The exam format will be different from the online quizzes. It will be written on the test paper with questions similar to those shown on the following pages. The exam will be closed book, but students can use one 8.5"x11" sheet of notes, in any size type or hand-written, front and back of the page.

Study recommendations for the Mid-Term Exam

Review Chapters 4 and 5 in your textbook.

Review the following web pages about:

Chapter 4 section of the Glossary at: http://www.gibsonr.com/classes/cop2000/glossary.html#CHAP4
Chapter 4 section of the Examples at: http://www.gibsonr.com/classes/cop2000/examples.html#C4
Chapter 5 section of the Glossary at: http://www.gibsonr.com/classes/cop2000/glossary.html#CHAP5
Chapter 5 section of the Examples at: http://www.gibsonr.com/classes/cop2000/examples.html#C5

Review the revised Order of Precedence Table (including arithmetic, relational and Boolean operators):

```
()                                                  Highest / First
postfix ++    postfix --                                   ^
!    prefix ++    prefix --                                |
unary +   unary -   unary &   casts                        |
*   /   %                                                  |
+   -                                                      |
<   >   <=   >=                                           |
==  !=                                                     |
&&                                                         |
||                                                  Lowest / Last
```

Remember the Truth Tables for Boolean Operators:

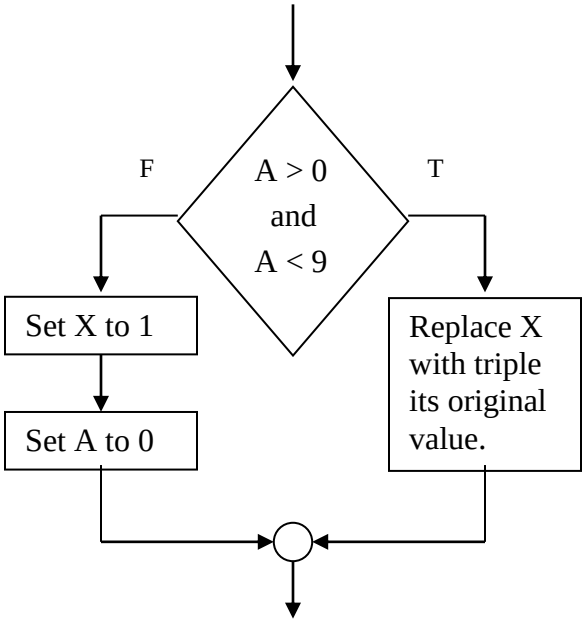
(Op = Operand, a relational expression or Boolean value)

Op.A	Op.B	Op.A **&&** Op.B
F	F	F
F	T	F
T	F	F
T	T	T

Op.A	Op.B	Op.A **||** Op.B
F	F	F
F	T	T
T	F	T
T	T	T

Op.A	**!** (Op.A)
F	T
T	F

SELECTION STRUCTURES:



Structures such as the one on the left are coded in C++ using the reserved word **`if`**.

The *diamond* shape encloses the *condition* that is being tested (in this example, a *Boolean expression*). Remember that the **`if`** statement can only execute one statement on each *leg* (T or F), so the pair of steps on the False leg must be coded inside of a **`{`** and **`}`** pair to make them a single *compound statement*. Thus the C++ code would be:

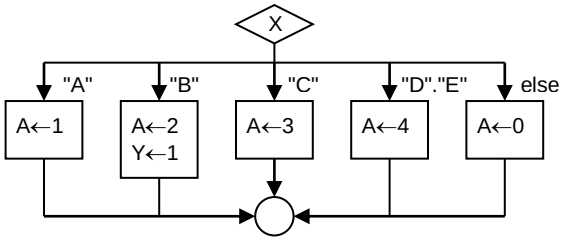
```
if (A>0 && A<9)
  X=X*3;
else
  {
    X=1;
    A=0;
  }
```

Note the use of the parentheses around the condition in the code above. Additional parentheses are often necessary because of the *order of precedence* of operators in C++ (see the table on the previous page).

Remember also that the false leg may be empty, but not the true leg. If your flowchart shows the opposite, then reverse the labeling of the True and False legs and reverse the logic of the condition inside the diamond. If you can't determine what the logical opposite of the condition would be, simply use the NOT operator like this: **`!`** **`(`** `A>0 && A<9` **`)`**

CASE Selection

The C++ **`switch`** statement can be used in the special situation that you are testing for many possible values in a single ordinal storage location. In the example below, five possible paths might be followed depending on the value stored in the character storage location X. Each path (leg) is selected based on the individual value(s) that might be stored in X. The C++ code would be:




```
switch (X)
{
case 'A': A=1; break;
case 'B': A=2; Y=1; break;
case 'C': A=3; break;
case 'D':
case 'E': A=4; break;
default: A=0; break;
}
```

Notice:

- Paretheses are required around the character expression (X).
- A "break;" statement is required to end each each case option (including the default if it is not the last option). Note that the case and break statements act like braces enclosing the steps on each leg and allowing multiple statements without the need for braces around them.
- The order of the case options (including the default statement) is not important, but all options must be mutually exclusive.
- Braces are required to enclose the entire list of case options.

The use of indentation and the positions of the carriage returns in the code are irrelevant.

REPETITION STRUCTURES:

Structure:

We studied two repetition structures (loops); one known as **leading decision (a.k.a. pretest)** and another known as **trailing decision (a.k.a. posttest)**. The primary difference between them is in where the test that controls the loop is performed. **Leading decision** loops test before each pass and are coded in C++ using

```
while (condition) { body_statements; }
```

Trailing decision loops test after each pass and are coded in C++ using

```
do { body_statements; } while (condition);
```

We can design a loop that test somewhere in the middle, but that is consider very poor programming practice.

Control:

We studied two methods of controlling loop passage and exit. **Counting control** is based on the testing of values that have been set and altered by the programmer. Programmers know in advance how many times any counting loop will pass (execute its body) because all values of its *control variable* are predictable. **Sentinel control** is based on the testing of external values that have been read into the program. Programmers do not know in advance how many times a sentinel loop will pass (execute its body) because the values of its *control variable* are unpredictable.

Boundary Values:

The values of variables just prior to the entry into a loop and just after the exit from a loop are called boundary values. Steps inside a loop can be organized so that the value of a variable is changed after it is displayed, so don't expect that variables will always contain the last value that you saw displayed.

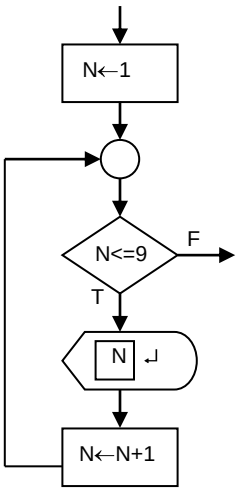
Counting Loops:

All counting loops have at least four parts:

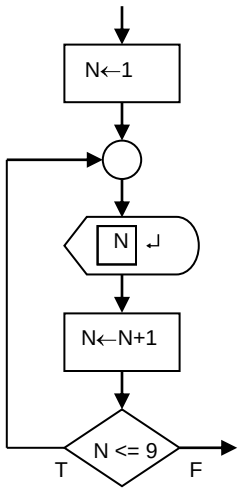
- **initialization** - prior to the loop entry, a first value of the control variable is set
- **body** - where the step(s) to be repeated belong
- **increment** (or **decrement**) - where the control variable is increased (or decreased)
- **test** - where the control variable is tested to determine whether the loop should pass or exit.

The bottom three steps are not always in the order listed above. The order of the steps will effect the boundary values of the control variable.

The two examples below show loops that will count from 1 to 9 and display the value of the control variable (also called a **counter** in a counting loop). Notice the **exit value** on N will be 10.




```
N=1;
while (N<=9)
  {
   cout <<N<<endl;
   N = N+1;
  }
```




```
N=1;
do
  {
   cout <<N<<endl;
   N = N+1;
  }
while (N<=9);
```

If a loop: (1) uses the Leading Decision structure, (2) uses counting control, and (3) increments as the *last* step of its pass, C++ offers a special "automatic loop" statement combining the statements above (on the left) into the single statement: `for (N=1; N<=9; N=N+1) cout << N << endl;`
Note: if the body of the loop has more than one statement to repeat, block it inside of braces {}.

SAMPLE QUESTIONS: (AND ANSWERS)

Circle any SYNTAX (not run-time) ERRORS (there may be more than one) in each of the following C++ code segments. If a segment is valid answer "OK". Assume all variables have been correctly declared.

```
cin >> A;
if ( A<10 && >20 )
cout << "ok");
```

>20 is not a complete relational expression

```
for (I=1, I<=79, I++)
   cout << "-";
```

the commas `(,)` should be semi-colons `(;)`

```
X=1; cin >> Y;
if X > Y cout << "OK"
else cout << "NO";
```

missing parentheses around condition **and** missing semi-colon after if statement

Indicate the Boolean results (TRUE or FALSE) for each of the following expressions, given:

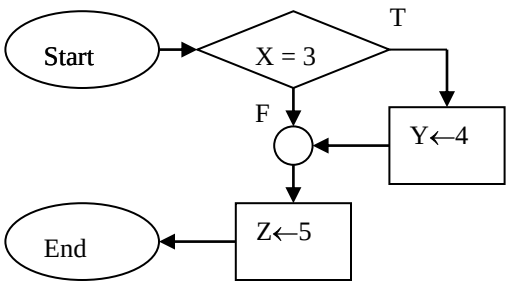
```
#define S "123"  /* a string of digits */
int A=0, B=2, C=-3;
```

`A==0 || B<2 && C<0`	True
`!(A>C) && A<C`	False
`A!=0 || B<2`	False
`S>"8"`	False (strings sort like the phonebook)
`S>100`	Logically invalid (data type mismatch)

Draw a flowchart that matches the following code.

```
{
  if (X == 3)
     Y = 4;
     Z = 5;
}
```

Answer:



Notice that the `Z = 5` step follows the selection. It is not part of the true leg because it was not included inside of a { … } pair of braces. The semicolon following the `Y = 4` statement terminated the if statement.

SAMPLE QUESTIONS: (AND ANSWERS)

Given the following declarations for employee data within a program:

```
int  A; /* Age */
char D; /* Highest Degree Earned (only possible values are uppercase:
         'N' = None, 'B' = Bachelor's, 'M' = Master's, or 'D' = Ph.D.) */
char G; /* Gender (only uppercase 'M' or 'F' are possible) */
```

Write a compound `if` statement in C++ which will display the message "OK" only when the employee is a *female*, between the ages of *20 and 30 (inclusive)* and holds a college degree (*Bachelor's, Master's or Ph.D.*). Do NOT check for case. Assume that previous statements have guaranteed uppercase values only, so that you need only check for uppercase values.

```
if ( G=='F' && A>=20 && A<=30 && ( D=='B' || D=='M' || D=='D' ) )
   cout << "OK";
```

Use the following `for` Statement to answer the next 3 questions (A-C).

```
for (Count=Start; Count<=Finish; Count++) cout << Count;
```

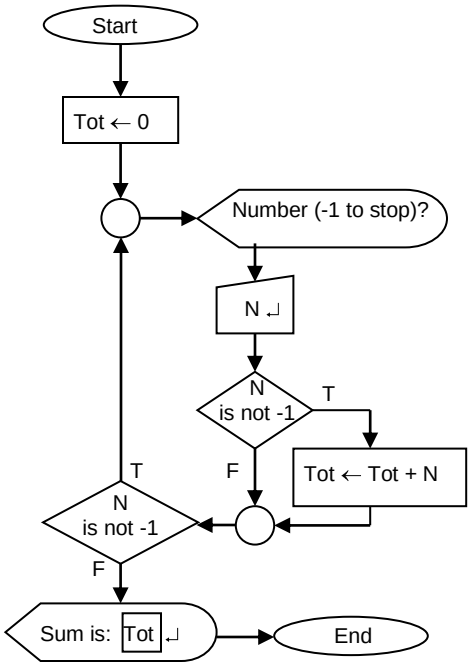
(A) Which variable is the "control variable"? Count

(B) What does `Count++` mean? Increment Count by 1 after a pass is complete

(C) If `Start` contains a value of -4 and `Finish` contains the value 0, how many times will the loop execute?

Answer: 5

Flowchart a program with a loop that accumulates whole numbers entered by the user until a `-1` is entered and then identifies and displays the sum. Then write the C++ code to the right of it.




```
/* accumulate.cpp */

#include <iostream>
using namespace std;

int main ()
{
 int N, Tot=0;

 do
  {
   cout << "Number (-1 to stop)? ");
   cin >> N;
   if (N != -1) Tot = Tot + N;
  }
 while (N != -1);

 cout << "Sum is: " << Tot << endl;

 return 0;
}
```

For <u>each</u> question below, circle <u>any and all</u> loop structures for which the claims are true:

(A) The test for each repetition is performed after the pass through the loop body.

leading decision (trailing decision)

(B) The body will always be executed at least once.

leading decision (trailing decision)

(C) This structure is appropriate for sentinel controlled loops that use a "prime read".

(leading decision) trailing decision

(D) Sentinel controlled loops using this structure need an extra test to guard the body from the sentinel value.

leading decision (trailing decision)

For <u>each</u> question below, circle <u>any and all</u> loop control methods for which the claims are true:

(A) This control method is based on values assigned by the programmer, not given by the user.

(counting) sentinel

(B) This control method is based on values entered by the user.

counting (sentinel)

(C) The number of passes that the loop will perform can always be predetermined when the loop starts.

(counting) sentinel

DISK FILE INPUT AND OUTPUT:

Review the last part of Chapter 5 in your textbook that discuss reading and writing of sequential text files on disks. Then study the practice exercise on the following page. It involves a file with a simple name that is located in the same folder as the program. Be aware that special consideration must be given when providing filenames or paths that contain whitespaces or characters that C++ interprets as having special meaning, such as backwards slashes. For example, to open a file named "My File.txt" entered at the keyboard, you would need to input the string using the getline() function to prevent the whitespace from truncating the portion of the filename starting at the whitespace before assigning it to a string variable, as in:

```
getline (cin, StringVar);  // read a string containing whitespaces
```

To associate a file accessible from the disk location "C:\User\JStudent\Data.txt" with a file input stream object named "inFile", you would have to escape the backwards slashes (\) as follows:

```
inFile.open("C:\\User\\JStudent\\Data.txt ");
```

```
/*************************************************************
* Program: diskio.cpp - Practice Exercise                    *
* Opens a file in the same folder as the program based on a  *
* user supplied name (without whitespaces). Then writes ten  *
* lines of text, closes the file, then reopens the file and  *
* reads and displays each of the ten lines from the file.    *
*************************************************************/

#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
 string FILENAME;   // String object to hold a filename

 system ("cls");       /* Clear the Screen */
 cout << "Disk I/O Program\n\n";

 cout << "What is the filename (w/o path)? ";
 cin >> FILENAME;

 ofstream fileOut;
 fileOut.open(FILENAME.c_str()); // open requires a C-String
 if (fileOut)  // Verify successful open
 {
  cout << FILENAME << " opened successfully for writing.\n\n";
  for (int C=1; C<=10; C++) fileOut << "Line " << C << endl;
  fileOut.close();
 }
 else cout << FILENAME << "\a failed to open.\n";

 string OSTR;
 ifstream fileIn;
 fileIn.open(FILENAME.c_str());
 if (fileIn)
 {
  cout << FILENAME << " opened successfully for reading.\nContents:\n";
  for (int C=1; C<=10; C++)
  {
      getline(fileIn,OSTR);  // read a line of text with whitespaces
      cout << OSTR << endl;
  }
  fileIn.close();
 }
 else
  cout << FILENAME << "\a failed to open.\n";

 system ("pause");
 return 0;
}
```