sum of any two arithmetic-type arguments passed to it regardless of what those types are.

2. …Outside of any function, write the definition of a function-like macro named `Elements` that has one parameter named `arrayDesig`. `Elements` must return a count of the number of elements in any 1-dimensional array whose array designator is passed to it, regardless of the data type of that array.

3. …Write a function named `CreateArray` that returns type "pointer to **long**" and has a single parameter of type **size_t** named `elementCount`. `CreateArray` must dynamically allocate an array having exactly `elementCount` elements of type **long** in the most efficient way possible, where the initial values of the elements are not important. If dynamic allocation fails the program must display an error message and immediately terminate the program with an error code. Otherwise `CreateArray` must return a pointer to the first element of the array.

4. …Write a function named `OpenFile` that returns type "pointer to **FILE**" and has a single parameter of type "pointer to constant **char**" named `filePath`. `OpenFile` must open the file specified by `filePath` in the "binary read and append" mode, creating it if it doesn't already exist. If the open fails the program must display an error message and immediately terminate the program with an error code. Otherwise `OpenFile` must return the **FILE** pointer for the file.

5. …Write a function named `CopyString` that returns type **void** and has two parameters. The first parameter is named `destination` and is of type "pointer to **char**". The second parameter is named `source` and is of type "pointer to constant **char**". `CopyString` copies the string represented by `source` into the array represented by `destination`. No variables may be used other than the two parameters `source` and `destination` and no other functions or macros may be called.

6. …Write a function named `DisplayClearedArray` that returns type **void** and has no parameters. `DisplayClearedArray` must declare and initialize an automatic array of **double**s named `testArray` to all 0s, all in the same single statement. It must then display the value of each element on a separate line. The array must contain the same number of elements as the number of bytes of storage used for type **long double**. That is, if type **long double** requires 1 byte of storage the array will have 1 element, if type **long double** requires 217 bytes of storage the array will have 217 elements, etc. However, you may make absolutely no assumptions about the actual number of bytes used for type **long double** since this can vary between implementations.

7. …Write a function named `main` that has no parameters, displays the following message, then returns a success code:

```
Assignment 1 Exercise 1 Complete!
```

Submitting your solution

Send your source code file 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.

Exercise 2 (7 points – C++ Program)

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A1E2_main.cpp**. Write a program in that file that contains all of the items listed below, in order. Please keep your code simple but correct. Note that nothing is required that actually runs/tests your code. However, if you have time and wish to test it feel free to add the test code in your `main` function, although no additional credit will be given for doing this.

1. …Outside of any function, write the definition of structure type `PersonInfo` that contains only:
    a. a **private int** member named `age` and a **protected bool** member named `error`;
    b. the entire definition of a **public** constant member function named `GetAge` that has a "pointer to a constant **int**" parameter named `pInt` and returns the type **int** sum of the value of the `age` member and the value pointed to by `pInt`;
    c. the prototype only for a **protected** constant member function named `DoNothing` that returns type **void** and has a "pointer to a pointer to a **double**" parameter named `ppDbl`.

2. …Write a function named `DeclareCppStruct` that has no parameters, declares a variable of type `PersonInfo` (from above) named `person`, and returns a pointer to it. Do not use dynamic memory allocation.

3. …Write three functions, each of which returns type **void**, has the identical name `PrintSomething`, and has a single parameter named `param`:
    a. The first function will be called if an expression of type **double** is passed as an argument or if no argument at all is passed.
    b. The second function will be called if an expression of type **float** is passed as an argument.
    c. The third function will be called if any type of **char *** expression is passed as an argument.

    Each function must display the value of its parameter; the first function will display a value of zero if no argument at all is passed.

4. …Write a function named `TestPrintSomething` that returns type **void**, has no parameters, and calls `PrintSomething` from above 4 times, passing:
    a. No argument;
    b. An argument of type **double** having a value of `23.4`;
    c. An argument of type **float** having a value of `1825e-1`;
    d. An argument of type "pointer to constant **char**" representing the string **`Hello world!`**

5. …Write a function named `DisplayBases` that returns type **long** and has a single parameter of type **double** named `value`. `DisplayBases` must first use a single statement to display the decimal, octal, and hexadecimal representation of the integral part of `value` on separate lines. Then it must use another statement to return the integral part. You may assume that the value of the integral part will not exceed the range of type **long**.

6. …Write a function named `DisplayCharValues` that returns type **void** and has no parameters. It must declare and initialize three constant **char** automatic variables to different alphabetic characters of your choice then display the decimal numeric values of those variables on separate lines.

7. …Write a function named `main` that has no parameters, displays the following message, then returns a success code:

```
Assignment 1 Exercise 2 Complete!
```

Submitting your solution

Send your source code file 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.

Exercise 3 *(6 points total – No program required)*
Full credit will be given for this exercise if it's complete, even if some answers are not correct.

ANSI/ISO language standards compliance is assumed. Testing this code by running it can sometimes be misleading due to implementation dependence. You are not expected to know the answers to all of these questions so don't get discouraged. There is only one correct answer to each.

1. For C compilers that do not permit "//" style commenting syntax, if variables *x*, *y*, and *z* are properly declared, what is *syntactically* wrong with: *z = y//* division */x;*
   (Note 1.4)
   A. Nothing is wrong.
   B. Everything after the // is a comment so the statement is incomplete.
   C. It is not portable.
   D. A comment may not serve as whitespace.
   E. The value of *y* may be too large.

2. The value of ***sizeof**('A')* is always:
   (Note 1.5; Note 2.12)
   A. the same as the value of **sizeof**(**char**).
   B. the same as **sizeof**(**int**) in C and the same as **sizeof**(**char**) in C++.
   C. 65 if the ASCII character set is used.
   D. dependent upon the character set being used.
   E. none of the above.

3. Assuming a 16 bit type **int** and a 32 bit type **long**, what are the data types of:
   32767, *-32768*, *32768*, and *2.0*
   (Note 2.1; Note 2.2)
   A. **int**, **int**, **long**, **float**
   B. **int**, **long**, **long**, **float**
   C. **int**, **long**, **long**, **double**
   D. implementation dependent
   E. none of the above

4. Predict the output:
   cout << oct << 15 << dec << 15 << hex << 15
   (Note 2.6)
   A. oct 15 dec 15 hex 15
   B. 017 15 0xf
   C. 17 15 f
   D. 1715f
   E. *Output is implementation dependent.*

5. The values of *-5/4* and *-5%4* are:
   (Note 2.8)
   A. implementation dependent:
      -5/4 == -1 and *-5%4* == -1 or
      -5/4 == -2 and *-5%4* == 3
   B. -1 and -1
   C. -2 and 3
   D. -1 and -2
   E. none of the above.

6. Assuming ***int*** *ax[123]*, the data types of *+(**char**)65*, *(**short**)47+(**char**)65*, *2*6e2**L***, and ***sizeof***(*ax*) are:
   (Note 2.10)
   A. undefined, undefined, **double**, pointer to **int**
   B. **char**, **short**, **long**, size_t
   C. **int**, **int**, **long double**, pointer to **int**
   D. **int**, **int**, **long double**, size_t
   E. implementation dependent

7. In C, what is the value and data type of the expression *(**char**)25 < (**char**)100*
   (Note 2.10; Note 3.1)
   A. 1 and type **char**
   B. 1 and type **int**
   C. 75 and type **char**
   D. 0 and type **char**
   E. none of the above

8. Predict the output:
   ```
   const int i;
   for (i = 0; i < 5; ++i)
       cout << i << ' ';
   ```
   (Note 2.14)
   A. 0 1 2 3 4
   B. 0 1 2 3 4 5
   C. 1 2 3 4 5
   D. *It won't compile.*
   E. *Output is implementation dependent.*

9. Predict the output:
   printf("Goodbye") && printf(" Cruel") || printf(" World")
   (Note 3.2)
   A. Goodbye
   B. Goodbye Cruel
   C. Goodbye Cruel World
   D. Goodbye World
   E. *Output is implementation dependent.*

10. For ***int*** *x = 1;* predict the value in x after:
    ```
    x = ++x
    ```
    (Note 3.4)
    A. 1
    B. 2
    C. 3
    D. undefined
    E. implementation dependent

11. Predict final value of *i*:
    ```
    for (int i = 0; i < 5; ++i)
        break;
    ```
    (Note 3.10)
    A. 0
    B. 1
    C. 2
    D. 3
    E. none of the above

12. Predict the value in x after:
    ```
    auto int x =
      (4, printf("Hello"), sqrt(64.), printf("World"));
    ```
    (Note 3.11)
    A. 4
    B. 5
    C. 6
    D. 8
    E. implementation dependent

13. Predict the output:
    ```
    if (5 < 4)
        if (6 > 5)
                putchar('1');
    else if (4 > 3)
        putchar('2');
    else
        putchar('3');
    putchar('4');
    ```
    (Note 3.15)
    A. 4
    B. 2
    C. 24
    D. 4 *or* 24*, depending upon the implementation*
    E. *Nothing is printed.*

14. Predict the output:
    ```
    cout << (12 < 5 ? "Hello " : "World")
    ```
    (Note 3.16)
    A. Hello
    B. Hello World
    C. World
    D. World Hello
    E. *Output is undefined or implementation dependent.*

15. 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.

16. Predict the output:
    ```
    cout << __DATE__ << __FILE__ << __LINE__
    ```
    (Note D.5)
    A. *Today's date, the name of the executable file, and the number of lines in the file*
    B. *The compilation date, the source file name, and the source file line number containing __LINE__*
    C. *cout cannot use these macros without typecasts*
    D. *Output is implementation dependent.*
    E. *It won't compile.*

17. In C with no prototype, what data types get passed to *fcn* by the call:
    *fcn((**char**)23, (**short**)34, 87, 6.8**f**)*
    (Note 5.5)
    A. **char**, **short**, **int**, **float**
    B. **char**, **short**, **long**, **float**
    C. **int** *or* **unsigned**, **int**, **int**, **float**
    D. **int** *or* **unsigned**, **int**, **int**, **double**
    E. none of the above or implementation dependent

18. In C, what is the most serious problem?

```
long double fx(void)
{
    double answer = sum(1.1, 2.2, 3.3);
    return printf("answer = %f", answer);
}
double sum(double a, double b, double c)
{
    return(a + b + c);
}
```

(Note 5.4)

A. The name *sum* conflicts with a standard ANSI math function.
B. The **return** statement in *fx* returns type **double**.
C. Return statements may not contain an algebraic expression (a + b + c).
D. There is nothing wrong with the program.
E. The call to *sum* assumes that *sum* returns type **int**.

19. In C++, predict the output:

```
void print(int x = 1, int y = 2, int z = 3)
{
    cout << x << y << z;
}
int main()
{
    print(), print(4), print(5, 6), print(7, 8, 9);
    return(EXIT_SUCCESS);
}
```

(Note 5.7)

A. 123
B. 456789
C. 123456789
D. 123423563789
E. *It won't compile.*

20. In C++, predict the output:

```
void print(int x, int y = 2, int z = 3) { cout <<
    x << y << z; }
void print(long x, int y = 5, int z = 6) { cout
    << x << y << z; }
int main()
{
    print(4), print(4L);
    return(EXIT_SUCCESS);
}
```

(Note 5.8)

A. 44**L**
B. 423423
C. 423456
D. *Output is implementation dependent.*
E. *It won't compile because the print function definitions are ambiguous.*

21. 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. It contains implementation dependent problem(s).

22. Assuming

#define sum(a, b) a + b

predict the value of:

*5 * sum(3 + 1, 2)*

(Note 5.18)

A. 30
B. 18
C. 22
D. *none of the above*
E. *implementation dependent*

23. In C++, predict the output. (Assume a pointer is printed as a standard integer.)

```
  void print(int &x, int y, int *z)
  {
      x = 10;
      y = 20;
      z = (int *)30;
  }
  int main()
  {
      int a = 1, b = 2, *c = (int *)3;
      print(a, b, c);
      cout << a << b << c;
      return(EXIT_SUCCESS);
  }
```

(Note 6.9)

A. 123
B. 102*value* (where *value* represents some representation of decimal 30)
C. 102*value* (where *value* represents some representation of hex 30)
D. 102*value* (where *value* represents an unpredictable (garbage) value)
E. 102*value* (where *value* represents some representation of 3)

24. If a prototype for *fx* (below) is present, predict the output from: *printf("%d", *fx())*

```
    int *fx(void)
    {
        int x = 5;
        return(&x);
    }
```

(Note 6.12)

A. 5
B. *garbage*
C. *the address of the variable x*
D. *A compiler error occurs.*
E. *none of the above or implementation dependent*

25. If **char**s are 8 bits, **int**s are 16 bits, and

***int** *ip = (**int** *)20;*

predict the value of

++ip

(Note 6.14)

A. 20
B. 21
C. 22
D. 23
E. *none of the above or implementation dependent*

26. What is the most serious problem?

```
    int ip[] = {6, 7, 2, 4, -5};
    for (int i = 0; i < 5; ++i, ++ip)
        cout << *ip;
```

(Note 6.17)

A. Nothing is wrong.
B. An uninitialized pointer is being dereferenced.
C. An attempt is being made to modify the name of an array, a constant.
D. It contains a magic number, which is illegal in some compilers.
E. An out of bounds array access occurs.

27. What is wrong with the following string initialization?

```
    char s[] = {'H', 'E', 'L', 'L', 'O', NULL};
```

(Note 7.1)

A. Nothing is wrong.
B. The syntax is incorrect.
C. A character array can't hold a string.
D. *NULL* may be of the wrong data type.
E. Strings can't be initialized.

28. Assuming prototypes and **typedef**s are present for the following C code, what is wrong with it either programmatically or from a good programming practice point of view?

```
    char *cp = malloc(256);
    FILE *fp = fopen("hello", "a+");
    fprintf(fp, "Message\n");
    cp[0] = 'A';
```

(Note 8.4; Note 10.3)

A. Nothing is wrong.
B. The syntax is incorrect.
C. *cp* is not an array so the form *cp[0]* is not valid.
D. *malloc* and *fopen* are not portable.
E. All file opens and dynamic allocations must be checked before being used.

29. For

***typedef struct** {**char** x; **int** y;} FOO;*
FOO bar;

which of the following may be false?

(Note 9.12)

A. **sizeof**(FOO) == **sizeof**(bar)
B. **sizeof**(FOO) == **sizeof**(bar.x) + **sizeof**(bar.y)
C. *&bar* is numerically equal to *&bar.x*
D. *(**char** *)&bar + offsetof(FOO, y) == (**char** *)&bar.y*
E. they can all be false, depending upon implementation

30. 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. Members of the class and the structure have the same names.
    B. Public members of a structure are being accessed by an initializer list.
    C. Private members of a structure are being accessed by other than a member/friend function.
    D. Private members of a class are being accessed by other than a member/friend function.
    E. Nothing is wrong.

31. A file must never be opened in the text mode if:

    (Note 10.2)

    A. it will be used for binary data.
    B. *fprintf* or *cout* will be used to write to the file.
    C. the data to be written contains the newline character.
    D. the C++ *fstream* functions are to be used.
    E. compatibility with modern compilers is desired.

32. What is wrong?

    ```
    ofstream fout("file1");
    ifstream fin("file2");
    fstream fio("file3");
    ```

    (Note 10.4)

    A. The syntax is incorrect for all 3 opens.
    B. "file1" may not exist
    C. *ifstream, ofstream*, and *fstream* opens require two arguments.
    D. *fstream* opens require two arguments
    E. It's not portable

The following three questions address topics that are not covered in "C/C++ Programming I", but will be covered in "C/C++ Programming II". Give them your best "guess".

33. On a machine using 1's complement negative integers and 16 bit **int**s, what is the bit pattern for -2?

    (Note 11.1)

    A. 1111 1111 1111 1111
    B. 1111 1111 1111 1110
    C. 1111 1111 1111 1101
    D. 1000 0000 0000 0010
    E. implementation dependent

34. If an **int** is 16 bits and a **char** is 8 bits, the values in *sch* and *uch* after

    signed char *sch = 256;* and
    unsigned char *uch = 256;*

    are:

    (Note 11.2)

    A. *sch* is 256 and *uch* is 256
    B. *sch* is implementation defined and *uch* is 256
    C. *sch* is implementation defined and *uch* is 0
    D. *sch* is 0 and *uch* is 0
    E. The results of both are undefined.

35. Assuming a 16 bit **int** 2's complement implementation, predict the value of:

    -17 >> 1

    (Note 11.7)

    A. -9 or 0x7FF7, depending upon the implementation
    B. -8
    C. 17
    D. 8
    E. other implementation dependent values

Submitting your solution

Using the format below place your answers in a plain text file named **C2A1E3_Quiz.txt** and send it to the Assignment Checker with the subject line **C2A1E3_*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.

Get a Consolidated Assignment 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.