COMP-370
Software Design with OOP/C++
Homeworks

Instructions: All homeworks for this course are to be contained within a 3-tab paper folder with the following information clearly marked on front: Your name, COMP-370, Sr. Professor Wheeler, Spring 2000. The homeworks will be placed in reverse order within the folder so that the latest homework is visible upon opening the folder. *All work is to be typed or word-processed. Use complete sentences when answering questions.*

Homework 1: A Review of C Fundamentals

1. What are the objects in the C language?

2. Draw a block diagram illustrating each step that takes place when a C program is compiled.

3. Define the following terms using a complete sentence for each:

   (a) Source code
   (b) Assembly language
   (c) Object code
   (d) Library
   (e) Machine language

4. List the fundamental C object types in the order of least to most precise. For each one, give the type of information that it is normally used to store.

5. What is the type "`void`" normally reserved for in C?

6. What is a function *prototype*? Explain why they are needed in C.

7. What is the difference in action between the following INCLUDE statements:

   a) #include <MyFirst.h>
   b) #include "MyFirst.h"

8. At what function do all C programs begin execution at?

9. What is a C "expression?"

10. For each of the following C expressions, give the resulting (final) data type and result. Use a tabular format similar to what is presented here.

C Expression	Data Type	Result
`7`		
`25.`		
`'A'`		
`7+3`		
`'A' + 1`		
`22/7`		
`22/7.`		
`22 / ((double) 7)`		
`0x1f`		
`023`		
`7/3 + 1.5`		
`(7*3 - 2 ) / 10.`		
`(float) 3`		
`(int) (22. / 7. )`		
`22. / ((int) 7)`		

11. Describe the purpose of the following C operators:

(a) `++` (b) `--` (c) `=` (d) `==` (e) `>` (f) `<`
(g) `!` (h) `!=` (i) `&&` (j) `||` (k) `~`

12. For the following PRINTF format specifiers, give the data type expected and field-width. If applicable, give the number of decimals to the right of the decimal point.

Format Specifier	Data Type Expected	Width and Decimal Precision
`%d`		
`%x`		
`%c`		
`%ld`		
`%f`		
`%lf`		
`%10.2f`		
`%3c`		
`%s`		
`%20.2lf`		

Homework 2: More C Fundamentals

1. What is a C *function*?

2. What is meant when we state that C is a "one-pass" compiler?

3. What is the default data type for C and C++ functions?

4. How should a C function that returns *no* data (and therefore doesn't appear on the right-hand side of an equals sign) be declared?

5. What are the *parameters* of a function? When calling a function, does C automatically convert function arguments to the correct data type?

6. What is a *prototype*? Why are prototypes needed in C?

7. A *global variable* is visible to what parts of a C program? What are the initial contents of a global variable? How long does a global variable persist (exist)?

8. What parts of a C program have access to a *local variable*? What are the initial contents of a local variable? When does a local variable begin to exist, and when is it destroyed?

9. Write prototypes for the following short C functions:

```
// Function 1
void SetModifiedFlag( int nFlag )
{
// Set the global document-modified flag according to the argument.
if (nFlag) doc_mod = 1;
           else
           doc_mod = 0;
}

// Function 2
float PI(void)
{
// Return an approximation of the constant "PI"
return(3.1415927);
}

// Function 3
void Goodbye(void)
{
int i;
for(i=0;i<10;i++)
  printf("\nGoodbye, world!\n");
}
```

10. Explain what is wrong with the following C function. How could the error be fixed?

```
// Function 4, bad coding (compiler will flag an error on
// this one)

void Funct4( float value1 )
{
float temp;

temp = value1 * value1 + 1;  // Compute x^2 + 1

return(temp);
}
```

11. Give the returned data type of each of the prototyped functions below:

a) int GetProfileInt(char* szKey);
b) double sin(double* Angle);
c) SquareIt(float Value);
d) char* GetAppTitle(void);
e) HKEY AddRegistryKey(HKEY hkRoot , char* szKeyLabel);
f) BOOL CreateShellAssociation(char* szExtension , char* szOpenCommandLine);

Note: *HKEY* and *BOOL* are defined data types in the examples above.

Homework 3: Pointers, References, and Overloading

1. What is a C *pointer*?

2. What is meant by the phrase "dereferencing a pointer?"

3. How are the arguments being passed to the function below? Can the function modify the caller's original variables that have placed their data into *value1* and *value2*? Why or why not?

```
void MyFunction( int value1, float value2)
{
// ... function code here ..
}
```

4. How are the arguments being passed to the function below? What are the advantages and disadvantages of this method of passing parameters?

```
void NewFunction( int* value1, float* value2)
{
// ... function code here ..
}
```

5. How are the arguments being passed to the function below? What is *really* happening when arguments are passed this way?

```
void MyFunction( int& value1, float& value2)
{
// ... function code here ..
}
```

6. When using arguments that have been passed by reference, can the programmer modify the original value(s)? Why or why not?

7. Write a program statement that properly calls the function of question 4.

8. Write a program statement that properly calls the function of question 5.

9. Write a version of *NewFunction()* [according to the convention of question 4] that takes the sum of the contents of *value1* and *value2* and stores the result into the variable stored at *value2.* Note that the arguments are passed by pointer.

10. Write a version of *NewFunction()* [according to the convention of question 5] that takes the sum of the contents of *value1* and *value2* and stores the result into the variable stored at *value2.* These arguments are passed by reference.

11. A program has the following variables and declarations:

```
int nCount=5;
int nIncrement=6;

float Result;

int* pPtr1;
int* pPtr2;
float* pPtr3;

pPtr1 = &nCount;
pPtr2 = &nIncrement;
pPtr3 = &Result;

*pPtr3 = ( *pPtr1 + *pPtr2 ) * 3.1415927;

(*pPtr2)++;

*pPtr1 = 25;
```

What will be the final values in *nCount, nIncrement,* and *Result*?

12. Given the declarations of problem 11, fill in the information in the table below. Some expressions may be *invalid.*

C Expression	Data Type
`pPtr1`	
`*pPtr1`	
`pPtr3`	
`&nIncrement`	
`*pPtr3 / *pPtr2`	
`*pPtr2 - 25`	
`pPtr2`	
`*Result`	

13. What is *Hungarian notation*? Why is it used in C programming? Give at least two examples.

14. Given the function prototypes below, determine which version of the function will be called by each C statement:

```
1) int MultiplyNumbers( int Value1, int Value2 );
2) double MultiplyNumbers( double Value1, double Value2 = 1.0 );
3) double MultiplyNumbers( double Value1, int Value2);


  // Calling Statements

int nResult, nV1, nV2;
double Result, V1, V2;   // common for all four calling statements


a) nResult = MultiplyNumbers( nV1, nV2);
b) Result = MultiplyNumbers( V1, 2.0 );
c) Result = MultiplyNumbers( V1 );
d) Result = MultiplyNumbers( 3.14159, 2 );
```

Homework 4: Structures and Classes

1. What is a *structure?*

2. What is a *class*? List three differences between *classes* and *structures*.

3. List and explain the steps involved in defining and allocating memory for a structure or class.

4. What is meant by the phrase "Instantiating an object?"

5. How is the "." operator used with structures and classes?

6. For the structure below, write a `main()` procedure that does the following to each member of the structure array:

   (a) Initializes the "m_nNum" of the member with a unique numeric value.

   (b) Initializes the "m_szString" member with the phrase "Testing xx" where "xx" is the same numeric value loaded into "m_nNum." (Use `sprintf()` to do this).

   (c) Print out all members of the array in a neat format to demonstrate that each is holding correct data.

   Print out your `main()` function and a sample run and include it in your homework.

```
struct MyStruct     {
                    int m_nNum;              // serial number
                    char m_szString[40];     // Item name
                    } Database[100];
```

7. Convert the printing code in your `main()` function from problem 5 into a function called `printstruct()` that takes one instance of the structure and prints it to the screen. Print out and comment this function, and make the necessary changes to `main()` so that the program runs properly. The prototype for the function **must** be as follows:

   ```
   void PrintStruct( struct MyStruct aStruct );
   ```

   Print out your new `main()` function, the printstruct() function, and a sample run.

8. What are the disadvantages of passing structures by value as the function of problem 6 does?

9. Convert the function of problem 6 so that it uses a pointer to the structure, instead of a copy. Print out and comment this function, and make the necessary changes to `main() again` so that the program runs properly. The prototype for the function **must** be as follows:

```
void PrintPointerStruct( struct MyStruct* aStruct );
```

10. In a class, what are the *methods*?

11. Define the following terms as they are used in object-oriented programming. Give an example for each one.

a) Abstraction
b) Encapsulation
b) Polymorphism
c) Inheritance

12. Implement the data structure of problem 6 as a class and supply the method functions for the class. Test the operation of the class with the supplied `main()`.

```
class CMyStuff {
               public:

               void SetSerial(int nSerNum);
               int GetSerial(void);
               void SetString(char* szString);
               char* GetString(void);
               void PrintContents(void);

               private:

               int m_nNum;
               char m_szString[40];
               };
```

Explanation of methods:

Method Prototype	Purpose
`void SetSerial(int nSerNum);`	Sets the member m_nNum to the argument *nSerNum*.
`int GetSerial(void);`	Returns the value held in m_nNum.
`void SetString(char* szString);`	Copies the user-supplied string into the internal m_szString[] buffer. Use *strcpy()*.
`char* GetString(void);`	Returns a pointer to the internal m_szString[] buffer.
`void PrintContents(void);`	Prints the object's contents in a neat format.

The `main()` code for testing the class is on the next page.

```
#include <stdio.h>
#include <math.h>
#include "mystuff.h"     // your class defined in mystuff.h

void main(void)
{
CMyStuff Array[100];     // set up 100 CMyStuff objects.
int i;                   // loop counter
char temp[128];          // temporary buffer for generating
                         // formatted character strings

// Initialize the objects

for(i=0;i<100;i++)
  {
  sprintf(temp, "Object # %d", i+1 );

  Array[i].SetString(temp);
  Array[i].SetSerial(i);
  }

// Test the objects by asking them to tell us what we stored in
// them. Compared the returned values with what we originally put
// in.

for(i=0;i<100;i++)
  {
  sprintf(temp, "Object # %d", i+1 );

  if ( Array[i].GetSerial() != i)
          {
          printf("Element %d failed to return serial number.\n",i);
          break;
          }

  if ( strcmp( Array[i].GetString() , temp ) )
          {
          printf("Element %d failed to return proper string.\n",i);
          break;
          }

  // Contents were OK, print 'em out

  Array[i].PrintContents();
  }

if (i>=100)
  printf("\nCongratulations, all your MYSTUFF objects worked great!");

}
```

Homework 5: Constructors and Destructors

1. What is a *constructor?* How is a constructor function named? (Give an example.) When is a constructor called?

2. What is a *destructor*? How is a destructor function named? (Give an example.) When is a destructor called?

3. What two types of activities usually take place in a class constructor? List and explain each one.

4. What is the purpose of the `new` operator? Give an example of its use in the following ways: (a) allocating space for a single `int` type, and (b) allocating space for an array of 1000 `double` types.

5. What primary activity usually takes place in a class destructor? Explain what will happen if this activity does not take place.

6. Given the class definition:

```
class CInductor
{
public:

void SetInductance(double Val);
void SetFrequency(double w);
double GetInductance();
double GetInductiveReactance();

private:

double m_Inductance;
double m_RadianFrequency;
};
```

Write a default constructor that will set the inductor's value to 1 Henry (stored in `m_Inductance`) and the operating frequency to 0 r/s [DC], (stored in `m_RadianFrequency`).

7. Write a constructor for the class of problem 6 with a prototype as follows:

```
CInductor( double RadianFrequency, double InductorValue);
```

This constructor will set the internal members m_Inductance and m_RadianFrequency with the values passed in the two constructor parameters.

8. Write a copy constructor for the class of problem 6 that will properly copy all members to the newly-constructed object.

9. Write the code to overload the assignment operator *`operator=()`* so that when objects of the class of problem 6 are copied by assignment, all members are properly copied.

Homework 6: Inheritance

1. What is *inheritance*? Explain why it is needed in C++.

2. Define the terms *parent class*, *base class*, *child class*, and *derived class.*

3. What are the two elements that are *not* inherited by derived classes in C++?

4. Unless the `public` keyword is included as an access modifier, what is the default access attribute for inherited class members?

5. Given the following base class definition, state the access attribute for each member of the derived class `CNewClass`:

```
CBaseClass
{
public:
int m_x,m_y,m_z;     // 3D coordinates

protected:
double m_CashFlow;

private:
float m_InLake;
};

CNewClass : public CBaseClass
{
public:
double m_Gamma;
};
```

6. Given the base class definition of problem 5, state the access attribute for each member of the derived class `CNewClass`:

```
CNewClass : private CBaseClass
{
public:
double m_Gamma;
};
```

7. Given the base class definition of problem 5, state the access attribute for each member of the derived class `CNewClass`:

```
CNewClass : protected CBaseClass
{
public:
double m_Gamma;
};
```

8. An object of type CNewClass has been instantiated using the default constructor, as follows:

   CNewClass aObj;

   Explain the sequence of events that will take place to construct this object. Specifically, what constructors are called, and in what order? Assume that default constructors exist for both base and derived classes.

9. Explain the sequence of events that will take place when the object of question 8 goes out of scope (is destructed). What specific destructor(s) are called, and in what order?

10. If a base class called *CBase* has a constructor prototyped as follows:

```
CBase( int Value1 , double Value2 );
```

   Write a constructor for a derived class `CDerivedFromBase` that will call the base class constructor, passing *Value1* and *Value2* from the caller who is constructing the `CDerivedFromBase` object.