You may take this test with you afterwards, but you must turn in your answer sheet.

This test has the following sections:

I. True/False .................................. 60 points; (30 questions, 2 points each)
II. Multiple Choice...................... 40 points; (8 questions, 5 points each)

--------------
100 points total

This test is worth 10% of your final grade. You must put your answers on the bubble form. This test is open book and open notes, but no computers. For the multiple choice problems, select the best answer for each one and select the appropriate letter on your answer sheet. Be careful - more than one answer may seem to be correct. Some questions are tricky.

**True/False: (1 point each)** On your bubble form fill out A for true and B for false.

**T F 1.** C++ is a superset of C.

**T F 2.** The return type of `main()` is supposed to be int in C++.

**T F 3.** While a C program would use a `scanf` statement to read input from the keyboard into a variable, a C++ program could use either the `scanf` statement or the `cin` statement.

**T F 4.** To read in two values into two variables, two separate `scanf` statements are needed.

**T F 5.** A single `printf` statement in a program always generates a single line of output.

**T F 6.** A `char` value returned from a function can be stored into an `int` variable.

**T F 7.** Using `%d` in the format specifier for a `scanf` statement is used for double-precision floating point numbers.

**T F 8.** A `switch-case` statement can always be rewritten with multiple `if-else` statements.

**T F 9.** Multiple `if-else` statements can always be equivalently rewritten using a `switch-case` statement.

**T F 10.** In the code shown below, the comparison `y==5` will never be done:

```
int x=2, y=5;
if ( (x >= 3) && (y == 5)) {
    printf("Yes ");
}
```

**T F 11.** Any code written in C using one of the three looping structures (while, do, for) can be equivalently rewritten using any of the other looping structures with the possible addition of a few other lines of code.

**T F 12.** A `do` loop is the best type of loop to use to display a menu and prompt for user input, as compared to a `for` loop or a `while` loop.
13. Generally an if-else statement can be rewritten using two if statements.

14. Indentation helps us as humans understand programs in C, but does not in any way change the meaning of a program as far as the compiler is concerned.

15. Two different functions in C can have the same name, as long as the types or number of parameters are different.

16. A function with a return type of void can still have a return statement in it, but it just can’t return a value.

17. Recall that the ASCII table shows that ‘A’ is stored as 65, ‘a’ is stored as 97, and the space character ‘ ’ is stored as 32. The code below would print out: B a d

   ```c
   printf("%c %c %c", 65, 97, 100);
   ```

18. Miller’s Magic Number is the principle that on average people can remember a maximum of 10 unrelated items.

19. The statements below store the input value into variable x:

   ```c
   int x;
   scanf("%d", age);
   ```

20. The statements below store the input value into variable c:

   ```c
   char c;
   cin >> c;
   ```

21. Reference parameters in C in the calling part of the code should be passed with an asterisk.

22. Reference parameters in C++ in the function declaration should be caught with an asterisk.

23. Reference parameters are implemented in C by passing the address and then dereferenced it when it is used.

24. Changes to a value parameter are not reflected back to the calling part of the code because changes are made to a local copy of the original value.

25. The following code automatically figures out the length of the array and allocates enough memory for it.

   ```c
   int numbers[] = {1,2,3,4,5,6,7};
   ```

26. The following code allocates space for the 3 characters and in addition automatically adds a space for a NULL character which is automatically added to the end of the array.

   ```c
   char word1[] = "ABC";
   ```

27. The following code allocates space for the 3 characters and in addition automatically adds a space for a NULL character which is automatically added to the end of the array.

   ```c
   char word2[] = {'C','A','T'};
   ```
The following code will display the text: First letter is: D

```c
char letters[] = "DFR";
printf("First letter is: %c", letters[ 1]);
```

The section of C++ code shown below would compile and run and give as output: Is 0 Done

```c
int z = 0;
if( z = 1)
    printf("Is 1 ");
else
    printf("Is 0 ");
printf("Done \n");
```

The following code segment gives the following output: Yo! Dude

```c
char c='Y';
switch (c){
    case 'Y': printf("Y");
    case 'o': printf("o");
    case '!' : printf("!");
    break;
}
printf( " Dude");
```

Multiple Choice (5 points each)

31) This question refers to the point made with the video shown in class. How do you eat a bicycle?
   a) Dissolve in acid, clean, eat
   b) Start from one end and methodically work your way to the other end
   c) Cut in half, then cut each remaining piece in half again, and so on
   d) One small piece at a time
   e) None of the above

32) Consider a program where an array is passed as a parameter to a function. Upon returning from the function call, assume you notice that another variable’s value has now changed, even though it is declared locally within main() and not passed to the function. What is the best explanation for this?
   a) Your computer has been hacked. Update your anti-virus, reboot and recompile your code.
   b) There are hidden instructions past the right hand edge of your screen that are making the change.
   c) There are non-printable characters in your code that are modifying the instructions.
   d) You are overwriting the end of the array within the function.
   e) None of the above
33) What is the output from the following C program?

```
#include <stdio.h>
int x=1;

void f1( int x)
{
    x++;  
}

void f2( int y)
{
    x++; 
}

int main()
{
    int x=1;
    f1( x);
    f2( x);
    printf("x is: %d \n", x);
    return 0;
}
```

a) x is: 1  
b) x is: 2  
c) x is: 3  
d) x is: 4  
e) None of the above

34) Consider the code shown below:

```
void changeTwo( int a, int b)
{
    int temp = a++;  
    a = b++;  
    b = temp; 
}

void testChangeTwo()
{
    int x=3; 
    int y=5; 
    changeTwo( x, y); 
    printf("x and y are: %d %d\n", x, y);  
}
```

What is the result of calling function testChangeTwo()?

a) x and y are: 3 5  
b) x and y are: 5 3  
c) x and y are: 4 6  
d) x and y are: 6 4  
e) None of the above
35) What is the best description of what function \texttt{f1()} shown below does? Consider the test case illustrated by calling \texttt{f2()}. 

```c
char * f1( char theWord[], char c) 
{
    int x = 0; 
    int count = 0; 
    for( int i=strlen( theWord)-1; i>=0; i--) {
        if( theWord[ i] == c) {
            count++; 
            if( count > 1) {
                return &theWord[ i];
            }
        }
    }
    return theWord; 
}
```

a) Returns the index of the last \texttt{charToFind} 

b) Returns the index of the next-to-last \texttt{charToFind} 

c) Returns the address of the last \texttt{charToFind} 

d) Returns the address of the next-to-last \texttt{charToFind} 

e) None of the above

36) What is the best description of what function \texttt{f3()} shown below does? Consider the test case of calling it with: \texttt{f3("bookkeeper")}; 

```c
void f3( char text[])
{
    int lastIndex = strlen( text) - 1;
    int count = 0;
    for( char *pWord = text; pWord < &text[lastIndex]; pWord++) {
        if( *pWord == *(pWord+1) ) {
            printf("%c", *pWord);
        }
    }
}
```

a) Displays each character in \texttt{text} 

b) Displays every-other character in \texttt{text} 

c) Displays every character in \texttt{text} except the last one 

d) Displays the address of every other character in \texttt{text} except the last one 

e) Displays every character in \texttt{text} that is followed by an identical character
37) Consider the code shown below. Do a trace of the programming all the way through. Assuming that parameter size contains the size of the array, what ends up in array letters?

```c++
void modify(char letters[], int size)
{
    for(int i=0; i<size; i++) {
        // assume swap swaps the letters at the two positions
        swap(letters, i, size-i-1);
    }
}
```

a) The letters from parameter letters in reverse order
b) The letters from parameter letters in their original order
c) The letters from parameter letters in reverse order when the length of letters is odd
d) The letters from parameter letters in reverse order when the length of letters is even
e) None of the above

38) What is the output of the following C++ program?

```c++
#include <iostream>
using namespace std;

int x=2, y=3;

void confuse1(int &y, int x)
{
    x++;
    y++;
}

void confuse2(int b, int *x)
{
    y = ++(*x);
    *x = b;
}

void confuse3(int &a, int *x)
{
    a = *x + 1;
    (*x)++;
}

int main()
{
    int x=1;
    confuse1( x, y);
    confuse2( x, &y);
    confuse3( x, &y);
    printf("x + y = %d \n", x+y);
    return 0;
}
```

a) x + y = 3
b) x + y = 6
c) x + y = 9
d) x + y = 11
e) None of the above