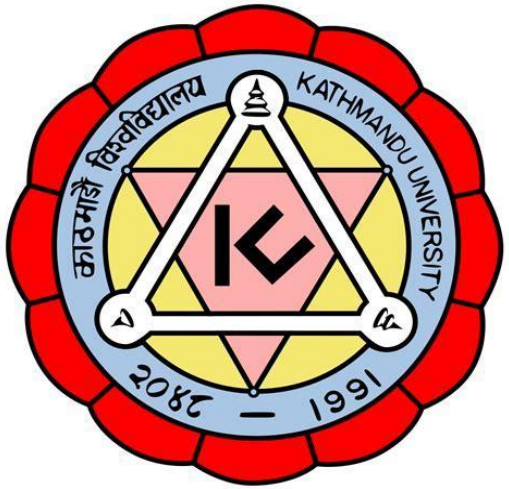
KATHMANDU UNIVERSITY SCHOOL OF MANAGEMENT

BBIS

COM 102 : 3 Credit Hours



6. Control structures and statements in C contd ..

Loops Control Statements

A loop or *repetition structure* goes through a block of code either a predefined number of times, or until something occurs to cause the program to break out of the loop.

Loops

- used for the program to be executed repeatedly while expression is true.
- When a block of code needs to be executed several number of times.
- When the expression becomes false ,the loop terminates and the control passes on to the statement following the loop.
- Generally a looping process involves:
  - Control variable initialization
  - Condition evaluation
  - Loop body execution
  - Control variable updation

Loop types

1. Entry Controlled Loop:-

Also known as pre-checking loop. Before executing the loop, the condition is checked first.

2. Exit Controlled Loop:-

Also known as post-checking loop. The condition is checked after executing the loop.

S.No	Loop type	Description	Remarks
1.	While loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.	Enrty Controlled
2.	For loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.	Enrty Controlled
3.	do...while loop	It is more like a while statement, except that it tests the condition at the end of the loop body.	Exit Controlled
4.	Nested loops	You can use one or more loops inside any other while, for, or do..while loop.	

Table 1: Loop types and its description

In programming, above loop types are considered as controlled statements that can regulate the flow of the program execution.

Loop Control statement (Jump statements)

S.No	Control Satement	Description
1.	break	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2.	continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3.	goto	Transfers control to the labeled statement.

Table 2: Loop control statement and its description

...

- Every loop consists of two sections:
  - **Loop body**
    - **A block that contains the intended iterative statements.**
  - **Control statement**
    - **A condition that dictates how long a loop would be iterated.**

- While Loop:

```
while(condition) // Control statement
{
    // Loop body
}
```

- Do-While Loop:

```
do {
    // Loop body
}While(condition)  // Control statement
```

- For Loop:

```
for(controlVariableInitialization; condition; increment/decrement) // Control statement
{
    // Loop body
}
```

While statement

- In a while loop, a block of code gets executed until the condition satisfies.
- **Syntax of while loop in C:-**

```
While(condition) // Control statement
{
    // Loop body
    a++;
}
```

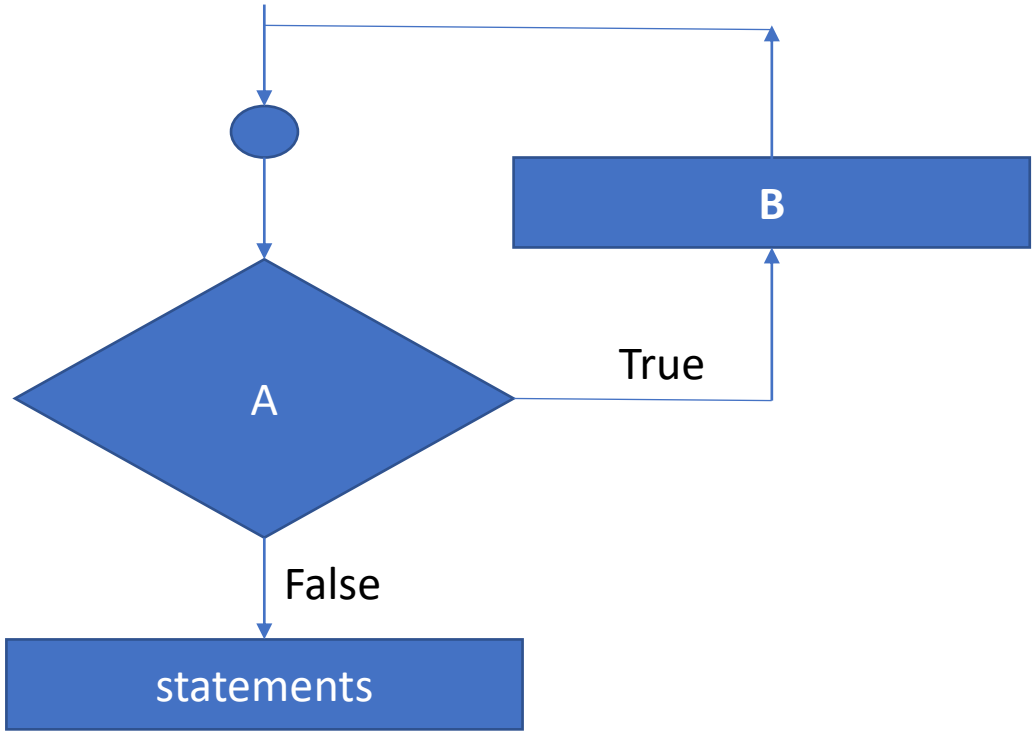
- Working of while loop:
  - First the while loop checks for the condition.
  - If the condition is true, the statements inside the body of the while loop get executed until the condition is false.
  - If the condition is false, the while loop terminates.

Flowchart of While loop

while(A=True) do

B

end while



Example

```
#include<stdio.h>
int main(){
	printf(“Multiplication table of 10:\n");
	int i=1,number=10;
	while(i<=10) //control-statements
	{
		printf("%d * %d = %d \n", number, i,
		(number*i));
		i++;
	}
	return 0;
}
```

```
main()
{
int a;
a = 0;
while (a <= 100)
{
	printf(“%d degrees F = %d degrees C\n", a, (a - 32) * 5 / 9);
	a = a + 10;
}
}
```

```
main( )
{
	int i = 1,sum=0;
	while(i <=100 ){
		sum=sum+i;
		printf(“Sum=%d \t & i=%d\n”,sum,i++);
	}
}
```

```
main( )
{
	int i = 1,sum=0;
	while(i <=100 ){
		sum=sum+i;
		printf(“Sum=%d \t & i=%d\n”,sum,i=i+5
		);
	}
}
```

```
main( )
{
	int i = 1,sum=0;
	while(i <=100 ){
		if(i%2==0)
		{
			sum=sum+i;
		}
		printf(“Sum=%d \t & i=%d\n”,sum,i++);
	}
}
```

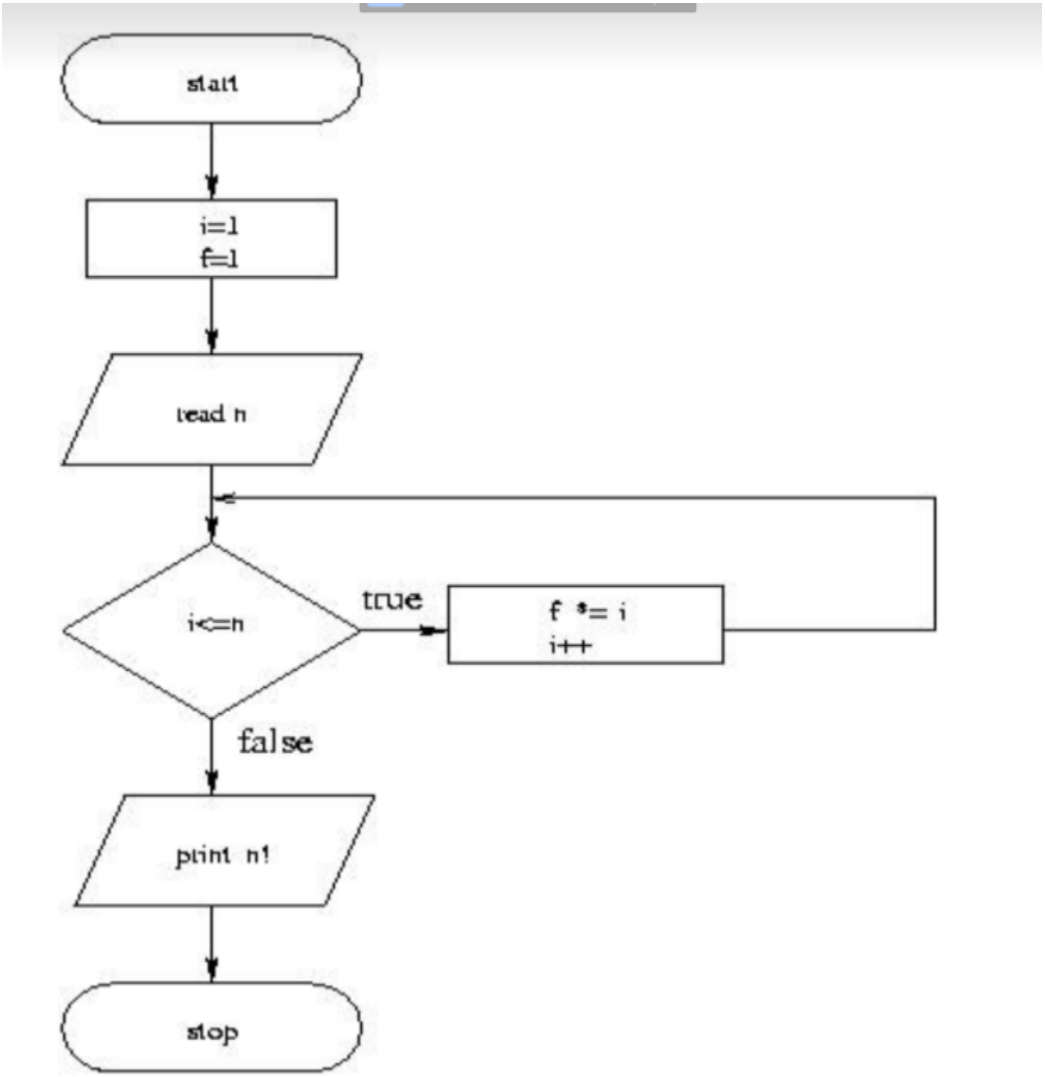
```
main( )
{
	int i = 1,sum=0;
	while(i <=100 ){
		if(i%2==0 && i%5==0)
		{
			sum=sum+i;
		}
		printf(“Sum=%d \t & i=%d\n”,sum,i++);
	}
}
```

Example:

- Calculating a factorial 5!. The factorial n! is defined as n*(n-1)!

```
main() {
        int i, f, n;                    // declaration
        i = 1;                          //initialization
        f = 1;                          //initialization
        /*Processing */
        printf("Please input a number\n");
        scanf("%d", &n);
        while (i <= n) {
                f *= i;
                i++;
        }

        /* termination */
        printf("factorial %d! = %d\n", n, f);
}
```



Control of Repetition

1. Sentinel-controlled repetition
2. Counter-controlled repetition

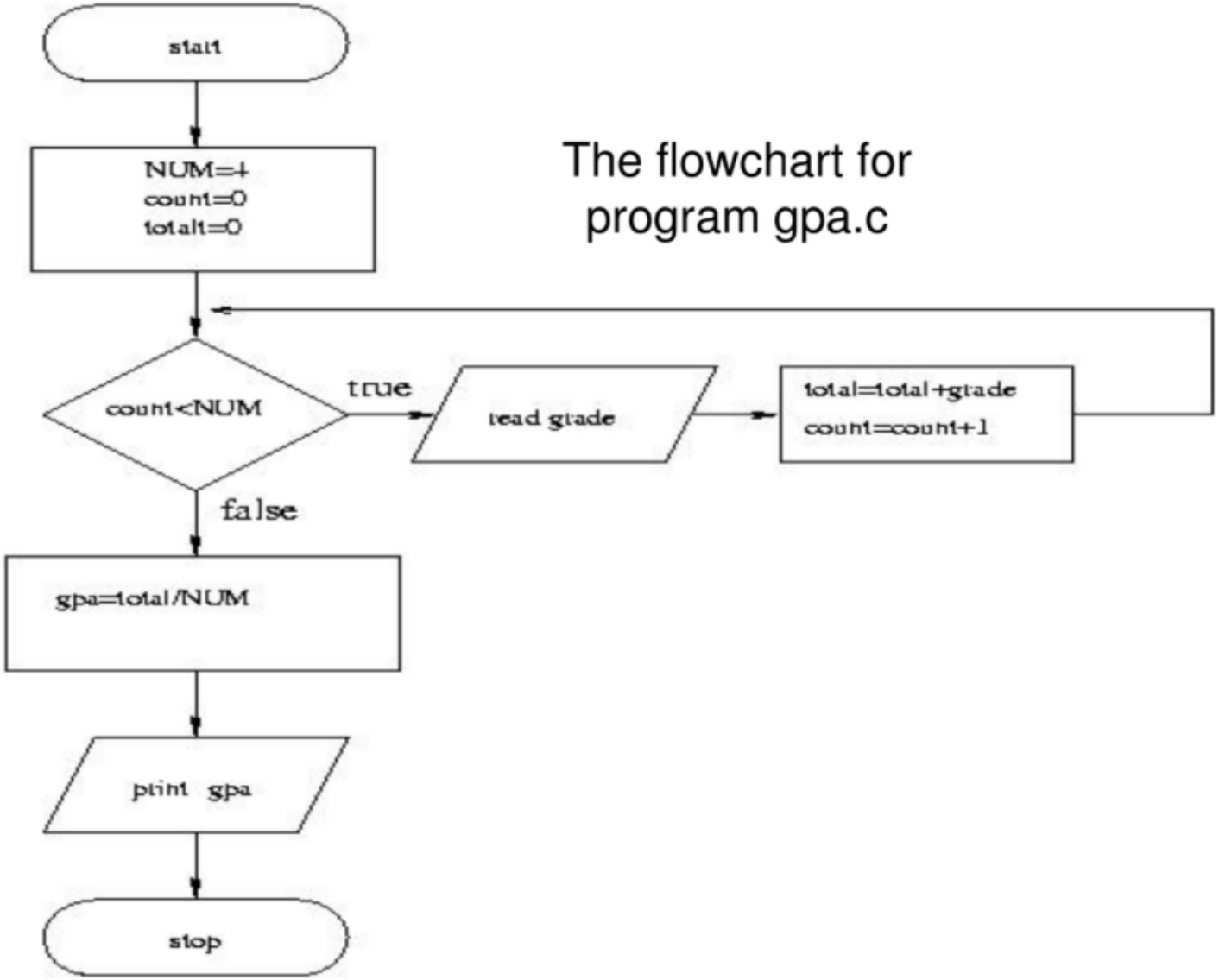
1. Counter-controlled repetition

- Loop repeated until counter reaches a certain value.
- Definite repetition: number of repetitions is known

Example:

- A student takes four courses in a quarter. Each course will be assigned a grade with the score from 0 to 4.
- Develop a C program to calculate the grade point average (GPA) for the quarter.

Flowchart



The flowchart for program gpa.c

```
#define NUM 4
main() {
        int count;
        double grade, total, gpa;
        count = 0;
        total = 0;
        /* processing */
        while(count < NUM) {
                printf("Enter a grade: ");
                scanf("%lf", &grade);
                total += grade;
                count++;
        }
        /* termination */
        gpa = total/NUM;
        printf("The GPA is: %f\n",
        gpa);
}
```

Another Example

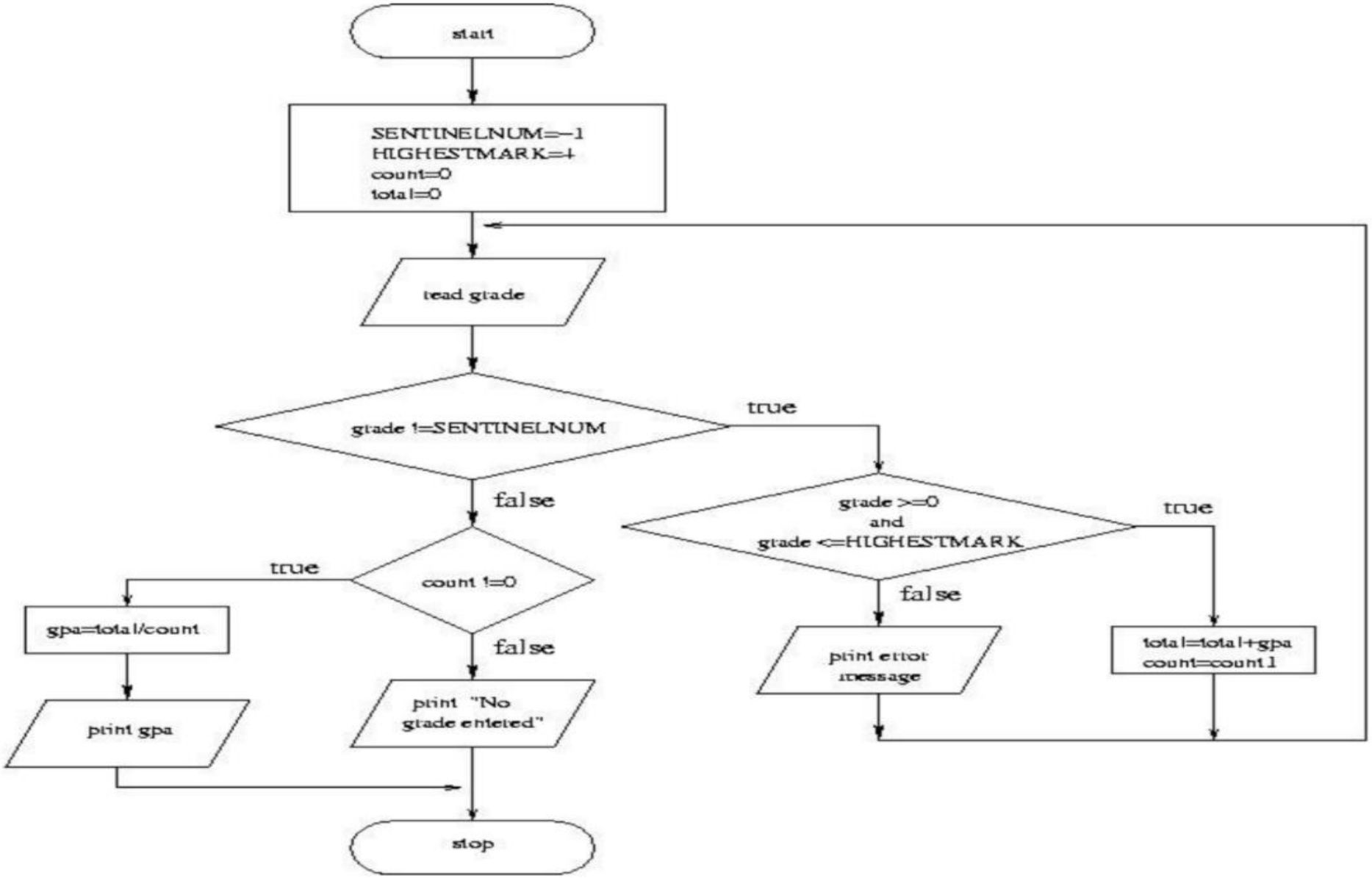
Display the character and its' ASCII value for all lower case characters.

```
#include <stdio.h>
int main()
{
   //1
   int start = 'a';
   //2
   while (start <= 'z')
   {
      //3
      printf("%c : %d\n", start, start);
      //4
      start++;
   }
   return 0;
}
```

2. Sentinel-controlled repetition

- Loop repeated until the sentinel (signal) value is entered.
- Indefinite repetition: number of repetitions is unknown when the loop begins execution.
- An example of when we would use sentinel-controlled repetition is when we are processing data from a file and we do not know in advance when we would reach the end of the file.
- Example:
  - Develop a GPA calculation program that will process grades with scores in the range of [0, 4] for an arbitrary number of courses.

Flowchart


start
SENTINELNUM=-1
HIGHESTMARK=1
count=0
total=0
read grade
grade !=SENTINELNUM
true
false
grade >=0
and
grade <=HIGHESTMARK
true
false
print error
message
total=total+gpa
count=count 1
count !=0
true
false
gpa=total/count
print gpa
print "No
grade entered"
stop

Source-code:

```
#define SENTINEL_NUM  -1
#define HIGHESTMARK  4
main() {
        int count;
        double grade, total, gpa;
        /* initialization */
        count = 0;
        total = 0;
        printf("Enter a grade [0, %d] or %d to end: ",

        HIGHESTMARK, SENTINEL_NUM);

        scanf("%lf", &grade);
        while((int)grade != SENTINEL_NUM) {
        if(0 <= grade && grade <= HIGHESTMARK) {
        total += grade;
        count++;
        }
        else {
        printf("Invalid grade %c\n", '\a');
        }
        printf("Enter a grade [0, %d] or %d to end: ",

        HIGHESTMARK, SENTINEL_NUM);

        scanf("%lf", &grade);
}
```

```
        /* termination */
        if(count != 0) {
                gpa = total/count;
                printf("The GPA is: %f\n",

                gpa);
        }
        else
        printf("No grade entered.\n");

}
```

Output:
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: 3.3
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 10
Invalid grade
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: -1
The GPA is: 3.740000

Do-while loop

- Syntax:

```
do {
    // Loop body
}While(condition)  // Control statement
```

- The evaluation of the controlling expression takes place after each execution of the loop body.
- The loop body is executed repeatedly until the return value of the controlling expression is equal to 0.

Example

```
a=10
B=10
do {
    printf("%d\n", 10);
    a = a + 1;
} while (a < b);
```

Syntax:

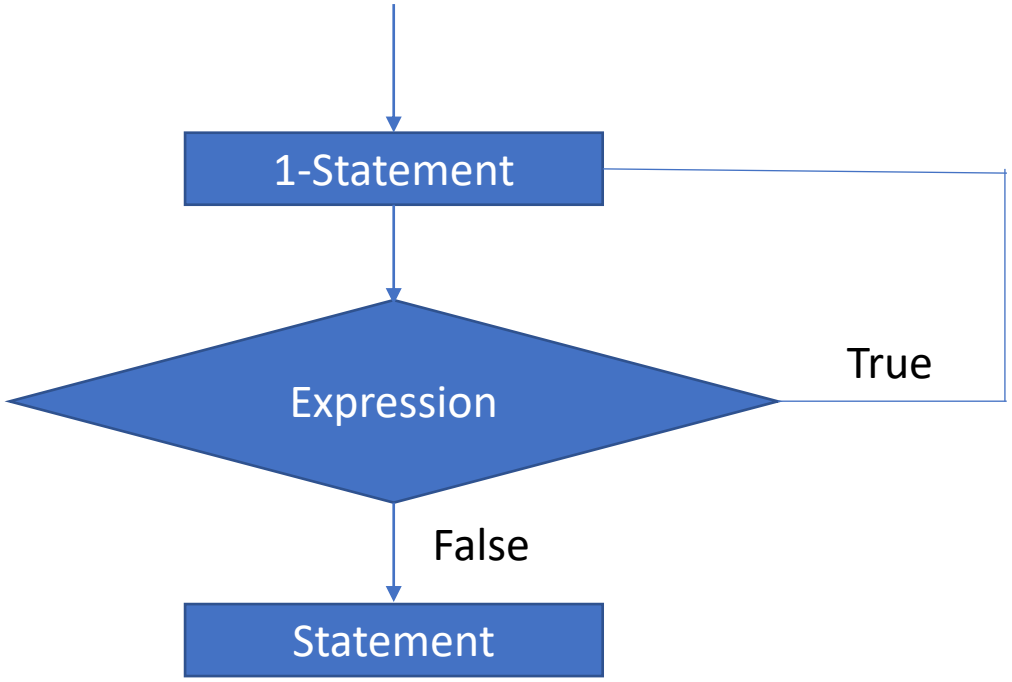
```
do {
    // Loop body
}While(condition)  // Control
statement
```

- Operation: Similar to the while control except that statement is executed before the expression is evaluated.
- This guarantees that statement is always executed at least one time even if expression is FALSE.

Flowchart of do-while loop

Syntax:

```
do
        statement
While(expression);
```



Example

– What is the output of the following example?

```
int i =4;
do {
      printf(“%d ”, 4);
      i++;
} while(5< 5);
```

Output:
0 1 2 3 4

```
int i = 10;
do {
      printf(“%d ”, 10);
      i++;
} while(11< 5);
```

Output: 10

Some programs to practice using While and do-while

- WAP to find sum of ten numbers.
- WAP to display all numbers between 1 to 1000 that are perfectly divisible by 10.
- WAP to display all numbers between 2000 to 100 that are perfectly divisible by 13 and 15 and 17 and 19.
- WAP to find sum of all numbers between 500 to 1500 that are perfectly divisible by 3 and 5 and 15 and 45.
- WAP to find sum of all numbers between 10000 to 1000 that are perfectly divisible by 3 and 7 and 9 and 42.
- WAP to find factorial of given number – N! = N * (N-1)!
- WAP to check whether a given number is prime or not.
- Hint– A number is prime if it is divisible by 1 and the number itself only
- WAP to generate all prime numbers between 100 to 1.

The for Statement

Structure: for(expr1; expr2; expr3)

```
        {
                statement;
        }
```

- simply a shorthand way of expressing a while statement:

```
expr1;
while(expr2)
{
    statement;
    expr3
}
```

```
for(controlVariableInitialization; condition; increment/decrement)
// Control statement
{
        // Loop body
}
```

Common errors to avoid

- Putting = when you mean == in an if or while statement
- Forgetting to increment the counter inside the while loop – If you forget to increment the counter, you get an infinite loop(the loop never ends).
- Accidentally putting a ; at the end of a for loop or if statement so that the statement has no effect

For example:

```
for (x=1; x<10; x++);
printf("%d\n",x);
```

only prints out one value because the semicolon after the

for statement acts as the one line the for loop executes.

…

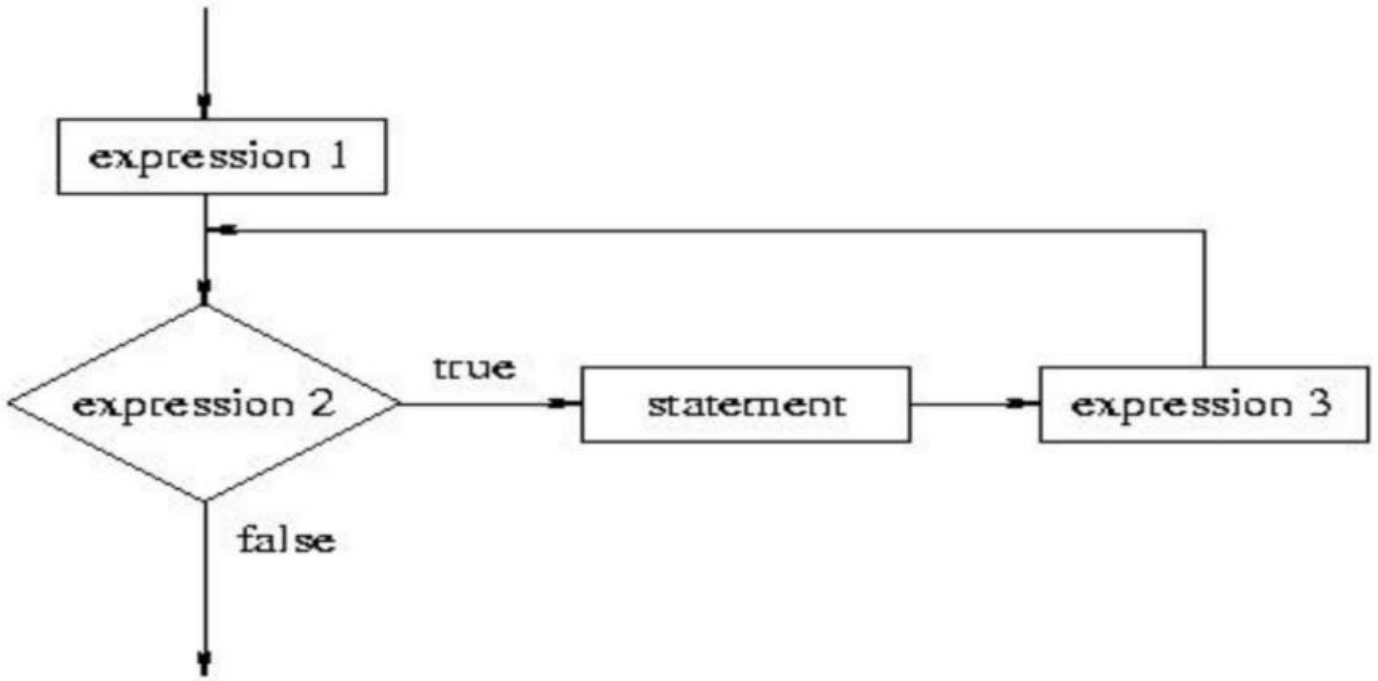
Structure: for(expr1; expr2; expr3)

```
{
        statement;
}
```

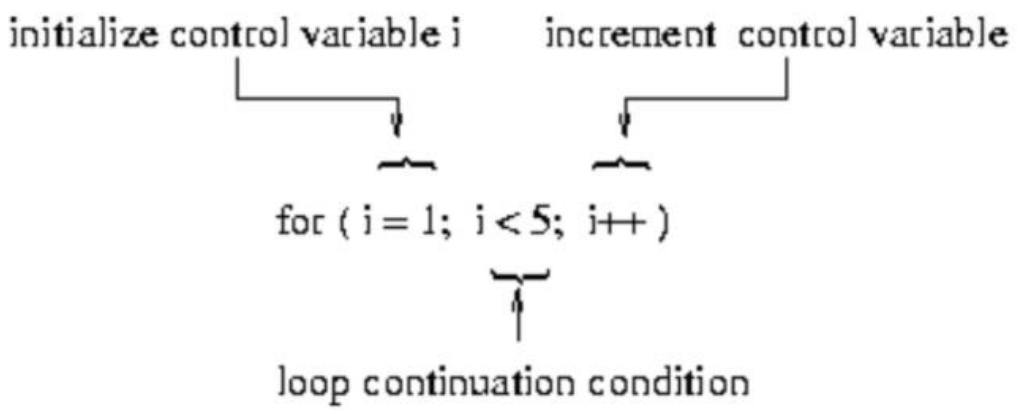
- The expression expression1 is evaluated as a void expression before the first evaluation of the controlling expression.
- The expression expression2 is the controlling expression that is evaluated before each execution of the loop body.
- The expression expression3 is evaluated as a void expression after each execution of the loop body.

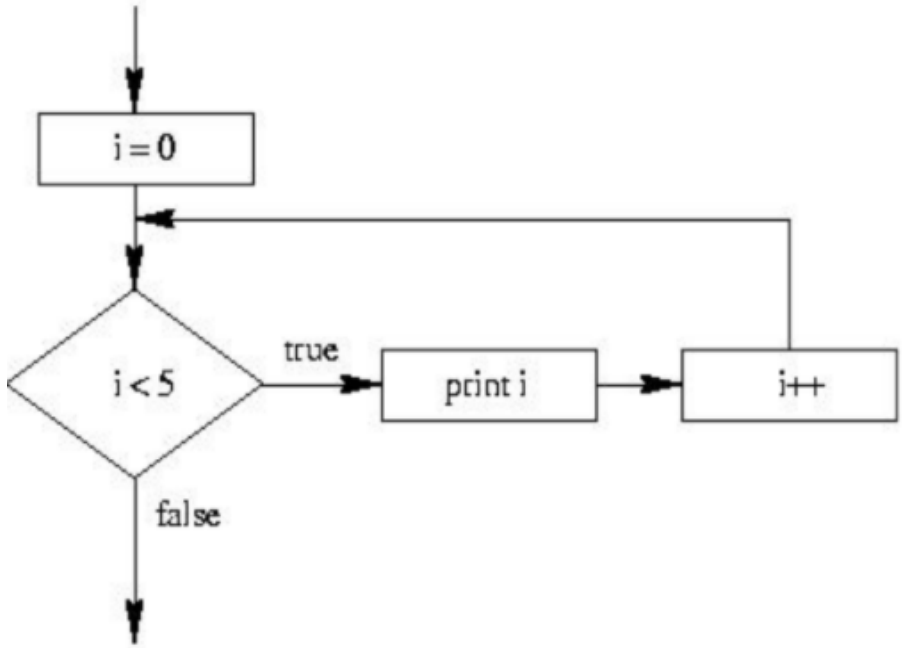
Flowchart of a for loop

```
for(expression1; expression2; expression3)
        statement
```



```
int i;
for(i = 0; i < 5; i++)
{
        printf("%d ", i);
}
```





Output: 1 2 3 4

Example:

- Calculating a factorial 5!.

```
main() {

        int i, f, n;
        printf(“Please input a number\n”);
        scanf(“%d”, &n);
        for(i=1, f=1; i<=n; i++)
        {
                f = f*i;
        }
        printf(“factorial %d! = %d\n", n, f);

}
```

Jump Statements

- Break Statements
  - The break statement provides an early exit from the for, while, do-while, and for each loops as well as switch statement.
  - A break causes the innermost enclosing loop or switch to be exited immediately.
  - Example:

```
int i;
for(i=0; i<5; i++)
{
        if(i == 3)
        {
                break;
        }
        printf("%d", i);
}
```

Output :0 1 2

Continue statements

- It helps skip the remaining part of the loop.
- It continues to execute the next iteration.
- It causes early execution of the next iteration of the enclosing loop.
- It can’t be used with ‘switch’ and ‘label’ since it is not compatible.
- A continue statement should only appear in a loop body.

```
int i;
for(i=0; i<5; i++)
{
        if(i == 3)
        {
                continue;
                i+=3;
        }
        printf("%d", i);
}
```

Output : 0 1 2 6 4

goto statements

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

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 to avoid them.

Syntax:

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

Example:

```
#include <stdio.h>
 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;
}
```

Result:
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

Nested Loop

- Nested loop = loop inside loop

- Program flow
- The inner loops must be finished before the outer loop resumes iteration.

Q. Write a program to print a multiplication table.

```
      1   2   3   4   5   6   7   8   9  10
 ---------------------------------------------
 1|   1
 2|   2   4
 3|   3   6   9
 4|   4   8  12  16
 5|   5  10  15  20  25
 6|   6  12  18  24  30  36
 7|   7  14  21  28  35  42  49
 8|   8  16  24  32  40  48  56  64
 9|   9  18  27  36  45  54  63  72  81
10|  10  20  30  40  50  60  70  80  90 100
 ---------------------------------------------
```

Some Programs to do.

- WAP to generate multiplication table of any given number
- WAP to generate multiplication table of number 1 to 5.
- WAP to check whether a given program in Armstrong ornot.
  - 153 is a Armstrong number because $1^3+5^3+3^3=153$
- WAP to check whether a number is strong or not.
  - 145 is a strong number because 1! + 4! +5 ! = 145
- WAP to check whether a number is perfect or not.
  - 28 is a perfect number because 1+2+4+7+14 = 28
- WAP to find GCD (HCF) of given two numbers.
- WAP to find LCM of given numbers.

Any Queries???