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..............................46 points; (46 questions, 1 point each)
   II. Multiple Choice......................52 points; (13 questions, 4 points each)
-----------------------------------------
         98 points total + 2 points for name = 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.  Assume that after creating a program called stuff.c and compiling it successfully from the command line on a UNIX system, typing in a.out gives a “not found” error in UNIX. To solve this you could instead type ./a.out

T  F   4.  The following UNIX command will compile the stuff.c program and create an executable called stuff:
            gcc -o stuff stuff.c

T  F   5.  The following UNIX command gives a directory listing:
            list

T  F   6.  As seen in demos in class using DevC++, when running a program the output window flashes and disappears. To solve this problem you could either use the system("pause") command or you could have your program read in a single char variable such as in the code:
            char c;
            scanf(" %c", c);

T  F   7.  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   8.  To read in two values into two variables, two separate scanf statements are needed.

T  F   9.  A single printf statement in a program always generates a single line of output.

T  F   10.  A function that returns a char value can be used in a situation where the return value is
stored instead into an int variable.

T  F  11. 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  12. A switch-case statement can always be rewritten with multiple if-else statements.

T  F  13. In the code shown below, the comparison y==5 will never be done:

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

T  F  14. The break and continue statements are constrained forms of a GOTO statement.

T  F  15. 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  16. A break statement with the addition of setting a variable and checking its value could be used to do the same thing that a continue statement does.

T  F  17. If a function returns an int, then the return value must be stored into an int variable and cannot simply be ignored.

T  F  18. A do loop is the best type of loop to use to display a menu and prompt for user input.

T  F  19. Any code that tests multiple values of a single variable using multiple if-else-if statements in C can be written using a switch-case statement instead.

T  F  20. 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  21. Almost every if-else statement can be rewritten using two if statements.

T  F  22. 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.

T  F  23. Two different functions in C can have the same name, as long as the types or number of parameters are different.

T  F  24. Parameters in C++ can have default values.

T  F  25. Consider the method used in class to count the total number of students, where in each "round" pairs of students combine their numbers and one of the students in each pair sits down. To count 40 students we would need 6 rounds.

T  F  26. A function with a return type of void can still have a return statement in it.

T  F  27. The system("pause"); command works on windows but not in UNIX.

T  F  28. Selecting the number of decimal places to be displayed to the right of the decimal point is possible in C, but not in C++. C++ programs always use the C method to do this.
T  F   29. 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 values below would form the letters: Oh Yes 79 104 32 87 101 115

T  F   30. Miller's Magic Number is the principle that on average people can remember a maximum of 10 unrelated items.

T  F   31. A `scanf` statement that reads in a character can skip leading space simply by including a leading space inside the parenthesis of the format specifier.

T  F   32. The following two statements are equivalent:
\[ \text{cin} \gg \text{age}; \]
\[ \text{scanf}("%d", \text{age}); \]

T  F   33. Given the declaration
\[ \text{char letter;} \]
the input statement in C++
\[ \text{cin} \gg \text{letter}; \]
would handle input in the same way as the following input statement in C:
\[ \text{scanf}("%c", \text{letter}); \]

T  F   34. The general problem-solving steps recommended by George Polya are OPIR, standing for: Outline steps, Plan algorithm needed, Implement plan, and Review.

T  F   35. In a program functions can be written after `main()` only if they have a function declaration before `main()`.

T  F   36. Reference parameters in C in the calling part of the code should be passed with an asterisk.

T  F   37. Reference parameters in C++ in the function declaration should be caught with an asterisk.

T  F   38. Reference parameters are implemented by passing the address and then dereferenced it when it is used.

T  F   39. 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.

T  F   40. The following code automatically figures out the length of the array and allocates enough memory for it.
\[ \text{char numbers[]} = {1,2,3,4,5,6,7}; \]

T  F   41. 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.
\[ \text{char letters[1][] = {"ABC"};} \]
**T F 42.** 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 letters2[] = {'A','B','C'};
```

**T F 43.** The following code will display the text: **It is:** A

```c
char letters[] = {"Able was I ere I saw Elba"};
printf("It is: %c", letters[0]);
```

**T F 44.** The section of C++ code shown below would compile and run and give as output:

```
int z = 0;
if( z = 1)
    printf("No ");
else
    printf("Yes ");
printf("Done \n");
```

**T F 45.** 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");
```

**T F 46.** The output of the following code is: **value is equal.**

```c
int value = 5;
printf("value is ");
if( value <= 6)
    if( value == 6) {
        printf("equal");
    }
else
    printf("Larger than ");
printf(".");
```
Multiple Choice (4 points each)

47) Think back to the activity we did on the first day of class, where we discussed different ways to count the number of students in the class. One method we experienced together was to have everyone stand, where each person represented the number 1. Then simultaneously people compared numbers, where one sat and the one remaining standing represented the sum of the two of them, and so on, until a single student remained standing. What was the point of this exercise?

a) There are many equivalent ways to accomplish the same purpose.
b) Some algorithms are better than others at scaling up to larger sizes of data
c) A precise set of instructions is called a computer algorithm
d) Computers require unambiguous instructions
e) None of the above

48) Think back to “Mustard & Jelly” sandwich example, where groups of students were asked to create a precise set of instructions to make a “Mustard & Jelly” sandwich. What was the point of this exercise?

a) There are many equivalent ways to accomplish the same purpose.
b) Some algorithms are better than others at scaling up to larger sizes of data
c) A precise set of instructions is called a computer algorithm
d) Computers require unambiguous instructions
e) None of the above

49) Consider function *getInput* shown below along with the program segment that calls it. What would be the result of running this segment of code?

```c
void getInput( int *theInput)
{
    printf("Enter an integer: ");
    scanf("%d", theInput);
}

//… other code ...
int number = 3
getInput( &number);
printf("%d ", number);
```

a) The number that was entered by the user would be displayed
b) The original value of variable number (3) would be displayed
c) The program would crash when reaching the scanf command because there is no ampersand
d) The program will run, but the value displayed is indeterminate because the scanf command is missing the ampersand
e) None of the above
50) 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

51) Consider code used to read in 100 test scores and find the minimum, average and maximum scores. The preferable type of loop to use for this would be a:

a) `while` loop
b) `do-while` loop
c) `for` loop
d) Either a `while` loop or a `do-while` loop
e) It doesn’t make any difference

52) Consider code used to display a menu and read in and handle the user input. The preferable type of loop to use for this would be a:

a) `while` loop
b) `do-while` loop
c) `for` loop
d) Either a `while` loop or a `do-while` loop
e) It doesn’t make any difference

53) Consider function `multiplicationTable` shown below, which would be part of a larger C++ program. What is the best description of its output?

```c
void multiplicationTable()
{
    for( int row=1; row<=10; row++) {
        for( int col=1; col<=10; col++) {
            printf("%4d", row*col);
        }
        printf("\n");
    }
}
```

a) A multiplication table displayed in rows and columns with at least one blank space between each entry on each row, however without all the columns lined up neatly.
b) A multiplication table displayed in rows and columns with at least one blank space between each entry on each row, with all the columns lined up neatly.
c) A multiplication table displayed in rows and columns with at least one blank space between each entry on each row except for one entry, with all the columns lined up neatly.
d) A multiplication table displayed in rows and columns with more than one instance of adjacent values on rows not having spaces between them.
e) None of the above
54) Function *declarations* can be made separately from function *definitions*. Function declarations must always have:

a) return type and name of function  
b) return type, name of function and types of parameters  
c) return type, name of function, types of parameters and names of parameters  
d) return type, name of function, types of parameters, names of parameters and default values  
e) None of the above

55) 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) You are overwriting the end of the array within the function.  
c) There are hidden instructions past the right hand edge of your screen that are making the change.  
d) There are non-printable characters in your code that are modifying the instructions.  
e) None of the above

56) Consider function *looping1* shown below. For positive numbers, how would you best describe its return value?

```c
int looping1( int a, int b)
{
    int answer = 1;
    int x = 0;

    while( true) {
        if( x < b) {
            answer *= a;
            x++;
        } 
        else {
            break;
        }
    }

    return answer;
}
```

a) a * a  
b) a + b  
c) a * b  
d) a^b  
e) None of the above
57) What is the output from the following C++ code segment?

```cpp
char values[] = "ABC";
int sum = 0;
for (int i=0; i<3; i++) {
    sum = sum + values[i] - 'A';
}
sum = sum / 3;
printf("%d", sum);
```

- a) 3
- b) 198
- c) 66
- d) 67
- e) None of the above

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

```cpp
void changeUp( 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 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
59) What is the output of the following C++ program?

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

int x=3, y=5;

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;  
    (*x)++; 
}

int main()
{
    int x=4;

    confuse1( x, y);  
    confuse2( &x, y);  
    confuse3( x, &y);  
    printf("x + y = %d \n", x+y);  

    return 0;
}
```

```none
a) x + y = 8  
b) x + y = 9  
c) x + y = 10  
d) x + y = 11  
e) None of the above
```