UNIVERSITY of SAN CARLOS
SCIENTIA • VIRTUS • DEVOTIO

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

CIS 1101 – PROGRAMMING 1

CONTROL STRUCTURES

Part 1

CONTROL STRUCTURES: DEFINITION

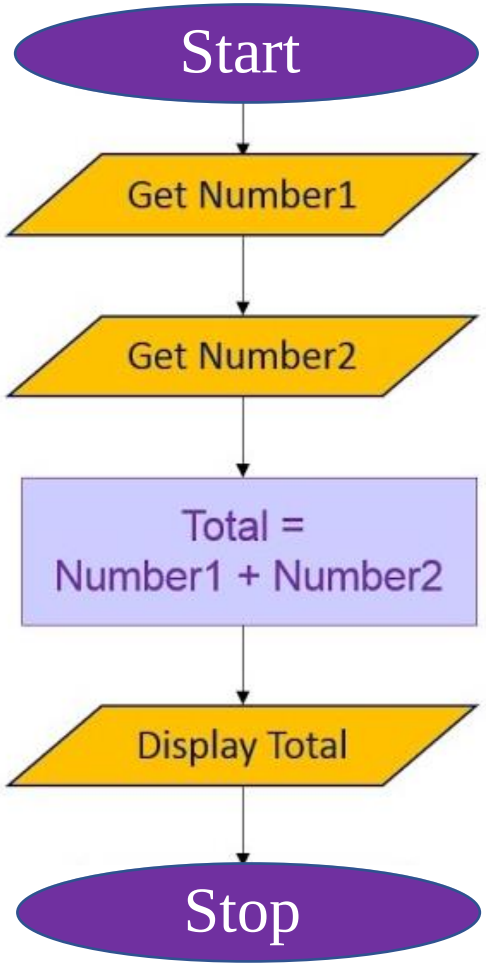
- These are the structures that are used to control the flow of a program.

- Structures are used to make decisions and alter the direction of program flow in one or the other path(s) available.

TYPES OF CONTROL STRUCTURES IN C: SEQUENCE

1. Sequence structure

- Statements are executed one after the other in the order in which they are written, that is, in sequence.



Start

Get Number1

Get Number2

Total = Number1 + Number2

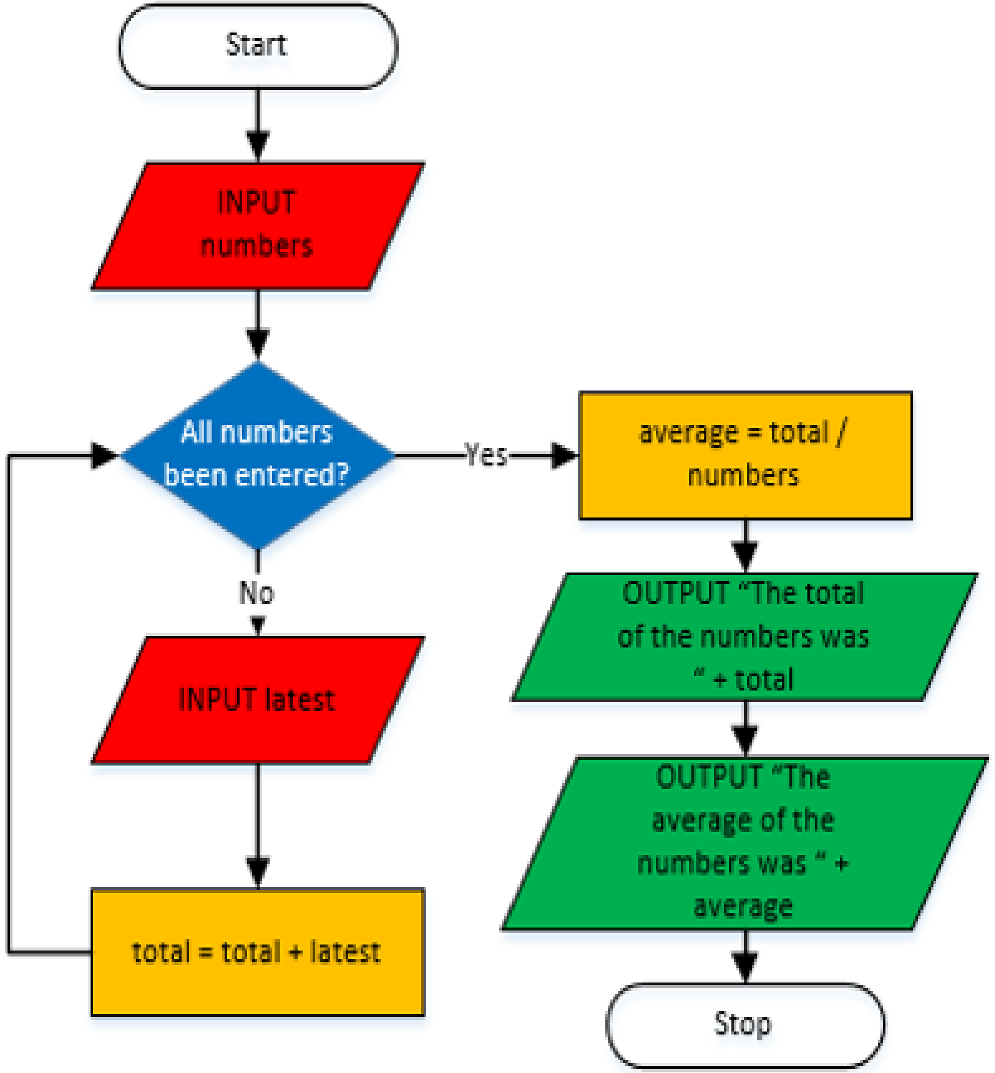
Display Total

Stop

TYPES OF CONTROL STRUCTURES IN C: SELECTION

2. Selection structure

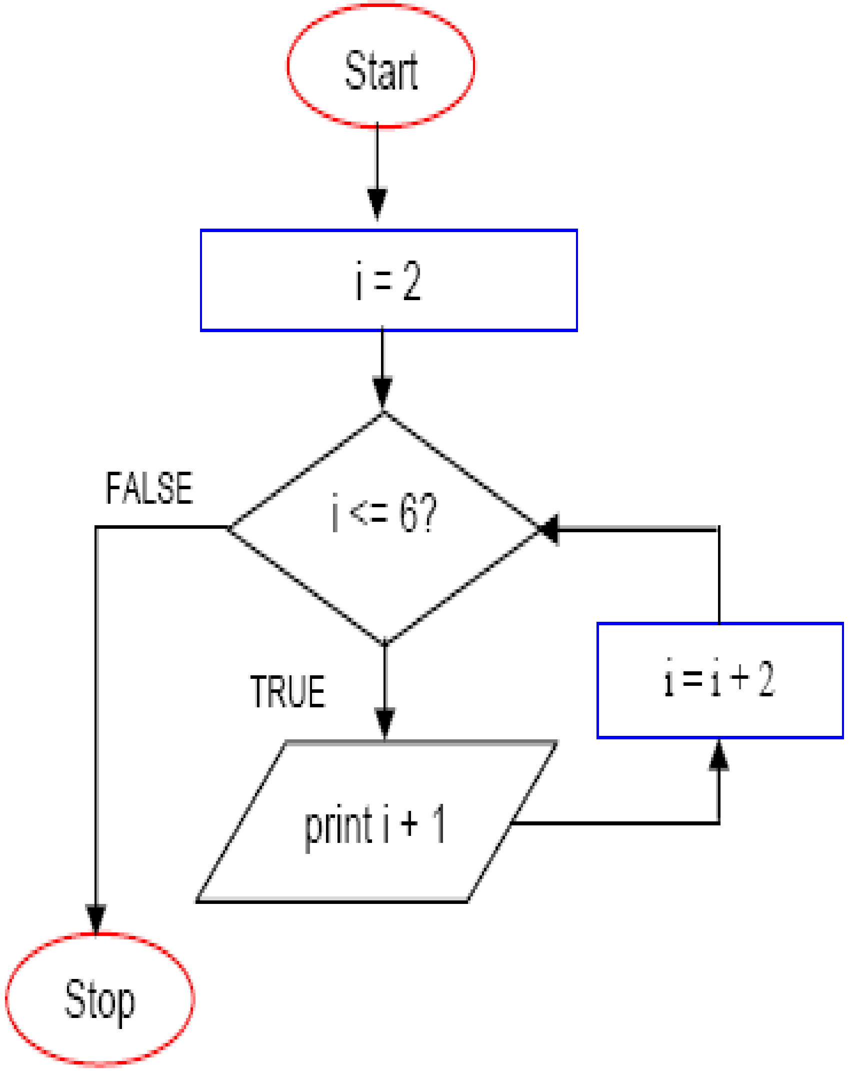
- It is a structure in which a question is asked, and depending on the answer, the program takes **one of two** courses of action, after which the program moves on to the next event.


Start
INPUT numbers
All numbers been entered?
Yes
average = total / numbers
OUTPUT "The total of the numbers was " + total
OUTPUT "The average of the numbers was " + average
Stop
No
INPUT latest
total = total + latest

TYPES OF CONTROL STRUCTURES IN C: ITERATION

3. Loop structure (iteration)

- A set of instructions or structures are repeated in a sequence a specified number of times or until a condition is met.



BRANCHING

- Basic concept in computer science which means an instruction that tells a computer to begin executing a different part of a program rather than executing statements one-by-one.

- Implemented as a series of control flow statements called **CONTROL STATEMENTS** in high-level programming languages.

CONTROL STATEMENTS

- Provided by the computer language to be used to implement the control structures.

- Statements that specify the flow of a program.
- Control the order in which the instructions in a program must be executed

- Statements that make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

CONDITIONAL BRANCHING

- When program statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not.

- generally implemented with IF, IF-ELSE, and SWITCH-CASE statements

IF STATEMENT

- A simple control statement that tests whether a condition is **true** or **false**.

- **Condition:** can include a variable, or be a variable.

- **Tests** for ONLY **one action**.

IF STATEMENT

- If the condition is true, then an action occurs.

- If the condition is false, nothing is done.

IF STATEMENT: SYNTAX

CONTROL STRUCTURES IN C

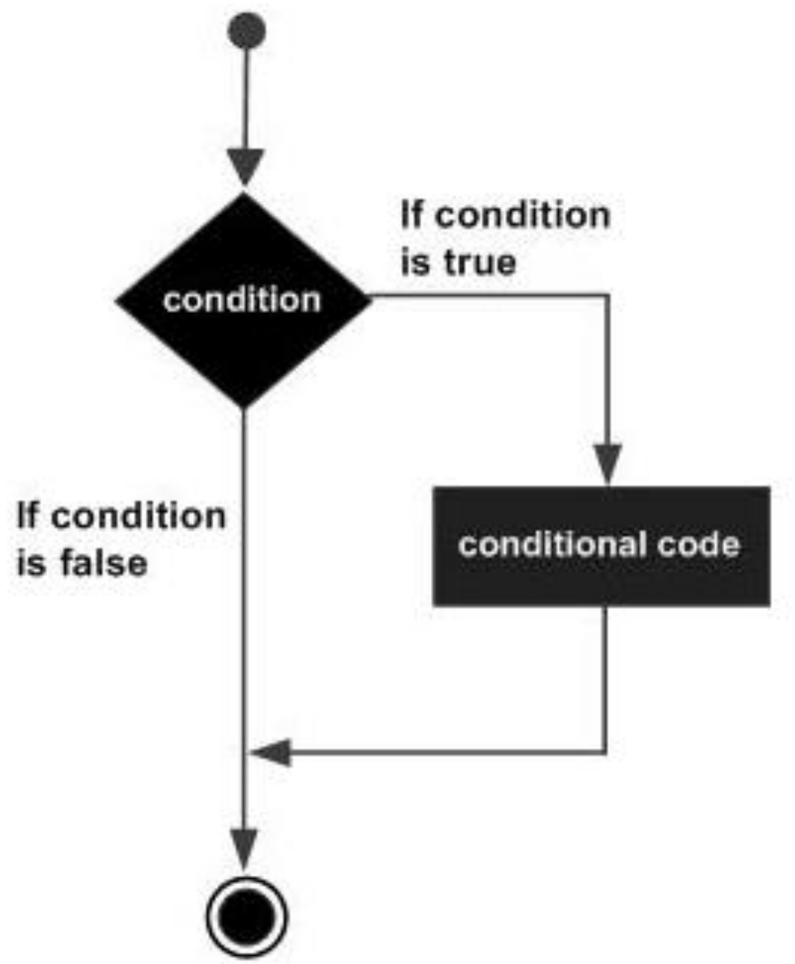
- if variable is true

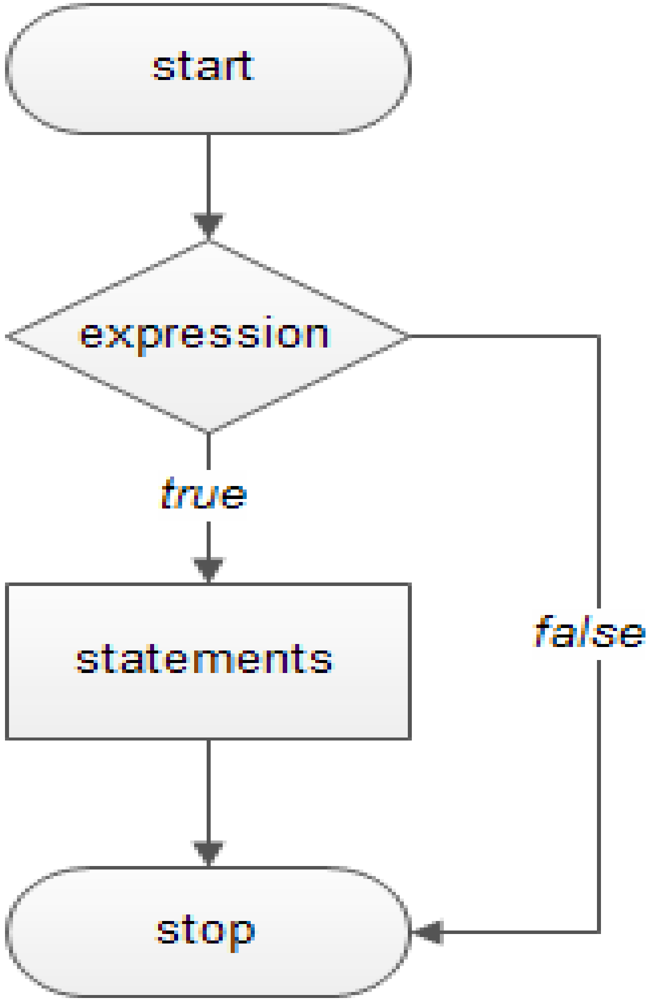
  Take this course of action

- if (condition)

```
{
     /*  Block of C statements here  */
     /*  These statements will only execute if the condition is true  */
}
```

IF STATEMENT: FLOWCHART





IF STATEMENT: IMPORTANT NOTES

- Condition must be a Boolean expression
- Statement is any valid C statement
- Statement is only executed if the condition evaluates to true.
- If the condition evaluates to **false**,
  - then the statement is **skipped** and
  - the next line of code following the **if statement** is executed.
- The **curly braces may be omitted**
  - when the body contains **only one statement**.

IF-ELSE STATEMENT

- If the condition returns true, then the statements inside the body of “if” are executed and the statements inside the body of “else” are skipped.
- If the condition is true,
  - then an action occurs.

- If the condition returns false, then the statements inside the body of “if” are skipped and the statements in “else” are executed.
- If the condition is false,
  - take an alternate action.

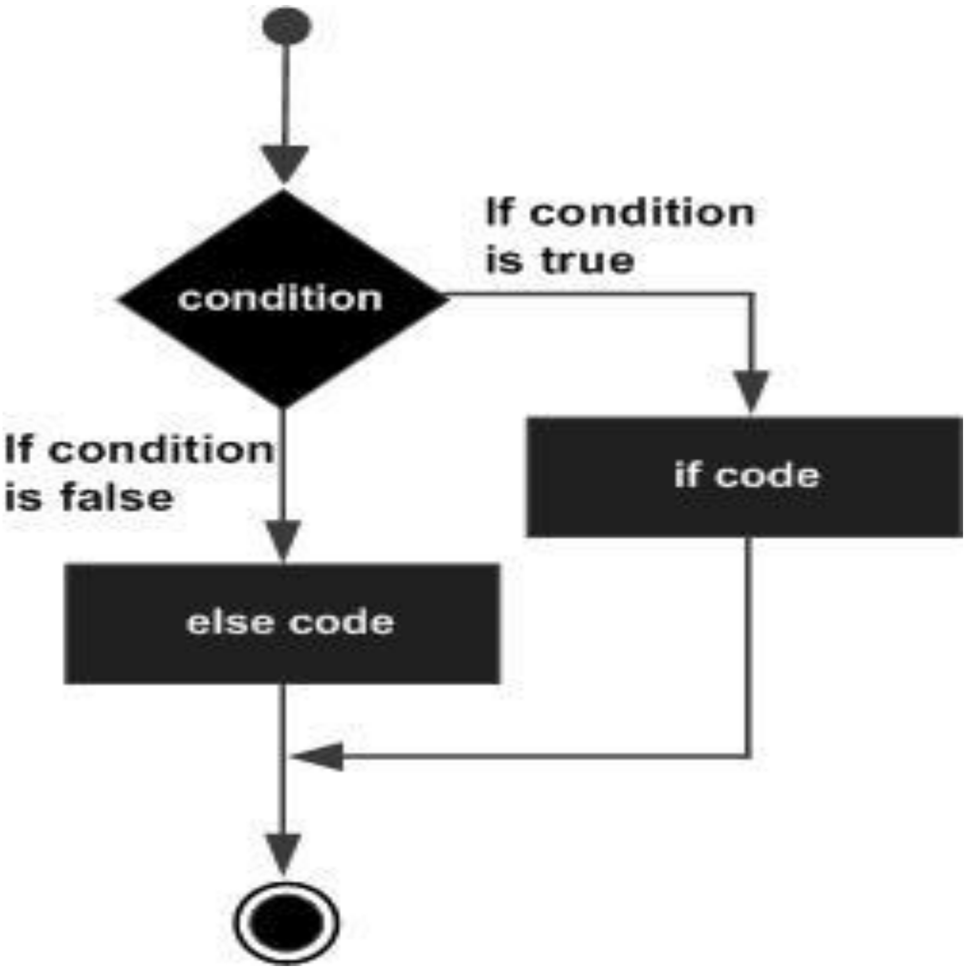
IF-ELSE STATEMENT: SYNTAX

-
  ```
  if variable is true
       Take this course of action
   else call another routine
  ```

-
  ```
  if (condition)
  {
       /* Statements inside body of if  */
  }
  else
  {
      /* Statements inside body of else */
  }
  ```

IF-ELSE STATEMENT: FLOWCHART

CONTROL STRUCTURES IN C



IF-ELSE STATEMENT: NESTED

- It is when an **if-else statement** is **present** inside the body of another “if” or “else” statement.

NESTED IF-ELSE STATEMENT: SYNTAX

```
if (condition1)
{
     /* Nested if else inside the body of "if" */
     if (condition2)
     {
          /* Statements inside the body of nested "if" */
     }
     else
     {
          /* Statements inside the body of nested "else" */
      }
}
else
{
     /* Statements inside the body of "else"  */
}
```

NESTED IF-ELSE STATEMENT: EXAMPLE

```
#include <stdio.h>

int main()
{
    int var1, var2;

    printf(“Enter the value of var1: ");
    scanf("%d", &var1);

    printf(“Enter the value of var2: ");
    scanf("%d", &var2);
```

```
    if (var1 != var2)
    {
        printf("var1 is not equal to var2.\n");
       /* Nested if else */
        if (var1 > var2)
        {
            printf("var1 is greater than var2.\n");
        }
        else
        {
            printf("var2 is greater than var1.\n");
        }
     }
    else
    {
       printf("var1 is equal to var2.\n");
    }
    return 0;
}
```

SWITCH-CASE STATEMENT

- A better way of writing a program when a series of **if-else statements** occurs.

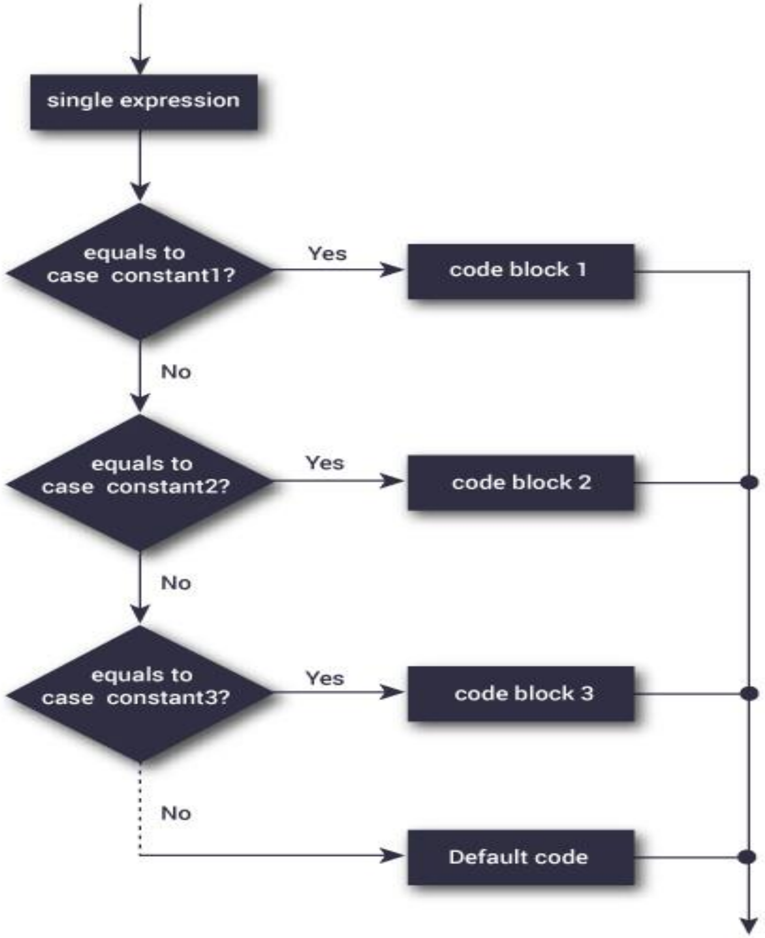
- Used for **multiple way selections** that will branch into different code segments or program statements based on the value of a **variable or expression**.

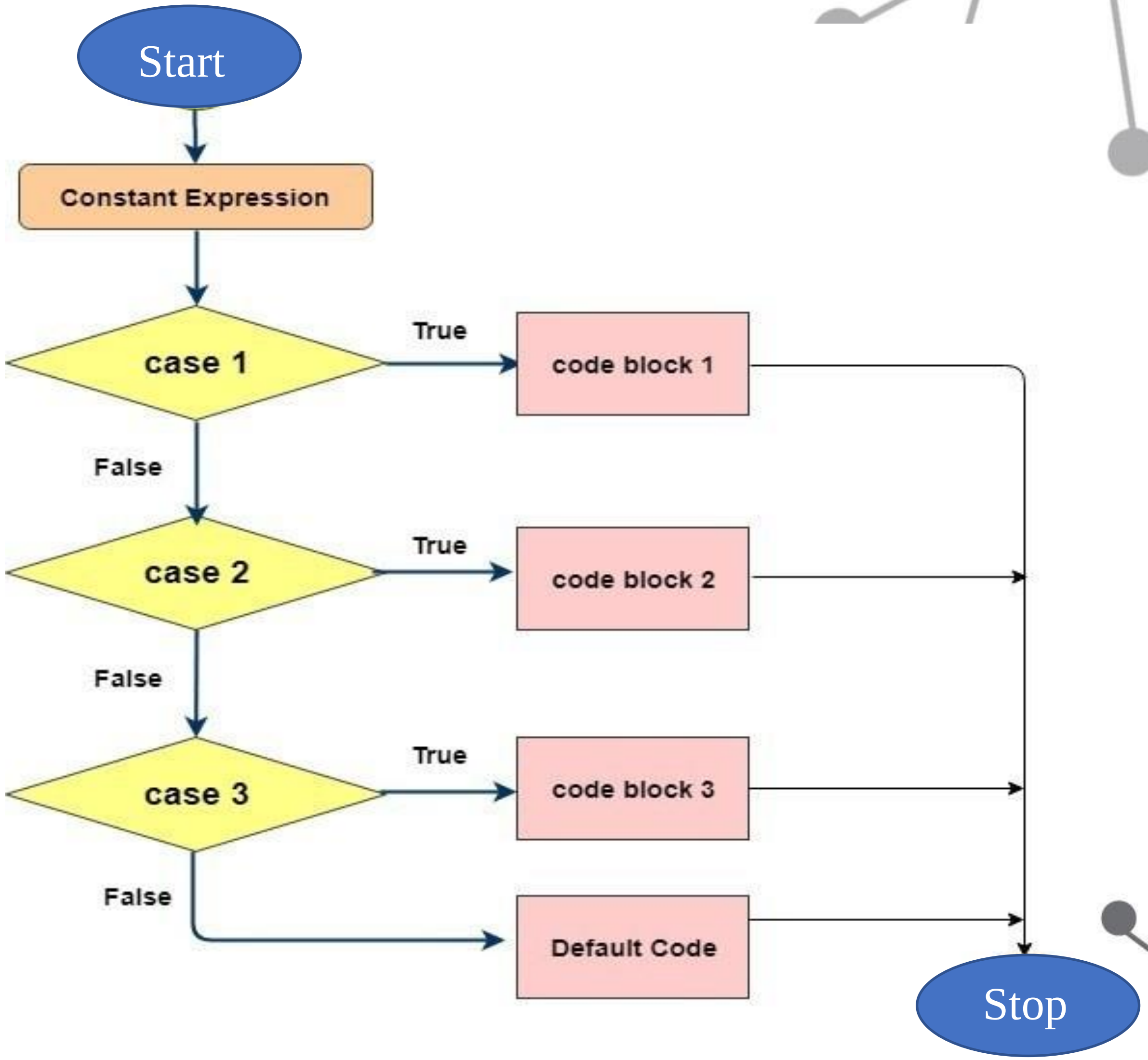
- **Variable or expression:**
  - **must** be of integer data type.

SWITCH-CASE STATEMENT: SYNTAX

```
switch (expression)
{
        case value1:
                program statement;
                break;
        case value2:
                program statement;
                break;
                .
                .
        case valueN:
                program statement;
                break;
        default:
                default program statement;
}
```

SWITCH-CASE STATEMENT: FLOWCHART





SWITCH-CASE STATEMENT: EXAMPLE

```
#include<stdio.h>

int main( )
{
   int day;

   printf(“\nEnter the number of the day: ");
   scanf("%d", &day);
```

```
        switch(day) {
              case 1: printf(“\nIt is a Sunday");
                          break;
              case 2: printf("\nIt is a Monday");
                          break;
              case 3: printf("\nIt is a Tuesday");
                          break;
              case 4: printf("\nIt is a Wednesday");
                          break;
              case 5: printf("\nIt is a Thursday");
                          break;
              case 6: printf("\nIt is a Friday");
                          break;
              case 7: printf("\nIt is a Saturday");
                          break;
              default: printf("\nIt is an Invalid choice!");
                            break;
        }
        return 0;
}
```

UNCONDITIONAL BRANCHING

- It is also known as UNCONDITIONAL CONTROL TRANSFER.

- It is when the programmer forces the execution of a program to jump to another part of the program.

- This can be done using a good combination of loops and if statements.

- This technique should always be avoided by a programmer and this is used only when it is very difficult to use a loop.

UNCONDITIONAL BRANCHING: GOTO

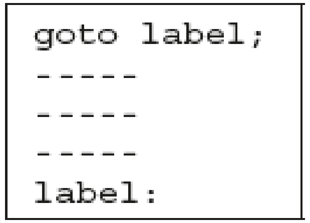
- The **goto statement** is used for unconditional branching or transfer of the program execution to the labeled statement.

- This statement does not require any condition.

- This statement passes control anywhere in the program without considering any condition.

- Avoid using the **goto** statement.

UNCONDITIONAL BRANCHING: GOTO



```
goto label;
-----
-----
-----
label:
```

The goto statement

```
#include<stdio.h>

int main()
{
        int x;
        printf("\nEnter a number: ");
        scanf("%d",&x);

        if(x%2==0) {
            goto even;
        }else {
            goto odd;
         }
        even: printf("\nThe number is even.");
        odd: printf("\nThe number is odd.");
        return 0;
}
```

UNCONDITIONAL BRANCHING: CONTINUE

- This statement works somewhat like the break statement.
- **This statement skips the current iteration of the loop and continues with the next iteration.**
- In the case of for loop, this statement initiates the testing condition and increment on steps has to be executed (while the other statements following continue are neglected).
- For while and do while, this statement causes control to pass on to conditional tests.
- It is useful in a programming situation where it is required that particular iterations occur only up to some extent or when some part of the code has to be neglected.

UNCONDITIONAL BRANCHING: CONTINUE

```
# include <stdio.h>

int main()
{
    int i;
    double number, sum = 0.0;
```

```
    for(i=1; i <= 10; ++i)
    {
        printf("Enter n%d: ",i);
        scanf("%lf", &number);
        if(number < 0.0)
        {
            continue;
        }
        sum += number;   /* sum = sum + number; */
    }
    printf("Sum = %.2lf",sum);
    return 0;
}
```

UNCONDITIONAL BRANCHING: BREAK

- Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the **loop** or **switch**.
- The control then automatically passes on to the first statement after the loop or the block.
- This statement can be associated with all the conditional statements (especially **switch case**).
- This statement can be used in the nested loops.
- The widest used of this statement is in the switch case where it is used to avoid flow of control from one ‘case’ to the other.

UNCONDITIONAL BRANCHING: BREAK

```
# include <stdio.h>

int main()
{
    int i;
    double number, sum = 0.0;
```

```
    for(i=1; i <= 10; ++i)
    {
        printf("Enter a n%d: ",i);
        scanf("%lf",&number);
        /* If the user enters a negative number, the loop ends */
        if(number < 0.0)
        {
            break;
        }
        sum += number;     /* sum = sum + number; */
    }

    printf("Sum = %.2lf",sum);
    return 0;
}
```

UNCONDITIONAL BRANCHING: RETURN

- The return statement terminates the execution of a function and returns control to the calling function.

- Execution resumes in the calling function at the point immediately following the call.

- A return statement can also return a value to the calling function.

UNCONDITIONAL BRANCHING: RETURN

- **Syntax:**

```
return expression;
```

- **return** keyword transfers program control to the caller.
- **expression** is optional and is used to return result of a valid C expression to the caller.