C - DECISION MAKING CONTROL STATEMENT AND BRANCHING

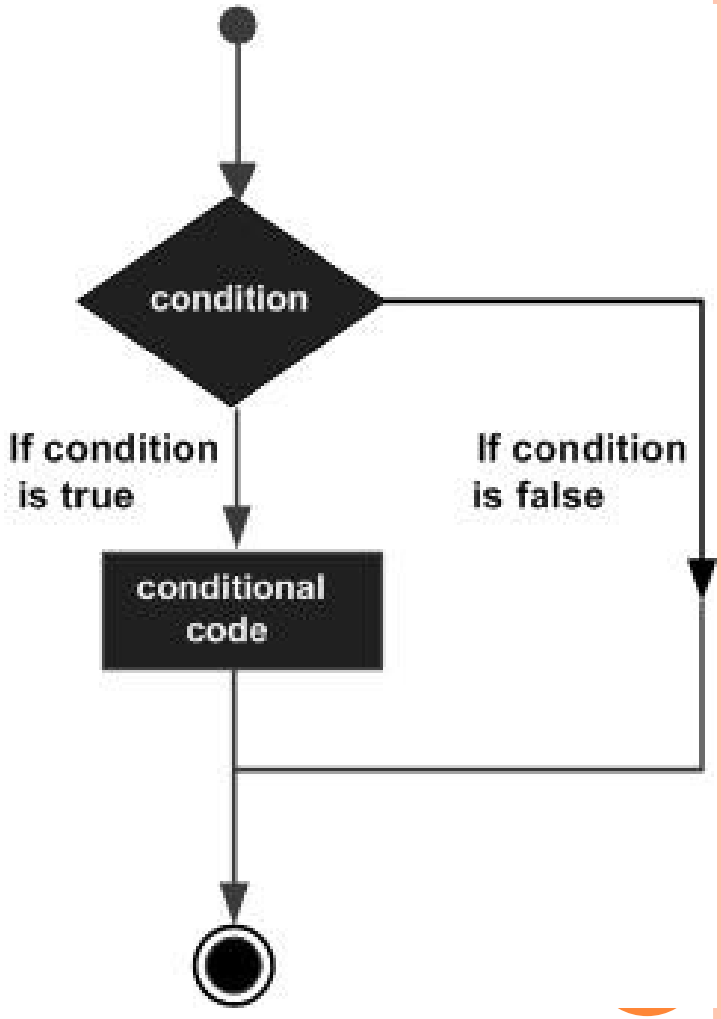
Dr. Sheak Rashed Haider Noori
Professor & Associate Head
Department of Computer Science

C - DECISION MAKING

- Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

C - DECISION MAKING CONT.

Following is the general form of a typical decision making structure found in most of the programming languages:



C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

TYPES OF CONTROL STATEMENTS

C language provides following types of decision making statements. Click the following links to check their detail.

Statement	Description
1. ***if*** statement	An **if statement** consists of a Boolean expression followed by one or more statements.
2. ***if...else*** statement	An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false.
3. **nested *if*** statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4. ***if...else if..else*** statement	An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.
5. ***switch case***-statement	A **switch** statement allows a variable to be tested for equality against a list of values.
6. nested ***switch*** statements	You can use one **switch** statement inside another **switch** statement(s).
7. Ternary operator **? :**	?: can be used to replace **if...else** statements
8. ***goto*** statement	The **goto** statement transfers control to a label.

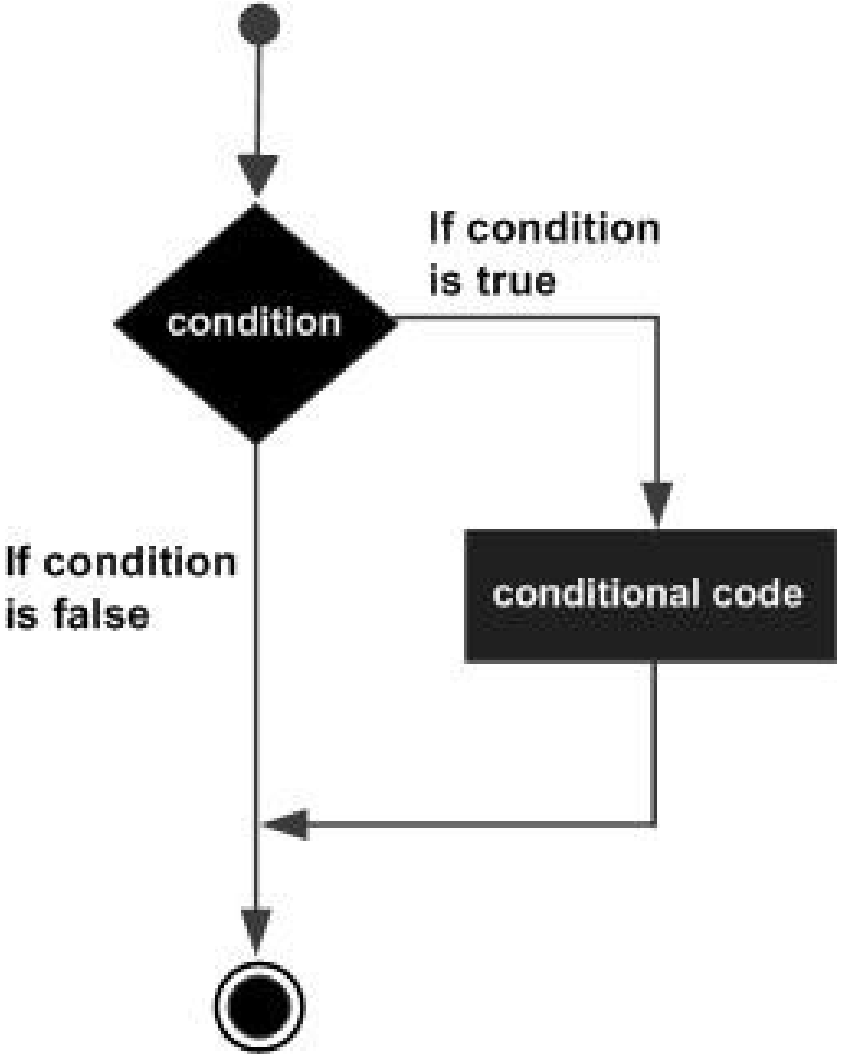
1. IF STATEMENT

- An **if** statement consists of a Boolean expression followed by one or more statements.
- **Syntax:**

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
```

- If the expression evaluates to **true**, then the block of code inside the if statement will be executed.
- If expression evaluates to **false**, then the first set of code after the end of the if statement will be executed.
- C language assumes any **non-zero** and **non-null** values as **true** and if it is either zero or null, then it is assumed as **false** value.

IF FLOW DIAGRAM



CODE EXAMPLE

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* check the boolean condition using if statement */
   if( a < 20 )
   {
       /* if condition is true then print the following */
       printf("a is less than 20\n" );
   }
   printf("value of a is : %d\n", a);

   return 0;
}
```

```
a is less than 20;
value of a is : 10
```

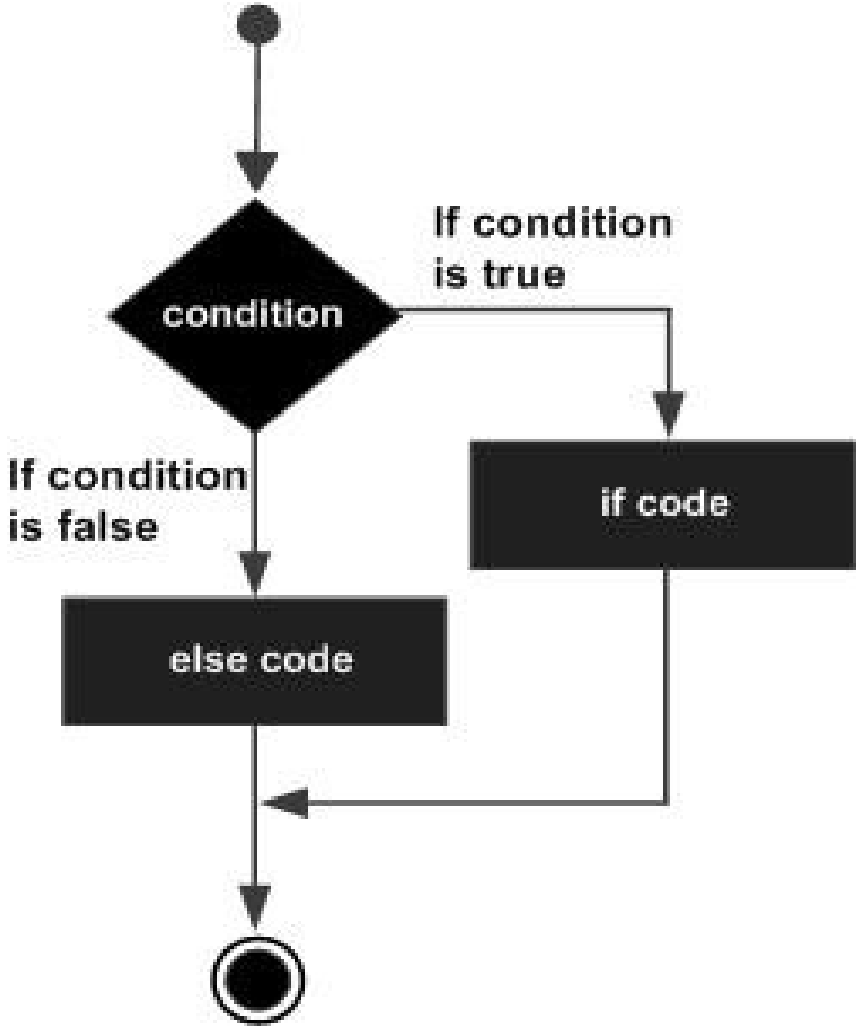
2. IF…ELSE STATEMENT

- An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.
- **Syntax**:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

- If the Boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

IF…ELSE FLOW DIAGRAM



CODE EXAMPLE : IF ELSE

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 )
   {
       /* if condition is true then print the following */
       printf("a is less than 20\n" );
   }
   else
   {
       /* if condition is false then print the following */
       printf("a is not less than 20\n" );
   }
   printf("value of a is : %d\n", a);

   return 0;
}
```

```
a is not less than 20;
value of a is : 100
```

3. THE IF…ELSE IF…ELSE STATEMENT

- An **if** statement can be followed by an optional **else if…else** statement, which is very useful to test various conditions using single if…else if statement.
- When using if , else if , else statements there are few points to keep in mind:
  - An if can have zero or one else's and it must come after any else if's.
  - An if can have zero to many else if's and they must come before the else.
  - Once an else if succeeds, none of the remaining else if's or else's will be tested.

THE IF...ELSE IF...ELSE SYNTAX.

```
if(boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
   /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
   /* Executes when the boolean expression 3 is true */
}
else
{
   /* executes when the none of the above condition is true */
}
```

CODE EXAMPLE: IF...ELSE IF...ELSE

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a == 10 )
   {
       /* if condition is true then print the following */
       printf("Value of a is 10\n" );
   }
   else if( a == 20 )
   {
       /* if else if condition is true */
       printf("Value of a is 20\n" );
   }
   else if( a == 30 )
   {
       /* if else if condition is true  */
       printf("Value of a is 30\n" );
   }
   else
   {
       /* if none of the conditions is true */
       printf("None of the values is matching\n" );
   }
   printf("Exact value of a is: %d\n", a );

   return 0;
}
```

4. C - NESTED IF STATEMENTS

- It is always legal in C to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).
- Syntax:

```
if( boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */
   if(boolean_expression 2)
   {
      /* Executes when the boolean expression 2 is true */
   }
}
```

- You can nest **else if...else** in the similar way as you have nested *if* statement.

CODE EXAMPLE: NEST **ELSE IF...ELSE**

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   /* check the boolean condition */
   if( a == 100 )
   {
       /* if condition is true then check the following */
       if( b == 200 )
       {
          /* if condition is true then print the following */
          printf("Value of a is 100 and b is 200\n" );
       }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

5. C - SWITCH STATEMENT

- A **switch** statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each **switch case**.
- Syntax:

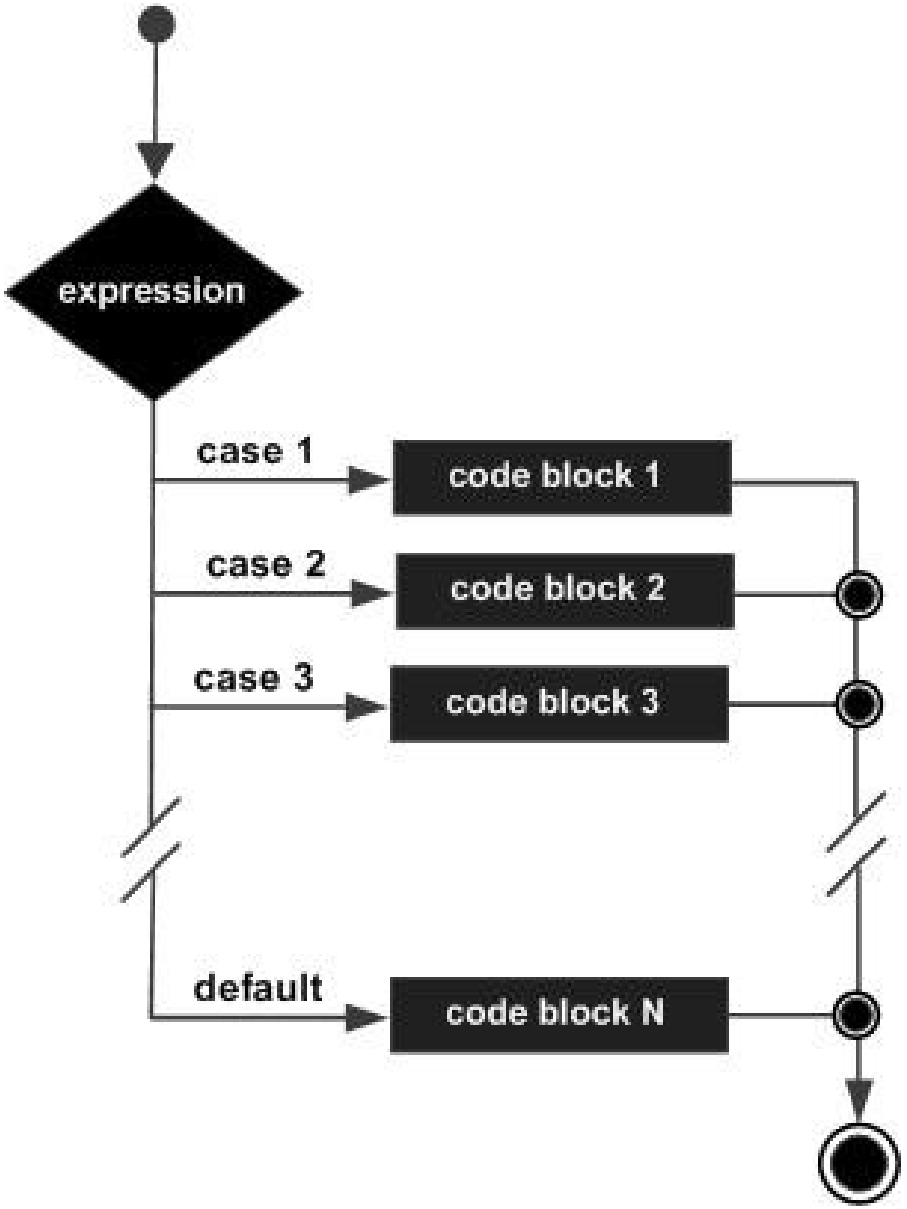
```
switch(expression){
    case constant-expression  :
       statement(s);
       break; /* optional */
    case constant-expression  :
       statement(s);
       break; /* optional */

    /* you can have any number of case statements */
    default : /* Optional */
       statement(s);
}
```

RULES APPLY TO A **SWITCH** STATEMENT:

- The **expression** used in a **switch** statement must have an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch.
- The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

SWITCH-CASE FLOW DIAGRAM



CODE EXAMPLE: SWITCH-CASE

```
int main ()
{
   /* local variable definition */
   char grade = 'B';

   switch(grade)
   {
   case 'A' :
      printf("Excellent!\n" );
      break;
   case 'B' :
   case 'C' :
      printf("Well done\n" );
      break;
   case 'D' :
      printf("You passed\n" );
      break;
   case 'F' :
      printf("Better try again\n" );
      break;
   default :
      printf("Invalid grade\n" );
   }
   printf("Your grade is  %c\n", grade );

   return 0;
}
```

6. C - NESTED SWITCH STATEMENTS

- It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.
- **Syntax**:

```
switch(ch1) {
   case 'A':
      printf("This A is part of outer switch" );
      switch(ch2) {
         case 'A':
            printf("This A is part of inner switch" );
            break;
         case 'B': /* case code */
      }
      break;
   case 'B': /* case code */
}
```

CODE EXAMPLE: NESTED SWITCH-CASE

```
int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   switch(a) {
      case 100:
         printf("This is part of outer switch\n", a );
         switch(b) {
            case 200:
               printf("This is part of inner switch\n", a );
         }
   }
   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

7. THE ? : OPERATOR

- We have covered **conditional operator ? :** previously which can be used to replace **if...else** statements. It has the following general form:

```
Exp1 ? Exp2 : Exp3;
```

- Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.
- The ?: is called a ternary operator because it requires three operands.
- The value of a ? expression is determined like this:
  - Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
  - If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

THE ? : OPERATOR CONT.

- ?: operator can be used to replace if-else statements, which have the following form:

```
if(condition){
   var = X;
}else{
   var = Y;
}
```

- For example, consider the following code:

```
if(y < 10){
   var = 30;
}else{
   var = 40;
}
```

  - Above code can be rewritten like this:

```
var = (y < 10) ? 30 : 40;
```

Here, x is assigned the value of 30 if y is less than 10 and 40 if it is not. You can the try following example:

8. THE GOTO STATEMENT

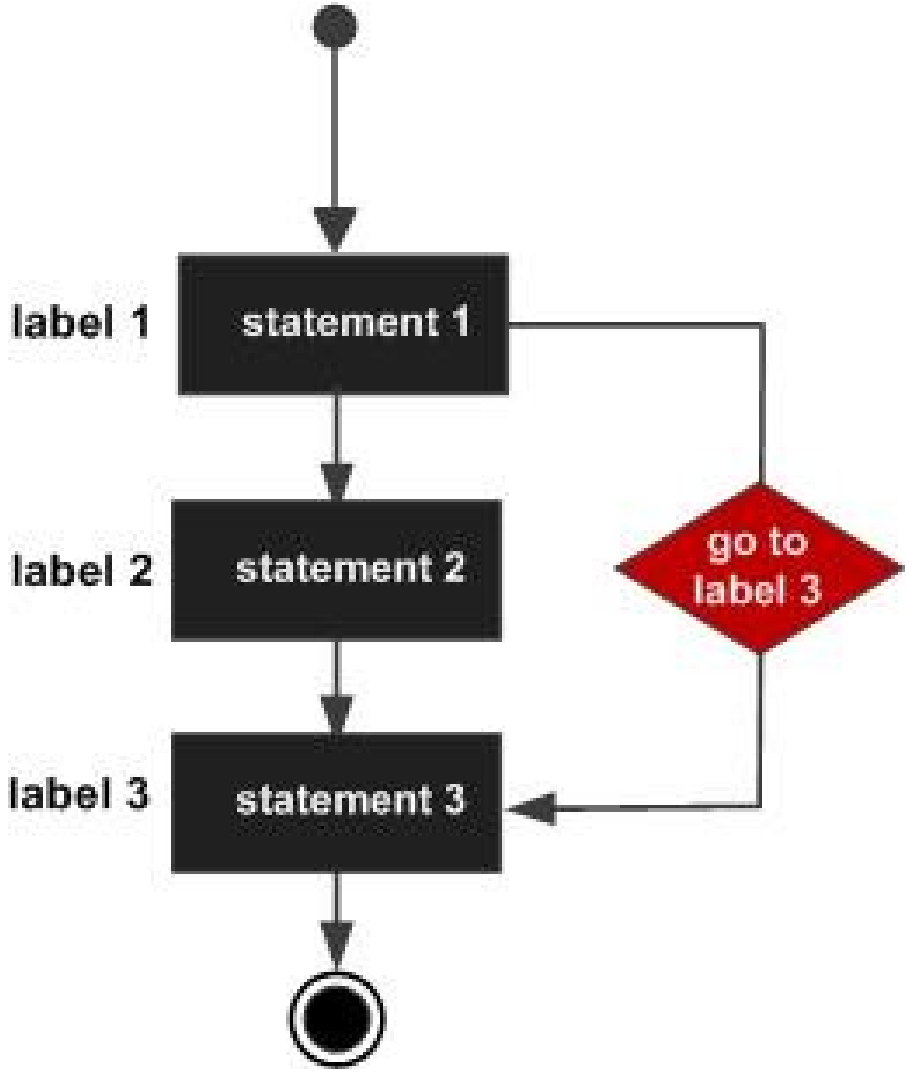
- A **goto** statement in C language provides an unconditional jump from the goto to a labeled statement in the same function.
- The given label must reside in the same function.
- **Syntax**:

```
goto label;
..
.
label: statement;
```

- Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

NOTE: Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten so that it doesn't need the goto.

GOTO STATEMENT FLOW DIAGRAM



CODE EXAMPLE: GOTO STATEMENT

```
int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }
      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```