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

This test has the following sections:
   I. True/False .......................... 40 points; (20 questions, 2 points each)
   II. Multiple Choice ................ 60 points; (12 questions, 5 points each)

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

This test is worth 15% of your final grade. You must put your answers on the bubble form. This test is open book and open notes. 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. When a problem describes a segment or fragment of code you may assume the rest of the program is correct and would be supplied to make it work.

I. True/False: (2 points each) On your bubble form fill out A for true and B for false.

   T   F  1. In the code shown below, the comparison $y==5$ will never be done:

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

   T   F  2. Two different functions in C can have the same name, as long as the types or number of parameters are different.

   T   F  3. The code below will print out: A

   ```c
   int x=65;
   printf("%c", x);
   ```

   T   F  4. The code below will print out: 3

   ```c
   char c='3';
   printf("%d", c-'0');
   ```

   T   F  5. In a scanf statement the format specifier describes the format of the input. We can use %d for double-precision floating point numbers.

   T   F  6. Most C programs can run in a C++ compiler.

   T   F  7. Most C++ programs can run in a C compiler.

   T   F  8. At most 10 guesses are needed when using Binary Search on a list of 1,000 elements.

   T   F  9. The name of an array is a constant address of the first element.

   T   F  10. & is the dereference operator in C
T  F  11. * is the address operator in C

T  F  12. The code below will compile and run:

```c
int numbers[]={1,3,4};
for( int i=0; i<5; i++) {
   printf("%d", numbers[ i]);
}
```

T  F  13. If two strings are not equal in C, then the `strcmp` function always returns a non-zero positive number.

T  F  14. When we use ++ to increment a pointer variable in C, under some conditions we add 1 to it, but other times we add 2 or 4.

For the next three problems consider the following declaration:

```
struct Person {
   char name[ 25];
   int age;
};

Person p1 = {"Erin", 23};
Person *pPerson = &p1;
```

T  F  15. Would the following code compile and run and allow storing first name and age properly?

```c
printf("Enter first name and age: ");
scanf("%s %d", p1.name, *pPerson.age);
```

T  F  16. Would the following code compile and run and allow storing first name and age properly?

```c
printf("Enter first name and age: ");
scanf("%s %d", p1.name, &(*pPerson).age);
```

T  F  17. Would the following code compile and run and allow storing first name and age properly?

```c
printf("Enter first name and age: ");
scanf("%s %d", p1.name, &pPerson->age);
```
Assume the following function is declared for the following 3 problems:

```c
int * add( int newNumber, int *pArray, int &size)
{
    int *pNewArray = new int[ size + 1];

    for( int i=0; i<size; i++) {
        pNewArray[ i] = pArray[ i];
    }

    pNewArray[ size] = newNumber;
    delete( pArray);

    size++;
    return pNewArray;
}
```

**T F 18.** Function `add(...)` shown above could be called using the following code and would run correctly:

```c
int numbers[] = {1,3,5};
int size = 3;
add( 7, numbers, size);
```

**T F 19.** Function `add(...)` shown above could be called using the following code:

```c
int size = 3;
int *pNumbers = new int( size);
pNumbers[0]=1; pNumbers[1]=3; pNumbers[2]=5;
pNumbers = add( 7, pNumbers, size);
```

**T F 20.** Function `add(...)` shown above is equivalent to function `add2(...)` shown below when called as follows:

```c
void add2( int newNumber, int * *pArray, int *size)
{
    int *pNewArray = new int[ *size + 1];

    for( int i=0; i<*size; i++) {
        pNewArray[ i] = (*pArray)[ i];
    }

    pNewArray[ *size] = newNumber;
    delete( *pArray);

    *pArray = pNewArray;
    (*size)++;
}
```

```c
int size = 3;
int *pNumbers = new int( size);
pNumbers[0]=1; pNumbers[1]=3; pNumbers[2]=5;
add2( 7, &pNumbers, &size);
```
II. Multiple Choice (6 points each)

21. What is the output from the following code when calling function `pointerIncrementProblem()`?

```c
void foo( int *pArray)
{
    *(pArray++);
    (*pArray)++;
}

void pointerIncrementProblem()
{
    int theArray[] = {1,3,5,7,8,9,10};
    foo( theArray);
    cout << "Pointer increment " << theArray[0] << " "
    << theArray[1] << " " << theArray[2] << endl ;
}
```

a) Pointer increment 1 3 5  
b) Pointer increment 2 3 5  
c) Pointer increment 1 4 5  
d) Pointer increment 1 3 6

22. What is the output from the following C program?

```c
#include <stdio.h>

int x=2;

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

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

int main()
{
    int x=2;
    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
23. What is the result of calling function testChangeTwo() below, as part of a C++ program?

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

a) x and y are: 3 5  
b) x and y are: 5 3  
c) x and y are: 3 3  
d) x and y are: 5 5  
e) None of the above

24. When a 2-D array is passed to a function, in the function declaration the size of the first parameter may be left blank, but the second dimension must be supplied. Why is this?

a) The total size of the array divided by the size given in the second dimension is used internally to calculate the number for the first dimension, so the user need not supply it.  
b) The size of the first dimension is always automatically supplied, even when not specified by the user, since it is included as part of the definition of every array.  
c) Code in C can overwrite the end of an array, however for a 2-D array the formula used to find the i
th row needs to know how many columns are on each row.  
d) A NULL character is always inserted at the end of every row so that the compiler can tell where one row ends and the next begins, so the size of the first dimension is not necessary.  
e) None of the above

25. Consider the function shown at right below.  For positive numbers, how would you best describe the return value of calling function first?

```cpp
int first( int x, int n)
{   if( n==0)
        return 0;
    else if( n>0)
        return first( x, n-1) + x;
        return 0;
}
```

a) x + n  
b) x * x  
c) x * n  
d) x  
e) None of the above

26. Consider the function shown at right below.  When f5() is called and input is abc! then what is the output?

```cpp
void f5() {
    char c;
    cin >> c;
    if (c != '!' ) {
        f5();
        cout << c;
    }
}
```
27. Consider the code segment shown below. If after the function call the value of `number` has changed, what is the most likely cause?

```c
int number = 5;
printf(" %d", number);
char letters[]="ABCD"
changeLetters( letters);
printf(" %d", number);
```

- a) `number` is a global variable instead of a local variable as it should be
- b) Although `number` is not passed to function `changeLetters()`, function `changeLetters()` itself calls a second function which changes `number`
- c) Function `changeLetters()` overwrites the end of array `letters`
- d) There is some ASCII control characters that are present in the code even though they are not visible
- e) None of the above

28. What is the output of the C++ program shown at right below?

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

int x=3, y=1;

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 = 2;
    confuse1( x, y);
    confuse2( x, &y);
    confuse3( x, &y);
    printf("x + y = %d \n", x+y);
    return 0;
}
```

- a) `x + y = 2`
- b) `x + y = 4`
- c) `x + y = 8`
- d) `x + y = 10`
- e) None of the above
29. What is the result of calling function `arrayPointerToBraces()` in the code below?

```c
void displayUsingBraces( int theArray[], int size)
{
    printf("Using braces Array is: ");
    for( int i=0; i<size; i++) {
        printf("%d ", theArray[ i]);
    }
    printf("\n");
}

void arrayPointerToBraces()
{
    int *pArray = NULL;
    int size = 3;
    pArray = (int *) malloc( sizeof( int) * size);

    // store some initial values into array
    int *pTemp = pArray;
    *pTemp = 9;  pTemp++;
    *pTemp = 11; pTemp++;
    *pTemp = 4;  pTemp++;
    displayUsingBraces( pArray, size);
}
```

a) Contents of the dynamically created array are printed out  
b) Compiler error in function `arrayPointerToBraces()`  
c) Compiler error in function `displayUsingBraces()`  
d) Run-time error

30. What is the result of calling function `arrayBracesToPointer()` in the code below?

```c
void displayUsingPointer( int *pArray, int size)
{
    printf("Using pointers Array is: ");
    for( int i=0; i<size; i++) {
        printf("%d ", *pArray);
        pArray++;
    }
    printf("\n");
}

void arrayBracesToPointer()
{
    const int size = 3;
    int theArray[ size];

    // store some initial values into array
    theArray[ 0] = 9;  theArray[ 1] = 11;  theArray[ 2] = 4;
    displayUsingPointer( theArray, size);
}
```

a) Contents of the array are printed out  
b) Compiler error in function `arrayBracesToPointer()`  
c) Compiler error in function `displayUsingPointer()`  
d) Run-time error
31. What is the result of calling function `arrayPointerChange()` in the code below?

```c
void displayUsingPointer( int *pArray, int size)
{
    printf("Using pointers Array is: ");
    for( int i=0; i<size; i++) {
        printf("%d ", *pArray);
        pArray++;
    }
    printf("\n");
}

void arrayPointerChange()
{
    int *pArray = NULL;
    int size = 3;
    pArray = (int *) malloc( sizeof( int) * size);

    // store some initial values into array
    int *pTemp = pArray;
    *pTemp = 9;  pTemp++;
    *pTemp = 11; pTemp++;
    *pTemp = 4;  pTemp++;

displayUsingPointer( pArray, size);

    printf( "pArray now points to %d\n", *pArray);
}
```

a) Contents of the array are printed out and `pArray` points to the beginning of the array  
b) Contents of the array are printed out and `pArray` points past the end of the array  
c) Compiler error in function `arrayPointerChange()`  
d) Compiler error in function `displayUsingPointer()`  
e) Run-time error

32. Consider the code shown below. Assuming that parameter `size` contains the size of the array, what ends up in array `letters`?

```c
void change( char letters[], int size)
{
    for( int i=0; i<size/2; i++) {
        char temp = letters[i];
        letters[ i] = letters[ size - i - 1];
        letters[ size - i - 1] = temp;
    }
}
```

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