Introduction to C Programming

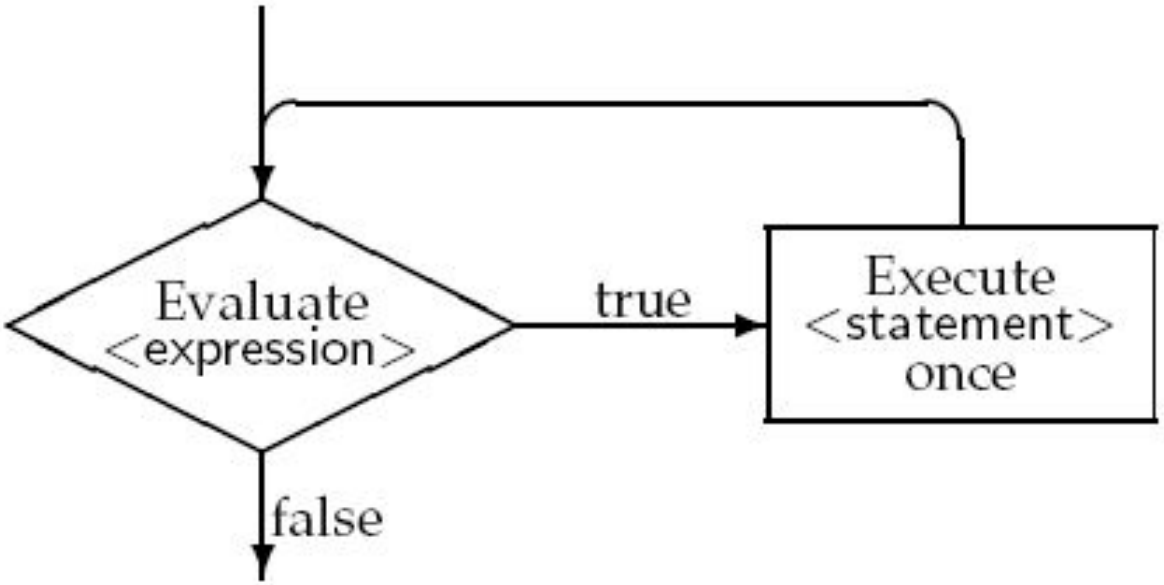
Section 4

Repetition

- Example: Write a program that read 3 integer and compute average
  - It is easy. 3 scanf, an addition, a division and, a printf
- Example: Write a program that read 3000 integer and compute average
  - ?? 3000 scanf !!!
- Example: Write a program that read n integer and compute average
  - N ??? scanf
- Repetition in algorithms
-

While statement

```
while ( <expression> )

    <statements>
```



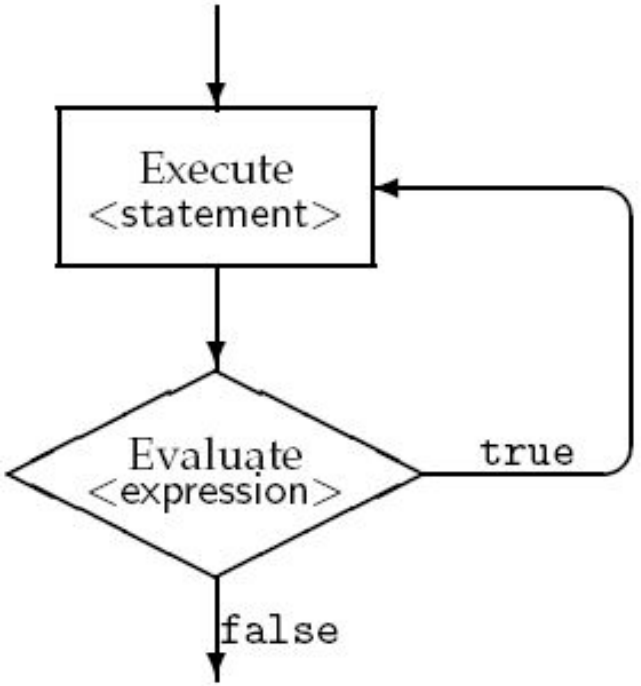
برنامه ای بنویسید که عدد n را از کاربر بگیرد و اعداد 0 تا n را چاپ کند.

```c
#include   <stdio.h>

int main(void){
  int n, number;
  number = 0;
  printf("Enter n: ");
  scanf("%d", &n);
  while(number <= n){
     printf("%d \n", number);
     number++;

  }
  return 0;
}
```

```c
number = -1;
while(++number <= n)
   printf("%d \n", number);
```

Do-while statement (not advised)

```
do
    <statements>
while (<expression>);
```



Nested loops

- <statement> in loops can be loop itself

```
while(<expression0>)
    for(<expression1>; <expression2>;<expression3>)

        <statements>


for(<expression1>; <expression2>;<expression3>)

     do
        <statements>
     while(<expression>);
```

```
#include <stdio.h>
int

main(void){

    int i, j, n;

   printf("Enter n: ");
   scanf("%d", &n);
    i = 1;
    while(i <= n){
      for(j = 0; j < i; j++)
            printf("*");
      printf("\n");
      i++;
   }
    return 0;
}
```

break statement

- The `break` and `continue` statements are used to alter the flow of control.
- The `break` statement, when executed in a `while`, `for`, `do…while` or `switch` statement, causes an immediate exit from that statement.
- Program execution continues with the next statement.

break statement

- Exit from loop based on some conditions

```
do{
   scanf("%d", &a);
   scanf("%d", &b);
   if(b == 0)
      break;
   res = a / b;
   printf("a /= %d\n", res);
}while(b > 0);
```

continue statement

- Jump to end of loop and continue repetition
- The continue statement, when executed in a while, for or do…while statement, skips the remaining statements in the body of that control statement and performs the next iteration of the loop.

```
do{
    scanf("%f", &a);
    scanf("%f", &b);
    if(b == 0)
        continue;
    res = a / b;
    printf("a / b= %f\n", res);
}while(a> 0);
```

```c
/* Fig. 4.12: fig04_12.c
   Using the continue statement in a for statement */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int x; /* counter */

   /* loop 10 times */
   for ( x = 1; x <= 10; x++ ) {

      /* if x is 5, continue with next iteration of loop */
      if ( x == 5 ) {
         continue; /* skip remaining code in loop body */
      } /* end if */

      printf( "%d ", x ); /* display value of x */
   } /* end for */

   printf( "\nUsed continue to skip printing the value 5\n" );
   return 0; /* indicate program ended successfully */
} /* end function main */
```

Fig. 4.12 | Using the `continue` statement in a `for` statement. (Part 1 of 2.)

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Fig. 4.12 | Using the `continue` statement in a `for` statement. (Part 2 of 2.)

Common bugs and avoiding them

- Loop should terminate
  - E.g., in `for` loops, after each iteration, we should approach to the stop condition

```
for(i = 0; i < 10; i++) //OK
for(i = 0; i < 10; i--) //Bug
```

- Initialize loop control variables

```
int i;

for( ; i < 10; i++) //Bug
```

Common bugs and avoiding them

- Don’t modify **`for`** loop controller in loop body

```
for(i = 0; i < 10; i++){
    ...
    i--; //Bug
```

- Take care about wrong control conditions
- < vs. <=
- = vs. ==

```
int b = 10;
while(a = b){//it means while(true)
  scanf("%d",&a)
  …
```