Decision Control Structures and Loop Structures

Selection (IF STATEMENTS)

The if statements allows branching (decision making) depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions. The basic format is,

```
if( expression )
  { program statement; }
```

Nested If Else

Test for multiple cases by placing if/else selection structures inside if/else selection structures.Once condition is met, rest of statements skipped

The general format is

```
if( condition 1 )
        statement1;
else if( condition 2 )
        statement2;
else if( condition 3 )
        statement3;
else
        statement4;
```

Switch Case

The *switch case* statement is a better way of writing a program when a series of *if elses* occurs. The general format for this is,

```
switch ( expression )
{ case value1:
                program statement;
                program statement;
                 ......
                break;
case valuen:
                program statement;
                .......
                 break;
default:
                 .......
                 .......
                 break;
}
```

The keyword *break* must be included at the end of each case statement. The default clause is optional, and is executed if the cases are not met. The right brace at the end signifies the end of the case selections.

Rules for switch statements

values for 'case' must be integer or character constants.The order of the 'case' statements is unimportant

Loops structures

1.for loop

for (*expr1*; *expr2*; *expr3*)
{statement/s;}

Grammatically, the three components of a for loop are expressions. Most commonly, *expr1* and *expr3* are assignments or function calls and *expr2* is a relational expression. Any of the three parts can be omitted, although the semicolons must remain. If *expr1* or *expr3* is omitted, it is simply dropped from the expansion. If the test, *expr2*, is not present, it is taken as permanently true, so
for (;;) {
...
}
is an ``infinite'' loop, presumably to be broken by other means, such as a break or return. The for is preferable when there is a simple initialization and increment since it keeps the loop control statements close together and visible at the top of the loop. This is most obvious in
for (i = 0; i < n; i++)
{...}

2.while loop

while (*expression*)
{statement/s;}
the *expression* is evaluated. If it is non-zero, *statement* is executed and *expression* is re-evaluated. This cycle continues until *expression* becomes zero, at which point execution resumes after *statement*.

3.do..while loop
The syntax of the do is

do
{statement/s;}
while (*expression*);
The *statement* is executed, then *expression* is evaluated. If it is true, *statement* is evaluated again, and so on. When the expression becomes false, the loop terminates.The while and for loops test the termination condition at the top.The do-while, tests at the bottom *after* making each pass through the loop body; the body is always executed at least once.Hence for and while loop are **pre-test loops** and do..while is a **post test loop**.

Break statement
A break statement terminates the execution of the loop.The control is transferred to the statement immediately following the loop.
Consider a program to calculate sum of positive integers entered by the user.

Code Using while loop

```
scanf(“%d”,&i);
while(i!=-1)
{
   sum+=i;
   scanf(“%d”,&i);
}
```

Alternate code using break to eliminate 2 scanf statements

```
while(1)
{scanf("%d",&i);
  if(i==-1)
    break;
sum+=i;
}
```

Whenever -1 is input,the condition i==-1 evaluates to true.Thus break statement is executed causing the execution of the loop to be terminated.Control passes to the statement following the while construct.The condition is while is given as 1 which is non-zero and hence is always true.This specifies an infinite loop.Break statement terminates its execution

Continue Statement

It is used to bypass the remainder of the current pass through a loop.The loop does not terminate when a continue statement is encountered.The remaining loop statements are skipped and the execution proceeds to the next pass/iteration .

Consider a program to find sum of 5 positive integers.

```
for(i=0;i<=5;i++)
{
   printf("enter an integer");
   scanf("%d",&n);
   if(n<0)
  { printf("negative number");
     continue; }
sum+=n;
}
```

goto statement
Used to alter the normal sequence of program execution by transferring control to some other part of the program.

goto label;

Label is an identifier used to label the target statement to which control would be transferred

Eg:

```
//WAP to demonstrate goto statement
int main() {
   int num ;

   printf("Enter a number: ");
   scanf("%d", &num);

   // go to action if the user enters a negative number
   if (num < 0)
   {
      goto action;// action is the label
   }
   else
   {
       if(num%2==0)
        printf("%d is positive and even",num);
       else
        printf("%d is positive and odd",num);
   }
        return 0;

   action:
   printf("%d can't be negative",num);
}
```