Computer Programming using C

Flow Control-Part II

Instructor: HOU, Fen

2025

The Use of the function scanf()

- It is used for input from keyboard
- The scanf() statement

```
scanf("%d", &x);
```

  - The above scanf() statement has two arguments which are separated by a comma;
  - The first argument is called the control string, where the conversion character (e.g.,d) defines the format that the input is interpreted.
  - In this example, the control string contains the format specifier %d, which causes the input typed at the keyboard to be interpreted as a decimal integer.
  - The second argument is address. The symbol & is the address operator; &x causes the value of x to be stored at the address of the variable x.

The Use of the function scanf()

- The scanf() statement `scanf(“%d", &x);`
- The conversion characters in scanf(f) are shown in the follow table

scanf( ) conversion	
Conversion character	How characters in the input are converted
c	character
d	decimal integer
f	floating-point number (float)
lf	floating-point number (double)
Lf	floating-point number (long double)
s	string

1. Repetition – The `while` Statement

```
while (expr)
      statement;
next_statement;
```

- As long as (i.e. while) **`expr`** is true (non-zero), repeatedly execute **`statement`**.

- When **`expr`** evaluates to false (zero), execute **`next_statement`**.

Examples

```
int i = 1, sum = 0;

while (i <= 4) {
  sum += i;
  i++;
}
printf ("%d\n", sum);
```

Examples

```
int i = 1, sum = 0;

while (i <= 4) {
  sum += i;
  i++;
}
printf ("%d\n", sum);
```

Output **10**

	Before		Condition	After	
	`i`	`sum`	`i <= 4`	`i`	`sum`
1st	1	0	true	2	1
2nd	2	1	true	3	3
3rd	3	3	true	4	6
4th	4	6	true	5	10
5th	5	10	false	/	/

```
#include <stdio.h>

int main(void)
{
    int cnt=0, n, max, x;

    printf ("How many numbers do you wish to enter? ");
    scanf ("%d", &n);
    printf ("\nEnter %d decimal numbers:\n", n);
    scanf ("%d", &x);

    max = x;
    cnt++;

    while (cnt < n) {
        scanf ("%d", &x);
        if (max < x)
            max = x;
        cnt++;
    }

    printf ("\nMaximum value: %d\n", max);

    return (0);
}
```

```c

    printf ("\nMaximum value: %d\n", max);

    return (0);
}
```

```
How many numbers do you wish to enter? 4

Enter 4 decimal numbers:
4 3 99 20

Maximum value: 99
```

```c
#include <stdio.h>

int main(void)
{
    int data, sum=0;

    scanf("%d", &data);

    while (data>=0) {
        sum += data;
        scanf("%d", &data);
    }

    printf ("The sum is %d\n", sum);

    return (0);
}
```

```
10
20
30
-1
```

```c
#include <stdio.h>

int main(void)
{
    int data, sum=0;

    scanf("%d", &data);

    while (data>=0) {
        sum += data;
        scanf("%d", &data);
    }

    printf ("The sum is %d\n", sum);

    return (0);
}
```

```
10
20
30
-1
The sum is 60
```

Example

```
#include <stdio.h>
int main(void)
{
  int number = 0;
 while (number < 5)
      {
          switch (number)
            {
              default:
                     printf("Gagusa\n");
              case 0:
                     printf("Girene\n");
                     break;
case 3:
     printf("Lefkosa\n");
     break;
case 2:
     ++number;
case -1:
     printf("Iskele\n");
     break;
case 1:
     printf("Karpaz\n");
            }
 ++number;
     }
     getchar();
     return(0);
}
```

Example

```
#include <stdio.h>
int main(void)
{
  int number = 0;
 while (number < 5)
       {
          switch (number)
              {
                 default:
                          printf("Gagusa\n");
                 case 0:
                          printf("Girene\n");
                          break;
                 case 3:
                          printf("Lefkosa\n");
                          break;
                 case 2:
                          ++number;
                 case -1:
                          printf("Iskele\n");
                          break;
                 case 1:
                          printf("Karpaz\n");
              }
 ++number;
       }
       getchar();
       return(0);
}
```

Example

```
#include <stdio.h>
int main(void)
{
   int number = 0;
 while (number < 5)
        {
              switch (number)
                   {
                         default:
                                  printf("Gagusa\n");
                         case 0:
                                  printf("Girene\n");
                                  break;
 case 3:
         printf("Lefkosa\n");
         break;
 case 2:
         ++number;
 case -1:
         printf("Iskele\n");
         break;
 case 1:
         printf("Karpaz\n");
                   }
  ++number;
        }
         getchar();
         return(0);
 }
```

Number=0

Example

Number=1

```
#include <stdio.h>
int main(void)
{
   int number = 0;
  while (number < 5)
        {
              switch (number)
                 {
                      default:
                              printf("Gagusa\n");
                      case 0:
                              printf("Girene\n");
                              break;
  case 3:
          printf("Lefkosa\n");
          break;
  case 2:
          ++number;
  case -1:
          printf("Iskele\n");
          break;
  case 1:
          printf("Karpaz\n");
                  }
   ++number;
          }
          getchar();
          return(0);
  }
```

Example

```
#include <stdio.h>
int main(void)
{
   int number = 0;
 while (number < 5)
        {
              switch (number)
                 {
                      default:
                                printf("Gagusa\n");
                      case 0:
                                printf("Girene\n");
                                break;
 case 3:
          printf("Lefkosa\n");
          break;
 case 2:
          ++number;
 case -1:
          printf("Iskele\n");
          break;
 case 1:
          printf("Karpaz\n");
                  }
  ++number;
          }
          getchar();
          return(0);
 }
```

Number=2

Example

Number=4

```
#include <stdio.h>
int main(void)
{
  int number = 0;
  while (number < 5)
        {
        switch (number)
           {
              default:
                    printf("Gagusa\n");
              case 0:
                    printf("Girene\n");
                    break;
  case 3:
        printf("Lefkosa\n");
        break;
  case 2:
        ++number;
  case -1:
        printf("Iskele\n");
        break;
  case 1:
        printf("Karpaz\n");
              }
   ++number;
        }
        getchar();
        return(0);
  }
```

Example

```
#include <stdio.h>
int main(void)
{
  int number = 0;
 while (number < 5)
     {
          switch (number)
             {
                 default:
                          printf("Gagusa\n");
                 case 0:
                          printf("Girene\n");
                          break;
case 3:
        printf("Lefkosa\n");
        break;
case 2:
        ++number;
case -1:
        printf("Iskele\n");
        break;
case 1:
        printf("Karpaz\n");
                }
 ++number;
        }
        getchar();
        return(0);
}
```

Number=5

Infinite Loop Using `while`

```
while (1)
    statement;
next_statement;
```

Infinite Loop Using `while`

```
while (1)
        statement;
next_statement;
```

- Stop the program in the operating system level, for example, Ctrl-C in DOS.
- There is a break statement for terminating the loop (discussed later).
- ***Never*** use!

2. Repetition – The **do-while** Statement

```
do
      statement;
while (expr);

next_statement;
```

- Similar to **while**.
- But **statement** executed at least once because **expr** is evaluated at bottom.

```c
#include <stdio.h>

int main(void)
{
    int cnt=0, max=0, n, x;

    printf ("How many numbers do you wish to enter? ");
    scanf ("%d", &n);
    printf ("\nEnter %d decimal numbers:\n", n);

    do {
        scanf ("%d", &x);
        if (max < x)
            max = x;
        cnt++;
    } while (cnt < n);

    printf ("\nMaximum value: %d\n", max);

    return (0);
}
```

Same output as the **`while`** loop example provided **`n >= 1`**.

3. Repetition – The **for** Statement

```
for (expr1; expr2; expr3)
      statement;
next_statement;
```

- **`expr1`** – initialization
- **`expr2`** – condition
- **`expr3`** – (increment)

Execution Steps:

1. **`expr1`** is evaluated.
2. **`expr2`** is evaluated.
   - if true (non-zero),
     (a) **`statement`** is executed.
     (b) **`expr3`** is executed.
     (c) goto step (2) again.
   - if false (zero), **`next_statement`** is executed.

`while` Equivalent to the **`for`** Statement

```
for (expr1; expr2; expr3)
      statement;
next_statement;
```

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

```c
#include <stdio.h>

int main(void)
{
    int count, i;

    printf("Count? ");
    scanf("%d", &count);
    printf("\n");

    for (i = 0; i < count; i++)
        printf("%d\n", count - i);

    printf("Go!\n");

    return (0);
}
```

```
Count? 5
```

```c
#include <stdio.h>

int main(void)
{
    int count, i;

    printf("Count? ");
    scanf("%d", &count);
    printf("\n");

    for (i = 0; i < count; i++)
        printf("%d\n", count - i);

    printf("Go!\n");

    return (0);
}
```

```
Count? 5

5
4
3
2
1
Go!
```

```c
#include <stdio.h>

int main(void)
{

    int i, n, factorial=1;

    printf ("n ? ");
    scanf ("%d", &n);

    for (i=1; i <= n; i++)
        factorial *= i;

    printf ("The factorial of %d is %d\n.", n,
            factorial);

   return (0);
}
```

```
n ? 4
```

```c
#include <stdio.h>

int main(void)
{

    int i, n, factorial=1;

    printf ("n ? ");
    scanf ("%d", &n);

    for (i=1; i <= n; i++)
        factorial *= i;

    printf ("The factorial of %d is %d\n.", n,
            factorial);

   return (0);
}
```

n factorial
= n!
= 1 * 2 * 3 * … *
(n-2) * (n-1) * n

```
n ? 4
The factorial of 4 is 24.
```

The Empty Statement and **`for`**

```
for (expr1; expr2; expr3)
      statement;
next_statement;
```

- **`expr1`** and/or **`expr2`** and/or **`expr3`** can be omitted.
- **`;`** is still needed at the proper position.
- **`;`** is called the *empty statement*.
- Useful when a statement is needed *syntactically* but no action is required *semantically*.

Examples

```
i=1;
sum=0;

for ( ; i<=10; i++)
   sum += i;
```

A more proper writing:

```
sum=0;
for (i=1; i<=10; i++)
   sum += i;
```

Infinite loop using **`for`** (*Never* use!)

```
for (;;)
   statement;
```

```c
#include <stdio.h>

int main()
{
    int i, j, row, column;

    printf("Row = ? ");
    scanf("%d", &row);
    printf("Column = ? ");
    scanf("%d", &column);

    for (i=1; i<=row; i++) {
        for (j=1; j<=column; j++)
            printf("*");
        printf("\n");
    }

    return (0);
}
```

Nested for Loop

A for loop can be nested inside another for loop.

```
Row = ? 3
Column = ? 5
```

```c
#include <stdio.h>

int main()
{
    int i, j, row, column;

    printf("Row = ? ");
    scanf("%d", &row);
    printf("Column = ? ");
    scanf("%d", &column);

    for (i=1; i<=row; i++) {
        for (j=1; j<=column; j++)
            printf("*");
        printf("\n");
    }

    return (0);
}
```

Nested for Loop

A for loop can be nested inside another for loop.

```
Row = ? 3
Column = ? 5
*****
*****
*****
```

```c
#include <stdio.h>

int main(void)
{
    int i, j, size;

    printf("Size? ");
    scanf("%d", &size);

    for (i=1; i<=size; i++) {
        for (j=1; j<=(size-i); j++)
            printf(" ");
        for (j=1; j<=i; j++)
            printf("*");
        printf("\n");
    }

    return (0);
}
```

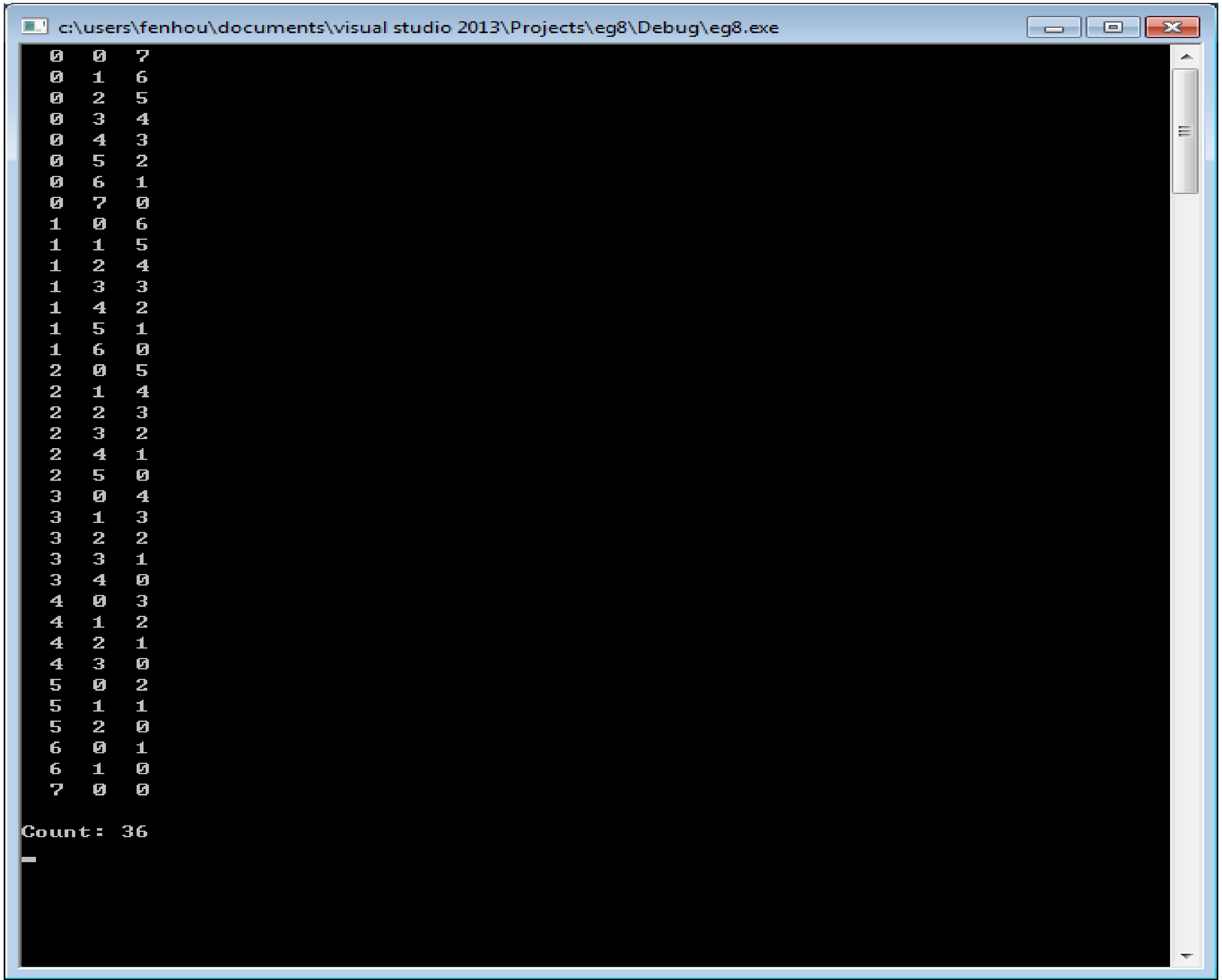
```
Size? 5
    *
   **
  ***
 ****
*****
```

Exercise

Study and execute the following program.

```c
#include <stdio.h>
#define N 7

int main(void)
{
    int cnt = 0, i, j, k;

    for (i = 0; i <= N; ++i)
        for (j = 0; j <= N; ++j)
            for (k = 0; k <= N; ++k)
                if (i + j + k == N) {
                    ++cnt;
                    printf("%3d%3d%3d\n", i, j, k);
                }

    printf("\nCount: %d\n", cnt);

    return (0);
}
```

Exercise



c:\users\fenhou\documents\visual studio 2013\Projects\eg8\Debug\eg8.exe

```
   0   0   7
   0   1   6
   0   2   5
   0   3   4
   0   4   3
   0   5   2
   0   6   1
   0   7   0
   1   0   6
   1   1   5
   1   2   4
   1   3   3
   1   4   2
   1   5   1
   1   6   0
   2   0   5
   2   1   4
   2   2   3
   2   3   2
   2   4   1
   2   5   0
   3   0   4
   3   1   3
   3   2   2
   3   3   1
   3   4   0
   4   0   3
   4   1   2
   4   2   1
   4   3   0
   5   0   2
   5   1   1
   5   2   0
   6   0   1
   6   1   0
   7   0   0

Count: 36
_
```

The Comma Operator and **for**

```
expr1, expr2
```

- Example,

```
for (i=1, factorial=1; i<=n; i++)
    factorial *= i;
```

Operator precedence and associativity

Operator	Associativity	Precedence
`()` `++` (postfix) `--` (postfix)	left to right	Highest
`+` (unary) `-` (unary) `++` (prefix) `--` (prefix) `!`	right to left	
`*` `/` `%`	left to right	
`+` `-`	left to right	
`<` `<=` `>` `>=`	left to right	
`==` `!=`	left to right	
`&&`	left to right	
`||`	left to right	
`?:`	right to left	
`=` `+=` `-=` `*=` `/=` etc.	right to left	
`,` (comma operator)	left to right	Lowest

The Comma Operator and **`for`**

```
expr1, expr2
```

- Lowest precedence of all operators.
- Left-to-right associativity.
- expr1 is evaluated first, then expr2. Value of **`expr2`** is taken as value of the whole expression.
- Example, **`a = 0 , b = 1`**;
- Example,

```
for (i=1, factorial=1; i<=n; i++)
        factorial *= i;
```

The Comma Operator and for

- Used in for statements, it allows multiple initializations and/or multiple processing of indices.

```
for (sum = 0, i = 1; i <= n; ++i)
    sum += i;
```

equivalent to

```
for (sum = 0, i = 1; i <= n; sum += i, ++i)
    ;
```

4. Controlling Repetition

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int x;

    while (1) {
        scanf("%d", &x);
        if (x <= 0)
            break;
        printf("square root = %.2f\n", sqrt(x));
    }

    printf("Bye!\n");
    return (0);
}
```

`break` and `while`

Causes an exit from the *innermost* enclosing loop.

`sqrt(x)` evaluates to "square root of x". The use of **`sqrt()`** requires **`#include <math.h>`**.

4. Controlling Repetition

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int x;

    while (1) {
        scanf("%d", &x);
        if (x <= 0)
            break;
        printf("square root = %.2f\n", sqrt(x));
    }

    printf("Bye!\n");
    return (0);
}
```

break and while

Causes an exit from the *innermost* enclosing loop.

sqrt(x) evaluates to "square root of x". The use of **sqrt()** requires **#include <math.h>**.

```
10
square root = 3.16
16
square root = 4.00
0
Bye!
```

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int x;

    do {
        scanf("%d", &x);
        if (x <= 0)
            break;
        printf("square root = %.2f\n", sqrt(x));
    } while (1);

    printf("Bye!\n");
    return (0);
}
```

break and **do-while**

Causes an exit from the *innermost* enclosing loop.

```
10
square root = 3.16
16
square root = 4.00
0
Bye!
```

break and **for**

Causes an exit from the *innermost* enclosing loop. Will **expr3** be executed before leaving the for loop?

```c
#include <stdio.h>

int main(void)
{
    int i, x;

    for (i=0; i<10; i++) {
        printf("i = %d\t", i);
        printf("x = ? ");
        scanf("%d", &x);
        if (x==0)
            break;
    }


    printf("After the loop, i = %d\n", i);
    printf("Bye!\n");

    return (0);
}
```

```
i = 0 x = ? 10
i = 1 x = ? 20
i = 2 x = ? 4
i = 3 x = ? 5
i = 4 x = ? 0
```

```c
#include <stdio.h>

int main(void)
{
    int i, x;

    for (i=0; i<10; i++) {
        printf("i = %d\t", i);
        printf("x = ? ");
        scanf("%d", &x);
        if (x==0)
            break;
    }


    printf("After the loop, i = %d\n", i);
    printf("Bye!\n");

    return (0);
}
```

break and for

Causes an exit from the *innermost* enclosing loop. Will **expr3** be executed before leaving the for loop?

```
i = 0 x = ? 10
i = 1 x = ? 20
i = 2 x = ? 4
i = 3 x = ? 5
i = 4 x = ? 0
After the loop, i = 4
Bye!
```

Expre3 will NOT be executed;
Will directly leave the for loop
from the break statement;

```c
#include <stdio.h>
#define MAX 5

int main(void)
{
    int data, sum=0, k;

    for (k=0; k<MAX; k++) {
        scanf ("%d", &data);
        if (data <= 0)
            continue;
        sum += data;
    }

    printf ("Sum of positive values is %d\n.", sum);
    return (0);
}
```

continue

Causes the current iteration of a loop to stop, and begins the next iteration.

```
10
20
-1
90
-5
```

continue as applied to **while** and **do-while**?
Leave to you as exercises!

```c
#include <stdio.h>
#define MAX 5

int main(void)
{
    int data, sum=0, k;

    for (k=0; k<MAX; k++) {
        scanf ("%d", &data);
        if (data <= 0)
            continue;
        sum += data;
    }

    printf ("Sum of positive values is %d\n.", sum);
    return (0);
}
```

continue

Causes the current iteration of a loop to stop, and begins the next iteration.

```
10
20
-1
90
-5
Sum of positive values is 120.
```

continue as applied to **while** and **do-while**?
Leave to you as exercises!

```
#include <stdio.h>
#define MAX 5

int main(void)
{
    int data, sum=0, k;

    for (k=0; k<MAX; k++) {
        scanf ("%d", &data);
        if (data <= 0)
            ~~continue;~~ break;
        sum += data;
    }

    printf ("Sum of positive values is %d\n.", sum);
    return (0);
}
```

continue

Causes the current iteration of a loop to stop, and begins the next iteration.

```
10
20
-1


                                         30
Sum of positive values is ~~120~~.
```

continue as applied to **while** and **do-while**?
Leave to you as exercises!

Example

```
#include <stdio.h>
int main(void)
{
   int  j;
  for (j=0; j <= 4;  j++)
         {
              if (j == 2)
                 {
                   continue;
                  }
          printf("%d\n", j);
         }
    getchar();
    return(0);
  }
```

Example

```
#include <stdio.h>
int main(void)
{
    int  j;
   for (j=0; j <= 4;  j++)
        {
            if (j == 2)
                {
                    continue;
                }
        printf("%d\n", j);
        }
    getchar();
    return(0);
 }
```

Example

Output:

```
0
1
3
4
```

Example

```
#include <stdio.h>
int main(void)
{
  int  j=10 ;
 while (j >=0 )
      {
          if (j == 5)
             {
              j--;
             continue;
              }
      printf(“%d  ”, j);
      j--;
      }
   getchar();
   return(0);
    }
```

Example

```
#include <stdio.h>
int main(void)
{
   int  j=10 ;
 while (j >=0 )
        {
           if (j == 5)
               {
                j--;
               continue;
                }
          printf(“%d  ”, j);
          j--;
          }
  getchar();
  return(0);
   }
```

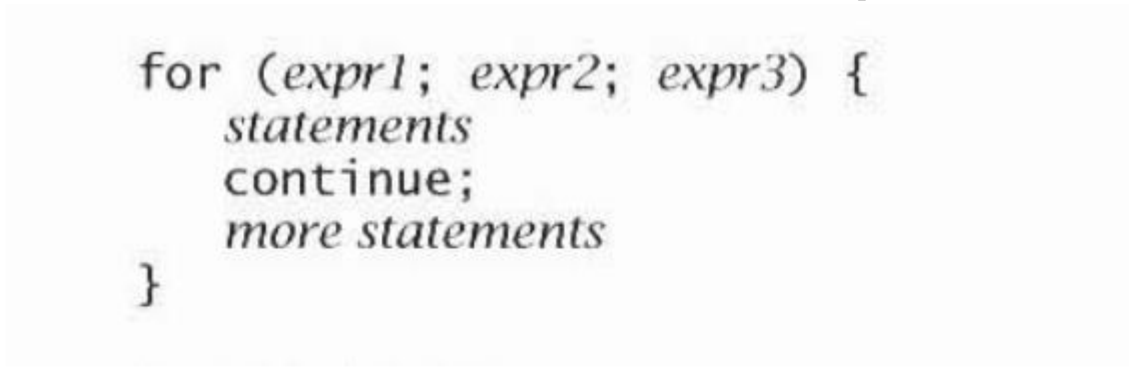
Example

Output:

```
10  9  8  7  6  4 3 2 1 0
```

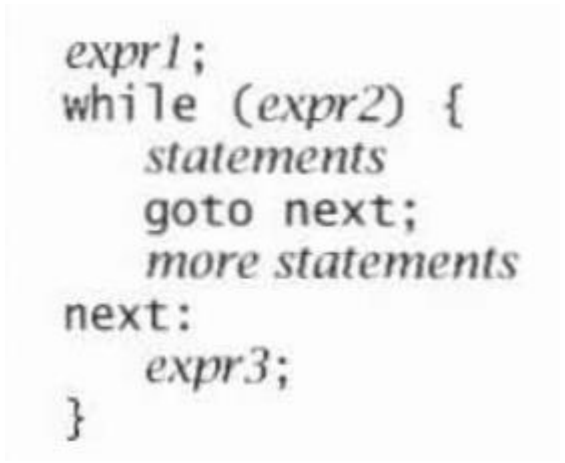
The continue statement in for loop

- The continue statement may only occure inside for, while, or do loops.



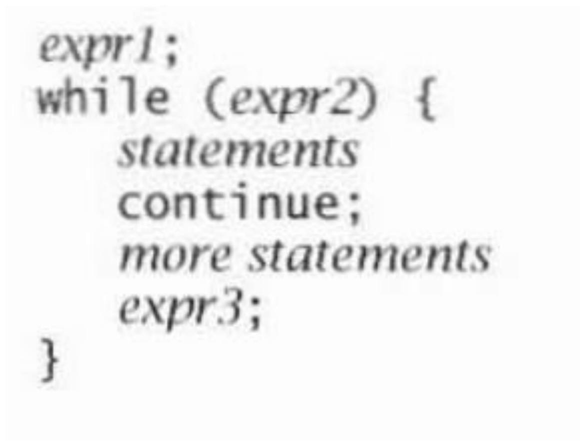
```
for (expr1; expr2; expr3) {
    statements
    continue;
    more statements
}
```

is equivalent to



```
expr1;
while (expr2) {
    statements
    goto next;
    more statements
next:
    expr3;
}
```

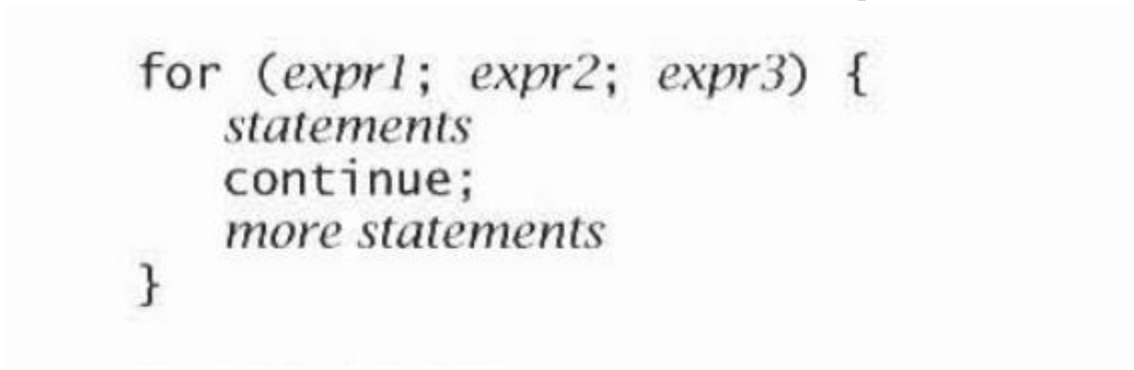
or



```
expr1;
while (expr2) {
    statements
    continue;
    more statements
    expr3;
}
```

The continue statement in for loop

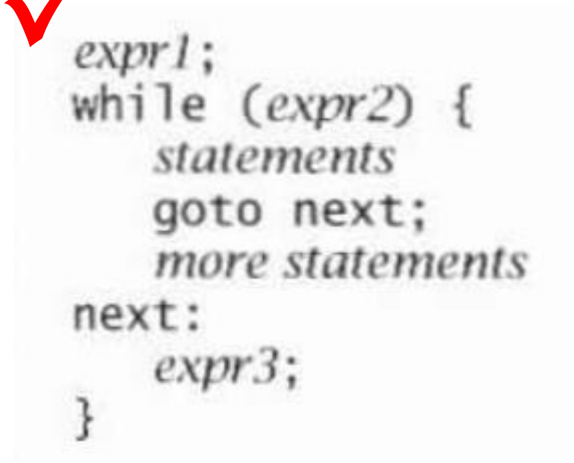
- The continue statement may only occure inside for, while, or do loops.



```
for (expr1; expr2; expr3) {
    statements
    continue;
    more statements
}
```

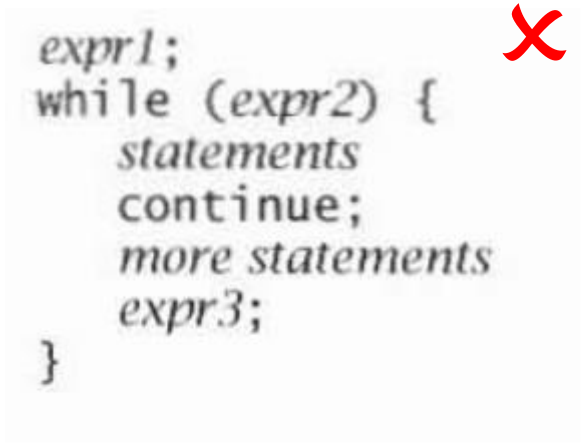
is equivalent to

✓



```
expr1;
while (expr2) {
    statements
    goto next;
    more statements
next:
    expr3;
}
```

or



```
expr1;
while (expr2) {
    statements
    continue;
    more statements
    expr3;
}
```

✗

4. Controlling Repetition - The goto Statement

goto label;

- When a program execution encounters a goto statement, execution immediately jumps to the labeled statement specified by the goto statement.
- A labeled statement is of the form

  label: statement

  where label is an identifier.
- Example:

  Loc1: a=a+b;

  - but not 333: a=a+b ; /* 333 is not an identifier*/

4. Controlling Repetition - The goto Statement

- Cause an unconditional jump to a labeled statement somewhere in the current function.
- A goto statement and its target label must be located in the same function, although they can be in different blocks.
- Strongly not recommended.