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 ........................................... 58 points; (29 questions, 2 points each)
  II. Multiple Choice.................. 42 points; (7 questions, 6 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 reset 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. A function that returns a char value can be used in a program fragment where the return value is stored instead into an int variable, such as in the code below. This will compile and run.

```c
char problem1()
{  
   return 'a';
}

int main()
{  
    int x;
    x = problem1();
    // rest of program...
}
```

T  F  2. In a scanf statement the format specifier describes the format of the input. For instance using %c would be used for a character input. We can use %f for small floating point numbers and %d for double-precision floating point numbers.

T  F  3. A continue statement with the addition of setting a variable and checking its value could be used to do the same thing that a break statement does.

T  F  4. Most C programs can run in a C++ compiler.

T  F  5. Most C++ programs can run in a C compiler.

T  F  6. Consider two sorting approaches, call them approach A and approach B. A can be faster than B on average at sorting items into ascending order, but could be slower than B when the data is in worst-case descending order to start.

T  F  7. At most 4 guesses are needed when using Binary Search on a list of 11 elements.
TF 8. An array that is defined as type
   char theArray[ 5];
could be passed to function foo and in the function declaration be caught as:
   foo(char *pArray)

TF 9. The name of an array is constant address of the first element.

TF 10. The output of the code segment below is: Problem 10. 6

```cpp
void foo10( int theArray[])
{
    theArray[ 2]++;
}

void Problem9()
{
    int theArray[ ] = {1,3,5,7};
    foo10( theArray);
}
```

TF 11. The following code segment will compile and run:

```cpp
void foo11( int theArray[])
{
    for( int i=0; i<10; i++ ) {
        theArray[ i] = 7;
    }
}

void Problem11()
{
    int theArray[ ] = {1,3,5,7,8,9,10};
    foo11( theArray);
    cout << "Problem 11. " << endl;
}
```

TF 12. The output from the following code segment is: Problem 12. 6

```cpp
void foo12( int *pArray)
{
    pArray = pArray + 2;
    (*pArray)++;
}

void Problem12()
{
    int theArray[ ] = {1,3,5,7,8,9,10};
    foo12( theArray);
}
```

TF 13. & is the address operator in C

TF 14. * is the dereference operator in C

TF 15. In C and C++ it is easy to overwrite the end of an array.

TF 16. All C-strings must be terminated by a NULL character.
T F 17. If two strings are not equal in C, then the `strcmp` function always returns a non-zero positive number.

T F 18. The `gets` function reads a whole line into a string.

T F 19. The `fgets` function reads a whole line into a string.

T F 20. The `fgets` function is different from the `gets` function because it allows formatted reading of an input line.

T F 21. 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.

T F 22. Assume we had the following declarations in a C++ program:

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

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

Would the following code compile and run and allow storing first name and age properly?

```cpp
printf("Enter first name and age: ");
scanf("%s %d", pl.name, &pPerson->age);
```

T F 23. Assume that we have the following function declaration:

```cpp
void Problem23()
{
    static int x=1;
    cout << "x is: " << x << "\n";
    x++;
}
```

The output after calling this function twice in a row:

```cpp
Problem23();
Problem23();
```

is the following:

```
x is: 1
x is: 1
```
Assume the following function is declared for the following 3 problems:

```c
void 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);
    pArray = pNewArray;
    size++;
}
```

**T F 24.** Function `add(…)` shown above could be called using the following code:

```c
t

```

**T F 25.** 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;
add( 7, pNumbers, size);
```

**T F 26.** Function `add(…)` shown above is equivalent to function `add2(…)` shown below when called as follows:

```c
int size = 3;
int *pNumbers = new int( size);
pNumbers[0]=1; pNumbers[1]=3; pNumbers[2]=5;
add2( 7, &pNumbers, &size);
```

**T F 27.** A struct can be nested inside another struct.

**T F 28.** Recursion is when a function directly or indirectly calls itself.

**T F 29.** In a C program with an array declared using square brackets [ ], the size of the array can be a variable.
II. Multiple Choice (6 points each)

30. What is the output from the following segment of code?

```c
char text[] = "All generalizations are false";
char first[ 81];
char second[ 81];
char *pCurrent = text;
char *pSpace;

pSpace = strrchr(pCurrent, ' ');
strcpy( first, pSpace);
*pSpace = '.';
pCurrent = strrchr(pCurrent, ' ');
*pSpace = NULL;
strcpy( second, pCurrent);

cout << first << " " << second << endl;
```

a) All generalizations
b) generalizations All
c) are false
d) false are
e) None of the above

31. 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) Code in C can overwrite the end of an array, however for a 2-D array the formula used to find the ith row needs to know how many columns are on each row.
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) 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.
d) 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.
e) None of the above

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

```c
void changeUp( char letters[], int size)
{
    for( int i=0; i<size; 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
33. What is the output from the following segment of code if function `Problem3()` is called?

```c
char * doStuff( char words[]) { 
    static char *pWords = words; 
    char *pNext = strchr( pWords, ' '); 
    *pNext = '\0'; 
    char * returnValue = pWords; 
    pWords = pNext + 1; 
    return returnValue; 
} 

void Problem3() { 
    char words[] = "yabba dabba do "; 
    cout << doStuff( words) << " "; 
    cout << doStuff( words) << " " << endl; 
}
```

(a) yabba  
(b) yabba yabba  
(c) yabba dabba  
(d) yabba dabba do dabba do  
(e) None of the above

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

```c
double first( int x, int y) { 
    if( y==0) 
        return 0.0; 
    else if( n>0) 
        return first( x, n-1) + x; 
}
```

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

35. 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  
(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
36. What is the output of the following C++ program?

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

int x=2, y=6;

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

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

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

int main()
{
    int x=4;
    confuse1( x, y);
    confuse2( &y, x);
    confuse3( x, &y);
    printf("x + y = %d \n", x+y);
    return 0;
}
```

a) x + y = 14  
b) x + y = 16  
c) x + y = 17  
d) x + y = 19  
e) None of the above