2
Control Flow

Why Learning Control Flow?

- The execution of C programming statements is normally in sequence from start to end.
- In the last lecture, we have discussed the simple data types, arithmetic calculations, simple assignment statements and simple input/output. With these statements, we can design simple C programs.
- However, the majority of challenging problems requires programs with the ability to make decisions as to what code to execute and the ability to execute certain portions of the code repeatedly.
- C provides a number of statements that allow **branching** (or selection) and **looping** (or repetition).
- In this lecture, we discuss the branching and looping constructs in C.

Control Flow

- **Relational Operator and Logical Operator**
- Branching: if, if…else, if….else if….else Statements; Nested if Statements; The switch Statement; Conditional Operators
- Looping: for, while, do-while; Nested Loops; The break and continue Statements

Relational Operators

- Used to define a condition for comparing **two values**.
- Return **boolean** result: **true** or **false**.
- **Relational Operators**:

operator	example	meaning
==	ch == 'a'	equal to
!=	f != 0.0	not equal to
<	num < 10	less than
<=	num <=10	less than or equal to
>	f > -5.0	greater than
>=	f >= 0.0	greater than or equal to

Logical Operators

- **Works on one or more relational expressions** - to yield a logical return value: **true** or **false**.
- Allows testing the results on comparing expressions.
- **Logical operators**:

operator	example	meaning
!	!(num < 0)	not
&&	(num1 > num2) && (num2 >num3)	and
||	(ch == '\t') || (ch == ' ')	or

A && B	A is true	A is false
B is true	true	false
B is false	false	false

	A is true	A is false
!A	false	true

A || B	A is true	A is false
B is true	true	true
B is false	true	false

Operator Precedence

- list of operators of **decreasing precedence**:

!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
||	logical or

Boolean Evaluation in C

- The **result** of evaluating an expression involving relational and/or logical operators is either **true** or **false**.
  - **true** is **1**
  - **false** is **0**

- In general, **any integer expression whose value is non-zero is considered as true**; else it is **false**.

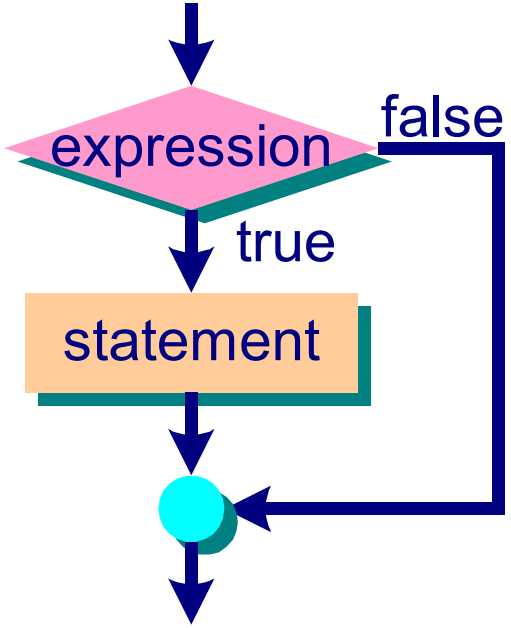
  For example:

3	**is true**
0	**is false**
1 && 0	is false
1 || 0	is true
!(5 >= 3) || (1)	is true

Control Flow

- Relational Operator and Logical Operator
- **Branching: if, if…else, if….else if….else Statements; Nested if Statements; The switch Statement; Conditional Operators**
- Looping: for, while, do-while; Nested Loops; The break and continue Statements

The if Statement




```
if (expression)
        statement;
        /* simple or compound statement
        enclosed with braces { } */
```

```
/* Program: check user number greater than 5 */
#include <stdio.h>
int main()
{
        int num;
        printf("Give me a number from 1 to 10: ");
        scanf("%d", &num);
        if (num > 5)
                printf("Your number is larger than 5.\n");
        printf("%d was the number you entered.\n",num);
        return 0;
}
```

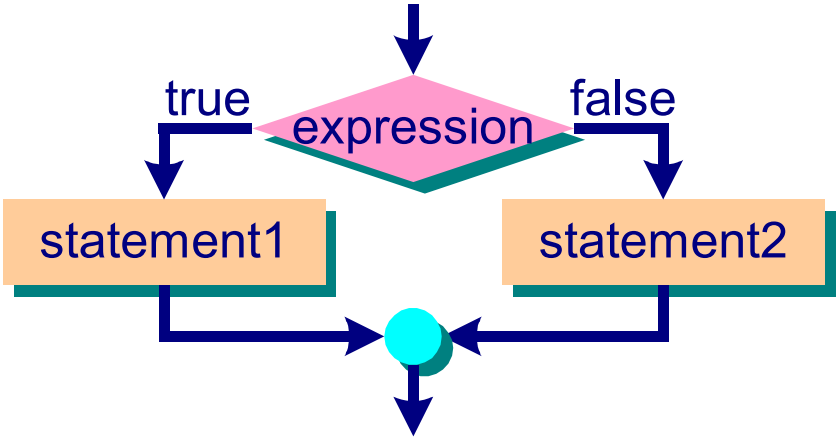
Output

```
Give me a number from 1 to 10:  3
3 was the number you entered.

Give me a number from 1 to 10:  7
Your number is larger than 5.
7 was the number you entered.
```

The if-else Statement

```
if (expression)
    statement1;
else
    statement2;
```




```
/* This program determines the maximum
value of num1 and num2 */
#include <stdio.h>
int main()
{
    int  num1, num2, max;
    printf("Please enter two integers:");
    scanf("%d  %d", &num1, &num2);
    if (num1 > num2)
        max = num1;
    else
        max = num2;
    printf("The maximum of the \
        two is %d\n",max);
    return 0;
}
```

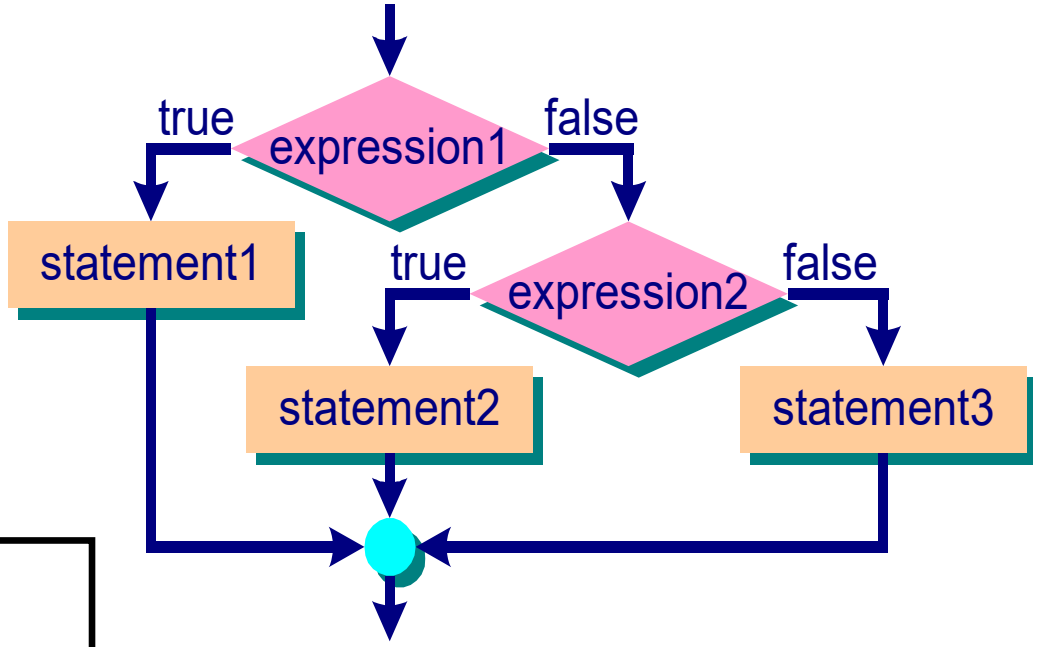
Output

Please enter two integers: 9 4
The maximum of the two is 9

Please enter two integers: -2 0
The maximum of the two is 0

The if-else if-else Statement

```
if  (expression1)
    statement1;
else if  (expression2)
    statement2;
...
else  statementN;
```




```
/* Program: Temperature reading. */
#include <stdio.h>
int main()
{
    float temp;
    printf("Temperature reading: ");
    scanf("%f",&temp);
    if ((temp >= 100.0) && ( temp <= 120.0))
        printf("Temperature OK.\n");
    else if (temp < 100.0)
        printf("Temperature too low.\n");
    else
        printf("Temperature too high.\n");
    return 0;
}
```

Output

Temperature reading: ***105.0***

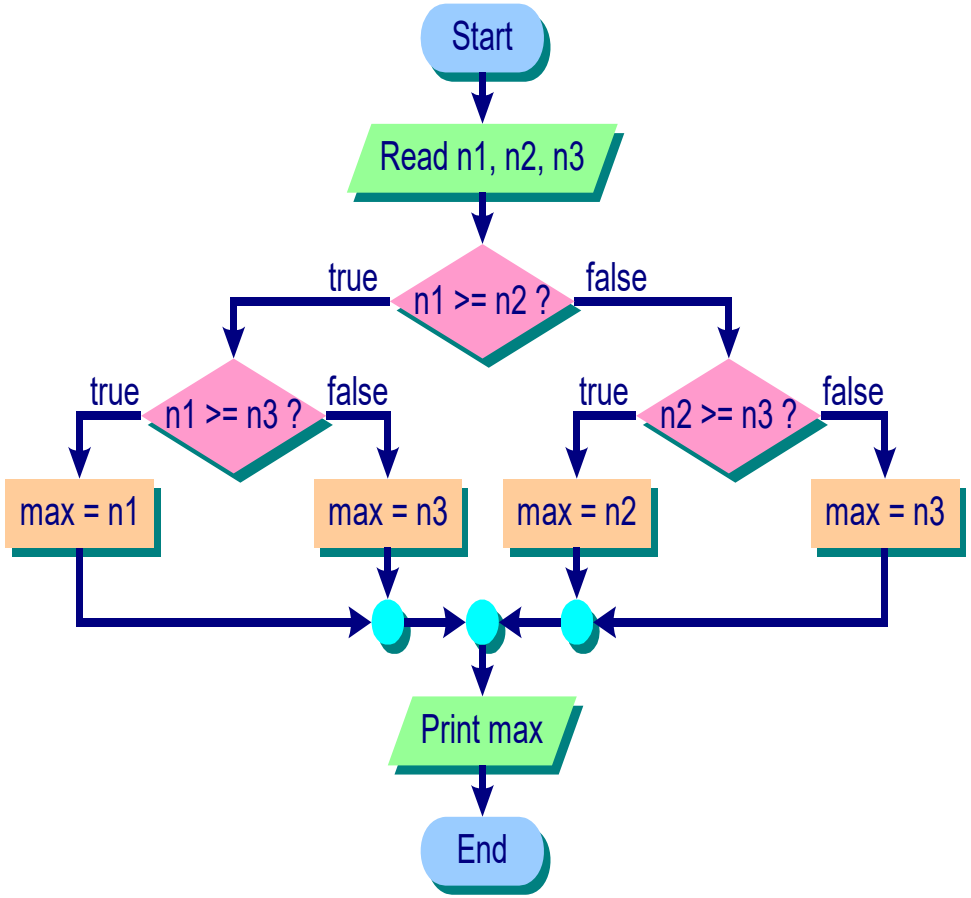
Temperature OK.

Temperature reading: ***130.0***

Temperature too high.

Nested-if

```
if (expression1)
    if (expression2)
            statement1;
    else
            statement2;
else
    statement3;
```




```
/* Determines max value of 3 num */
#include <stdio.h>
int main(){
    int   n1, n2, n3, max;
    printf("Please enter three integers:");
    scanf("%d  %d  %d", &n1, &n2, &n3);
    if (n1 >= n2) {
            if (n1 >= n3)
                max = n1;
            else max = n3;
    }
    else if (n2 >= n3)   max = n2;
    else max = n3;
    printf("The maximum is %d\n",max);
    return 0;
}
```

Output

Please enter three integers: *1 2 3*

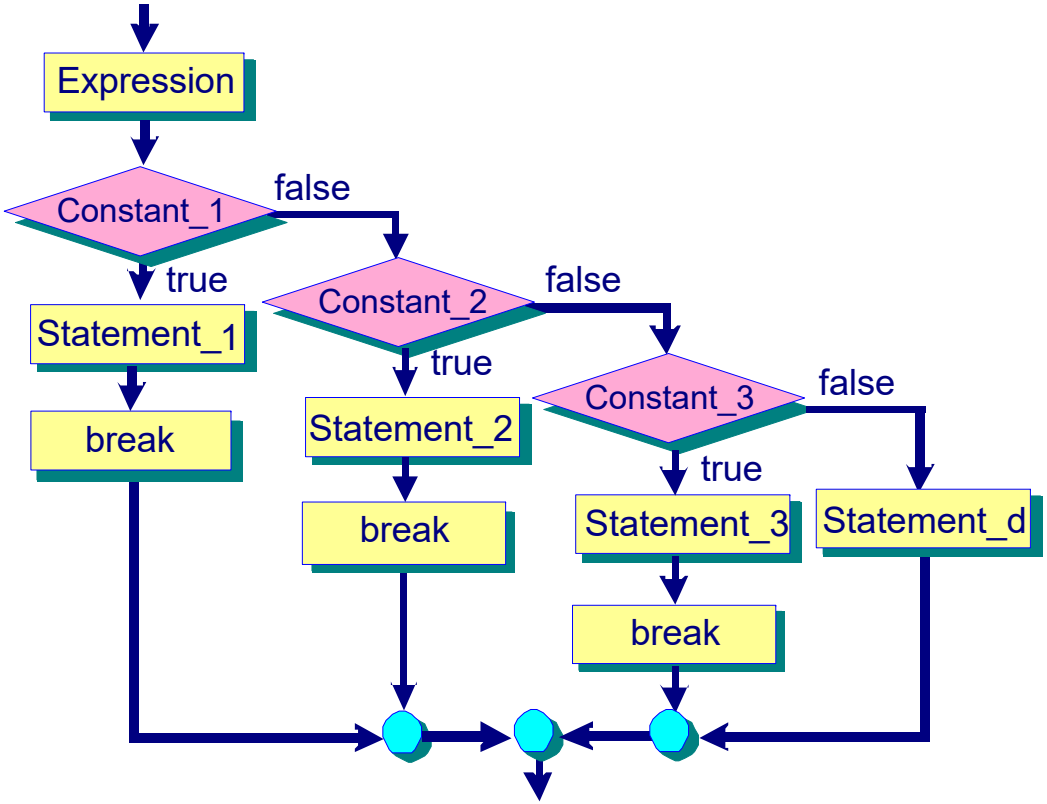
The maximum of the three is 3

The switch Statement

The **switch** is used for **multi-way selection**.

The syntax is:

```
switch (expression) {
  case constant_1:
          statement_1;
          break;
  case constant_2:
          statement_2;
          break;
  case constant_3:
          statement_3;
          break;
  default:
          statement_D;
}
```



The switch Statement: Syntax

The **switch** is used for **multi-way selection**.

The syntax is:

```
switch (expression) {
  case constant_1:
          statement_1;
          break;
  case constant_2:
          statement_2;
          break;
  case constant_3:
          statement_3;
          break;
  default:
          statement_D;
}
```

- **switch, case, break** and **default** - reserved keywords.
- The result of **expression** in ( ) must be of **integral type**.
- **constant_1, constant_2, ...** are called **labels**.
  - must be an **integer constant**, a **character constant** or an integer constant expression, e.g. 3, 'A', 4+'b', 5+7, ..., etc.
  - must deliver a **unique integer value**. Duplicates are not allowed.
  - may also have **multiple labels** for a statement, for example, to allow both lower and upper case selection.

switch: An Example

```
/* Arithmetic (A,S,M) computation of two user numbers */
#include <stdio.h>
 int main() {
     char choice;   int num1, num2, result;
     printf("Enter your choice (A, S or M) => ");
     scanf("%c", &choice);
     printf("Enter two numbers: ");
     scanf("%d %d", &num1, &num2);
     switch (choice)     {
       case 'a':
       case 'A':   result = num1 + num2;
         printf(" %d + %d = %d\n", num1,num2,result);
         break;
       case 's':
       case 'S':   result = num1 - num2;
         printf(" %d - %d = %d\n", num1,num2,result);
         break;
       case 'm':
       case 'M':   result = num1 * num2;
         printf(" %d * %d = %d\n", num1,num2,result);
         break;
       default :   printf("Not one of the proper choices.\n");
     }
     return 0;
 }
```

Output

Enter your choice (A, S or M) => ***S***

Enter two numbers: ***9 5***

9 – 5 = 4

If-else: Example

```
/* Arithmetic (A,S,M) computation of two user numbers */
#include <stdio.h>
 int main() {
     char choice;   int num1, num2, result;
     printf("Enter your choice (A, S or M) => ");
     scanf("%c", &choice);
     printf("Enter two numbers: ");
     scanf("%d %d", &num1, &num2);
     if ( (choice == 'a')  || (choice == 'A'))    {
         result = num1 + num2;
         printf(" %d + %d = %d\n", num1,num2,result);
      }
      else if ((choice == 's')  || (choice == 'S'))
         result = num1 - num2;
         printf(" %d - %d = %d\n", num1,num2,result);
      }
      else if ((choice == 'm')  || (choice == 'M') )
         result = num1 * num2;
         printf(" %d * %d = %d\n", num1,num2,result);
      }
        else printf("Not one of the proper choices.\n");
     return 0;
  }
```

Note:

1. The **switch** statements can be replaced by **if-else-if-else** statements.
2. Also, the labels in the **switch** construct must be **constant integral values** which make it not so flexible.

Output

Enter your choice (A, S or M) => ***S***
Enter two numbers: ***9 5***
9 – 5 = 4

Omitting Break Statement

```
switch (choice) {
        case 'a':
        case 'A':   result = num1 + num2;
                    printf("%d + %d = %d“, num1, num2, result);
        case 's':
        case 'S':   result = num1 - num2;
                    printf("%d - %d = %d“, num1, num2, result);
        case 'm':
        case 'M':   result = num1 * num2;
                    printf("%d * %d = %d" + num1, num2, result);
                    break;
        default:
                    printf("Not a proper choice!");
}
```

No break

No break

Program Input and Output

……

Your choice (A, S or M) => *A*

Enter two numbers: ***9 5***

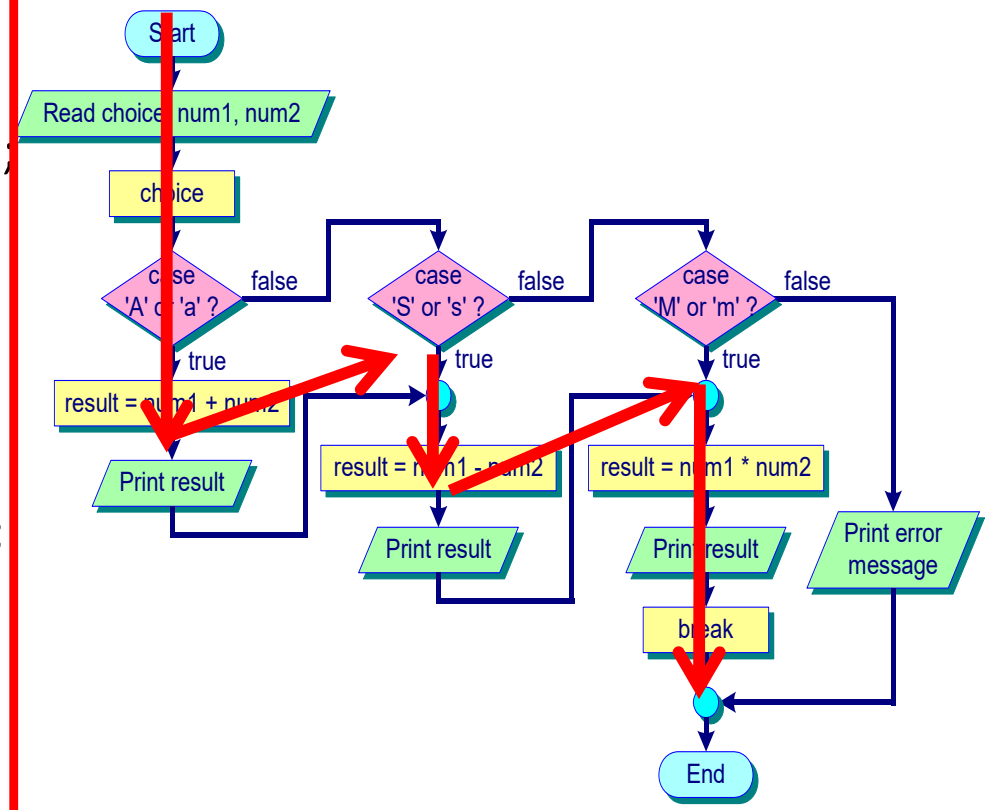
-- **WHAT WILL BE THE OUTPUTS?**

Omitting Break - Fall Through

```
switch (choice) {
    case 'a':
    case 'A':   result = num1 + num2;
      printf("%d + %d = %d", num1, num2, result);
    case 's':
    case 'S':   result = num1 - num2;
      printf("%d - %d = %d", num1, num2, result);
    case 'm':
    case 'M':  result = num1 * num2;
      printf("%d * %d = %d", num1, num2, result);
      break;
    default:
      printf("Not a proper choice!");
}
```

No break

No break



Program Input and Output

……

Your choice (A, S or M) => *A*

Enter two numbers: ***9 5***

-- **WHAT WILL BE THE OUTPUTS?**

9 + 5 = 14

9 – 5 = 4

9 * 5 = 45

- **Fall through** – if no **break** statement for the case block, execution will **continue** with the statements for the subsequent labels until a break statement or the end of switch statement is reached.

Conditional Operator

- The conditional operator is used in the following way:

```
expression_1 ? expression_2 : expression_3
```

The **value** of this expression depends on whether *expression_1* is true or false.

if expression_1 is true

the value of the expression is that of *expression_2*

else

the value of the expression is that of *expression_3*

<u>For example:</u>

```
max = (x > y) ? x : y;      <==>      if (x > y)
                                          max = x;
                                      else
                                          max = y;
```

Conditional Operator: Example

```
/* Example to show a conditional expression */
#include <stdio.h>
int main()
{
    int choice;    /* User input selection */
    printf("Enter a 1 or a 0 => ");
    scanf("%d", &choice);

    choice ? printf("A one.\n") : printf("A zero.\n");

    return 0;
}
```

Output

```
Enter a 1 or a 0 => 1
A one.
Enter a 1 or a 0 => 0
A zero.
```

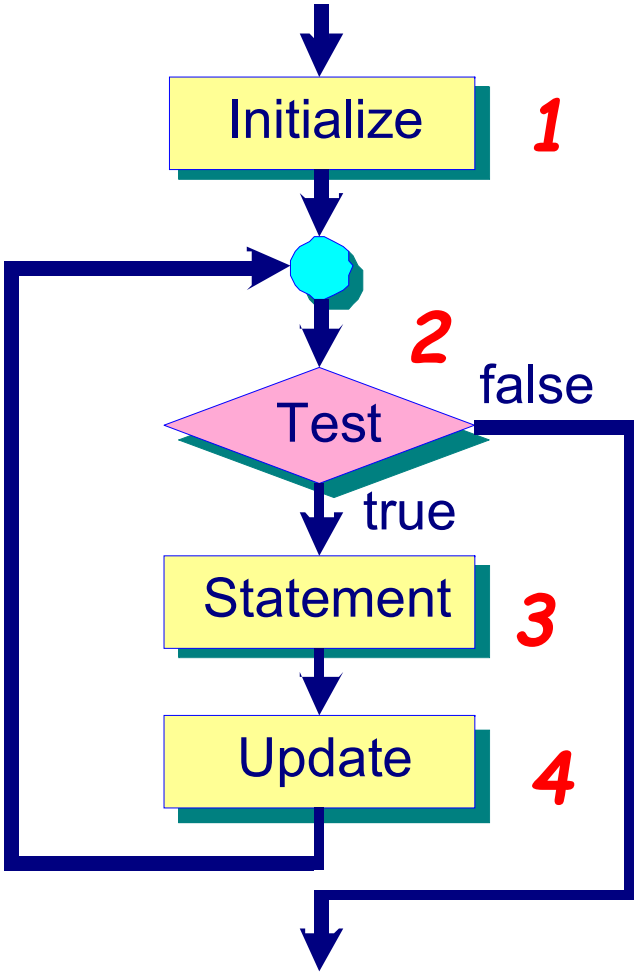
Control Flow

- Relational Operator and Logical Operator
- Branching: if, if…else, if….else if….else Statements; Nested if Statements; The switch Statement; Conditional Operators
- **Looping: for, while, do-while; Nested Loops; The break and continue Statements**

Looping

There are mainly 4 basic steps to construct a loop:

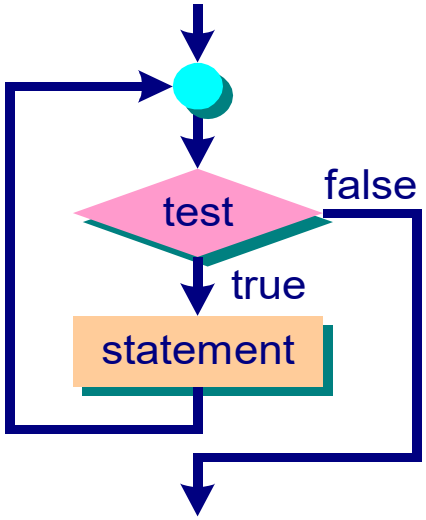
1. **Initialize** – We need to define **Loop Control Variable** and initialize the loop control variable.
2. **Test** – This evaluates the test condition which involves the loop control variable. If the **test** evaluation is true, then the loop body is executed, otherwise the loop is terminated.
3. **Loop body (or Statement)** – The loop body is executed if test evaluation is true.
4. **Update** – Typically, loop control variable is **modified** through the execution of the loop body on Update. It can then go through the **test** condition again to evaluate whether to repeat the loop body again.



In C, there are three types of looping constructs: **for, while, do-while.**

The while Loop

```
while (test)
        statement
```



Sentinel-controlled Loop

```
/* sum up a list of integers.
The list of integers is terminated by -1.  */
#include <stdio.h>
int main()
{
        int sum=0, item;
        printf("Enter the list of integers:\n");
        scanf("%d", &item);
        while (item != -1)  {
                sum += item;
                scanf("%d", &item);
        }
        printf("The sum is %d\n", sum);
        return 0;
}
```

Output

Enter the list of integers:

1 8 11 24 36 48 67 –1

The sum is 195

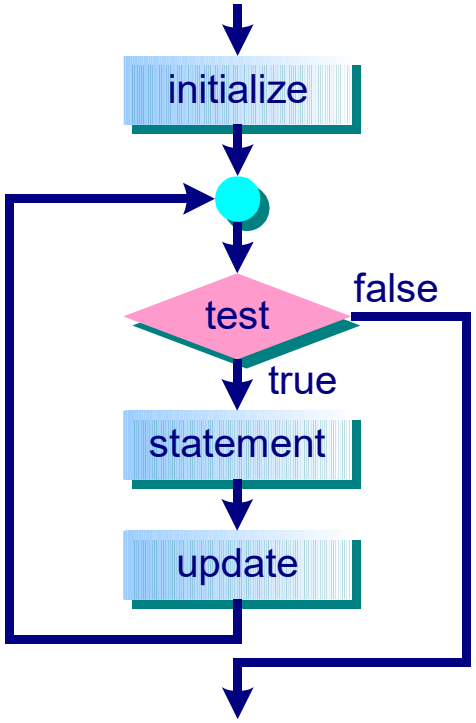
Enter the list of integers:

-1

The sum is 0

The for Loop

```
for (initialize; test; update)
        statement;
```



Counter-controlled Loop

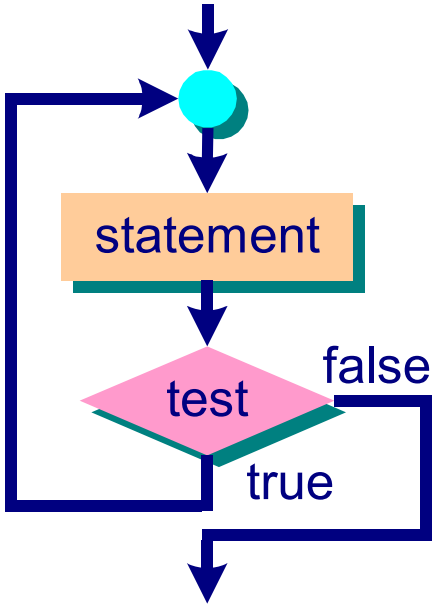
```
/* display the distance a body falls in feet/sec
for the first n seconds, n input by the user
*/
#include <stdio.h>
#define ACCELERATION 32.0
int main()
{
    int timeLimit, t;
    int distance;   /* Distance by the falling body. */
    printf("Enter the time limit (sec):");
    scanf("%d", &timeLimit);
    for (t = 1;     t <= timeLimit;     t++)  {
        distance = 0.5 * ACCELERATION * t * t;
        printf("Dist after %d sec is %d \
            feet.\n", t, distance);
    }
    return 0;
}
```

Output

```
Enter the time limit (sec): 5
Dist after 1 sec is 16 feet.
Dist after 2 sec is 64 feet.
Dist after 3 sec is 144 feet.
Dist after 4 sec is 256 feet.
Dist after 5 sec is 400 feet.
```

The do-while Loop

```
do
    statement;
while (test);
```



for Menu-driven Applications

```
/* Menu-Based User Selection */
#include <stdio.h>
int main()
{
    int  input;    /* User input number. */
    do  {
        /* display menu */
        printf("Input a number >= 1 and <= 5:  ");
        scanf("%d",&input);
        if (input > 5 || input < 1)
            printf("%d is out of range.\n", input);
    }  while (input > 5 || input < 1);
    printf("Input = %d\n", input);
    return 0;
}
```

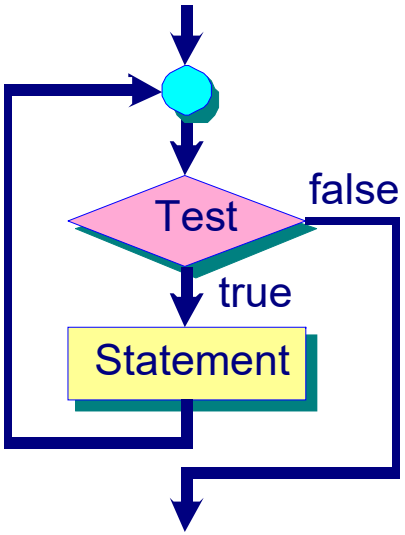
Output

```
Input a number >= 1 and <= 5: 6
6  is out of range.
Input a number >= 1 and <= 5:  5
Input = 5
```

Loops Comparison

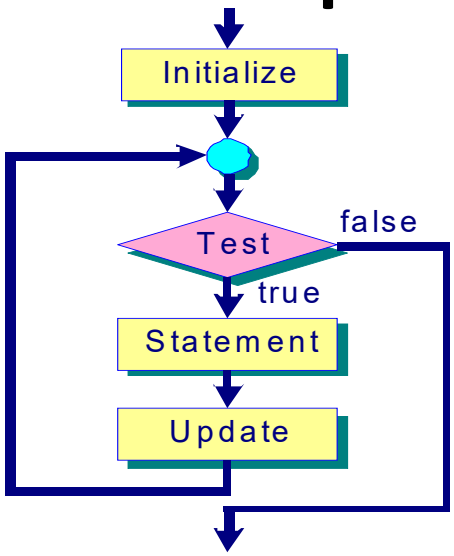
- **do-while** loop is different from the ***for*** and ***while*** loops:
  - In the *while* or *for* loop – the condition *Test* is performed **before** executing the Statement, i.e. the loop might **not** be executed even once.
  - In the *do-while* loop - the condition *Test* is performed **after** executing the Statement every time, i.e. the loop body will be executed **at least once**.

while loop



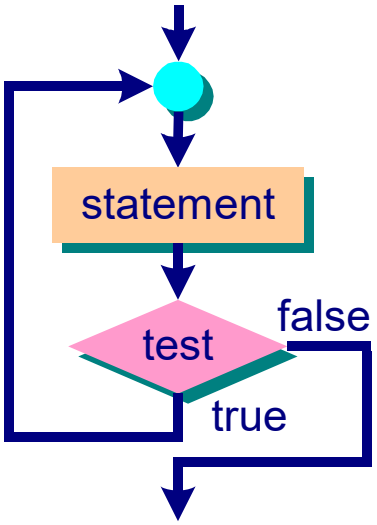
Sentinel-controlled Loop

for loop



Counter-controlled Loop

do-while loop



Menu-driven Applications

The break Statement

- The **break** statement alters the flow of control inside a **for**, **while** or **do-while** loop (as well as the **switch** statement).
- Execution of break causes **immediate termination** of the **innermost enclosing loop** or switch statement.

```
/* summing up positive numbers
from a list of up to 8 numbers */
#include <stdio.h>
int main() {
    int i;
    float   data, sum=0;
    printf("Enter 8 numbers: ");
    /* read 8 numbers  */
    for (i=0; i<8; i++) {
        scanf("%f", &data);
        if (data < 0.0)
            break;
        sum += data;
    }
    printf("The sum is %f\n",sum);
    return 0;
}
```

Output

Enter 8 numbers: *3 7 -1 4 -5 8 3 1*

The sum is 10.000000

The continue Statement

- The **continue** statement causes termination of the current iteration of a loop and the control is immediately passed to the test condition of the **nearest** enclosing loop.
- All subsequent statements after the continue statement are **not** executed for this particular iteration.

```
/* summing up positive numbers
from a list of 8 numbers */
 #include <stdio.h>
int main() {
      int i;
      float   data, sum=0;
      printf("Enter 8 numbers: ");
      /* read 8 numbers  */
      for (i=0; i<8; i++) {
              scanf("%f", &data);
              if (data < 0.0)
                  continue;
              sum += data;
      }
      printf("The sum is %f\n",sum);
      return 0;
}
```

- **Note**: the **break** statement terminates the execution of the loop and passes the control to the next statement immediately after the loop.

Output

Enter 8 numbers: *3 7 -1 4 -5 8 3 1*

The sum is 26.000000

Nested Loops

- A loop may appear inside another loop. This is called a **nested loop**. We can nest as **many levels** of loops as the hardware allows. And we can nest **different types** of loops.
- Nested loops are commonly used for applications that deal with 2-dimensional objects such as tables, charts, patterns and matrices.

```
/* count the number of different strings of a, b, c */
#include <stdio.h>
int main()
{
    char i, j;          /* for loop counters */
    int num = 0;        /* Overall loop counter */
    for (i = 'a'; i <= 'c'; i++) {
        for (j = 'a'; j <= 'c'; j++) {
            num++;
            printf("%c%c ", i, j);
        }
        printf("\n");
    }
    printf("%d different strings of letters.\n", num);
    return 0;
}
```

Output

aa ab ac

ba bb bc

ca cb cc

9 different strings of letters.

Thank you !!!