Jumping statements

Introduction

- There are 2 special statements that can *affect* the execution of loop statements (such as a *while-statement*)
- The special statements are:

  - break
  - continue

  We will study their meaning and how to use these special statements inside the while-statement

How Break Statement works?

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

The break statement

- Syntax:

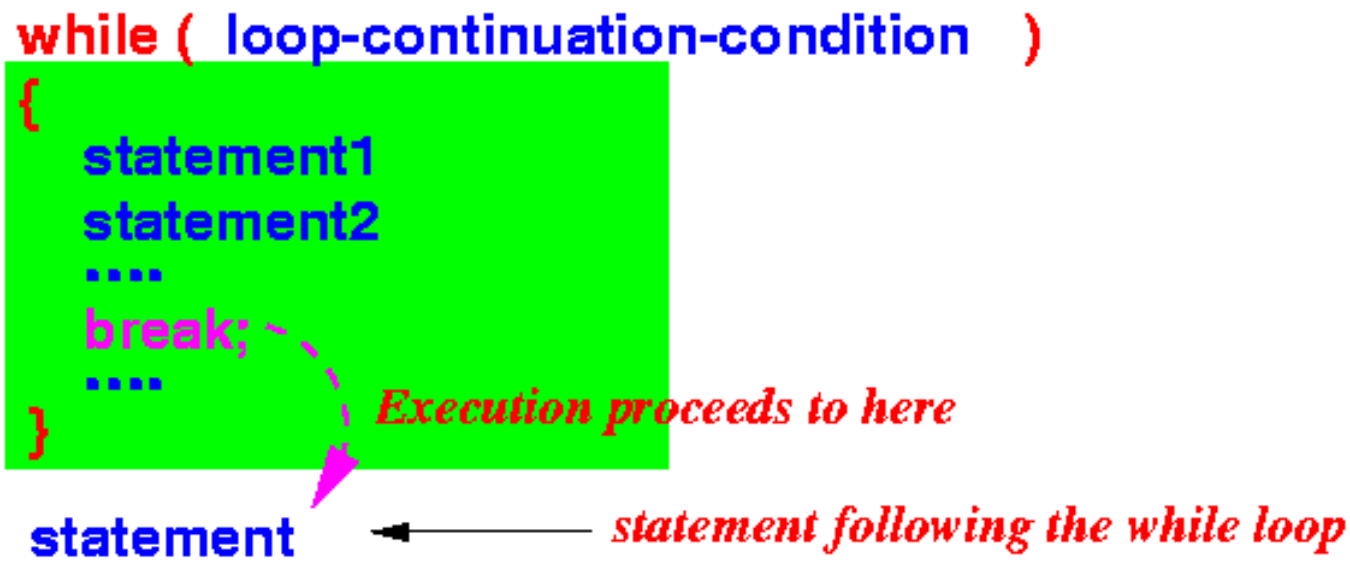
```
break;
```

Effect:

- When the break statement is *executed* inside a loop-statement, the loop-statement is terminated *immediately*
- The execution of the program will continue with the statement *following* the loop-statement
- The break statement is almost always used with if...else statement inside the loop.

The break statement (cont.)

- Schematically:



Write a program to calculate the sum of numbers (max 10 numbers) . If the user enters a negative number loop terminates.

Example 1:

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates

#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, break the loop
      if (number < 0.0) {
         break;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Example 1:

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates

#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, break the loop
      if (number < 0.0) {
         break;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Output:

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

Example 2:

```
void main()
{
    int x=1;
    while(x<=10)
    {
        printf("%d\n", x);
        if(x = = 5)
        {
            break;
        }
        x++;
    }
}
```

Example 2:

```
void main()
{
    int x=1;
    while(x<=10)
    {
        printf("%d\n", x);
        if(x = = 5)
        {
            break;
        }
        x++;
    }
}
```

output

```
1
2
3
4
5
```

Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

```
Step 1: Enter the num
Step 2: set i=2;
Step 3: repeat steps 4 to 5 while i < =num-1
Step 4: if(num%i==0)
            break and exit the loop
            else
             i=i+1;
Step 5: go to step 3
Step 6: if(i==num)
          Print num is a prime number
Step 7: End
```

```
main( )
{
    int   num, i ;

    printf ( "Enter a number " ) ;
    scanf ( "%d", &num ) ;

    i = 2 ;
    while ( i <= num - 1 )
    {
        if ( num % i == 0 )
        {
            printf ( "Not a prime number" ) ;
            break ;
        }
        i++ ;
    }
    if ( i == num )
        printf ( "Prime number" ) ;
}
```

Example 3: Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

The keywor n
which it is p

```c
#include<stdio.h>
int main()
{
int i = 1 , j = 1 ;
while ( i++ <= 10 )
 {
    while ( j++ <= 20 )
    {
        if ( j == 15 )
        break ;
        else
        printf ( "%d %d\t", i, j ) ;
    }
    printf("\n") ;
 }
return 0;
}
```

What would be its output?

Programming example using the break statement: find the GCD

- Problem description:

> • Write a C program that reads in 2 numbers x and y...
>
> • and prints the *largest* common divisor of both x and y

Programming example using the break statement: find the GCD (cont.)

- A *concrete* example:

- Input: x = 24 and y = 16
- Output: 8

Programming example using the break statement: find the GCD (cont.)

- What would *you* do to solve this problem ?

- Suppose: x = 24 and y = 16
  - The lesser of the values is 16
  - Therefore, all divisors are ≤ 16

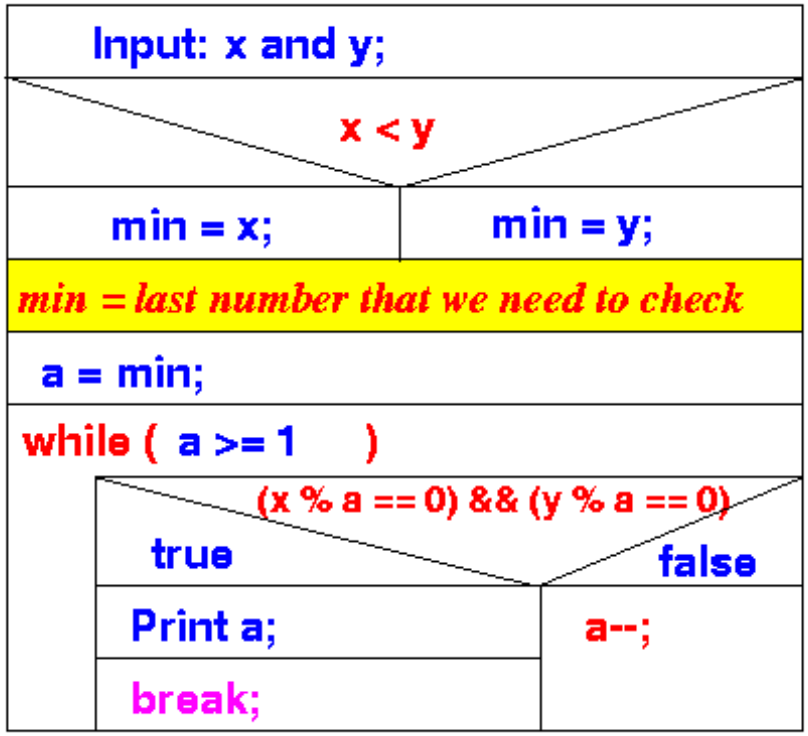
Programming example using the break statement: find the GCD (cont.)

- Check if 16 *and* 24 are *divisible by* 16: no
- Check if 16 *and* 24 are *divisible by* 15: no
- …
- Check if 16 *and* 24 are *divisible by* 10: no
- Check if 16 *and* 24 are *divisible by* 9: no
- Check if 16 *and* 24 are *divisible by* 8: YES

- Print **8** and **STOP**

Programming example using the break statement: find the GCD (cont.)

- Algorithm (structured diagram):



The continue statement

- Syntax:

```
continue;
```

The continue statement (cont.)

- **Effect**: When the continue statement is *executed* inside a loop-statement, the program will skip over the *remainder* of the loop-body to the end of the loop

- What happens next when the program *reaches* the end of a loop *depends on* the *type* of loop statement !!!

- The continue statement is almost always used with the if……else statement

- Continue doesn't terminate the loop but takes the loop to next iteration

The continue statement (cont.)

- Effect of a continue statement in a while-loop:

- As given previously:

  - the program will skip over the *remainder* of the loop-body to the end of the loop

- In the case of a *while*-loop, when the program reaches end of the loop, the program will jump back to the testing of the *loop-continuation-condition*

How continue statement works?

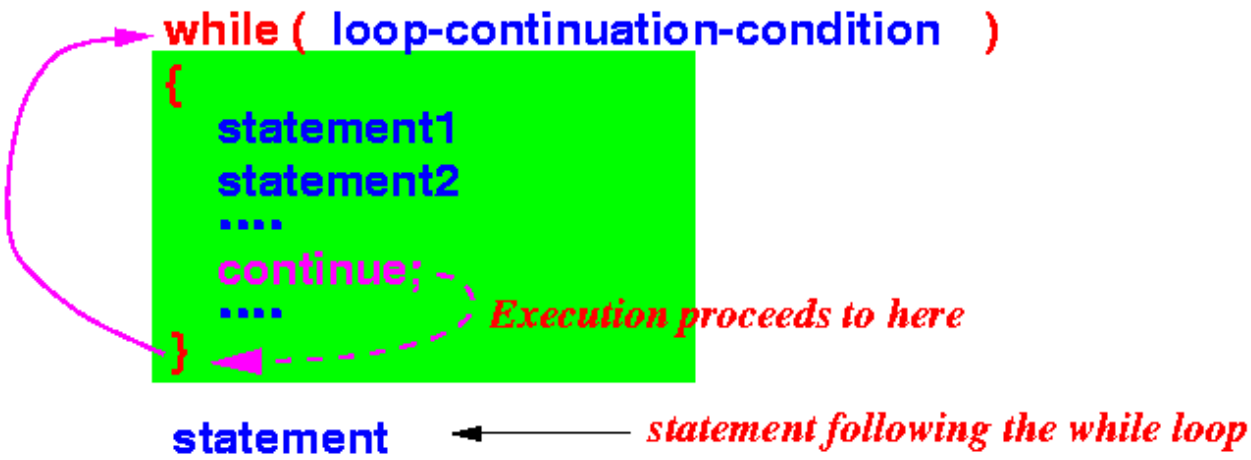
```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

The continue statement (cont.)

- Schematically:



continue

```
void main()
{
    int x=1;
    while(x<=10)
    {
        if(x % 3 = = 0)
        {
            x++;
            continue;
        }
        printf ("%d \n" , x);
        x++;
    }
}
```

Example: Continue

```
void main()
{
    int x=1;
    while(x<=10)
    {
        if(x % 3 = = 0)
        {
            x++;
            continue;
        }
        printf ("%d \n" , x);
        x++;
    }
}
```

Output

1
2
4
5
7
8
10

```
main( )
{
    int  i, j ;

    for ( i = 1 ; i <= 2 ; i++ )
    {
        for ( j = 1 ; j <= 2 ; j++ )
        {
            if ( i == j )
                continue ;

            printf ( "\n%d %d\n", i, j ) ;
        }
    }
}
```

Example

```
main( )
{
    int  i, j ;

    for ( i = 1 ; i <= 2 ; i++ )
    {
        for ( j = 1 ; j <= 2 ; j++ )
        {
            if ( i == j )
                continue ;

            printf ( "\n%d %d\n", i, j ) ;
        }
    }
}
```

The output of the above program would be...

```
1 2
2 1
```

Example 2: continue statement

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the result

#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 (number < 0.0) {
         continue;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Example 2: continue statement

```c
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the result

#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 (number < 0.0) {
         continue;
      }

      sum += number; // sum = sum + number;
   }

   printf("Sum = %.2lf", sum);

   return 0;
}
```

Output:

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

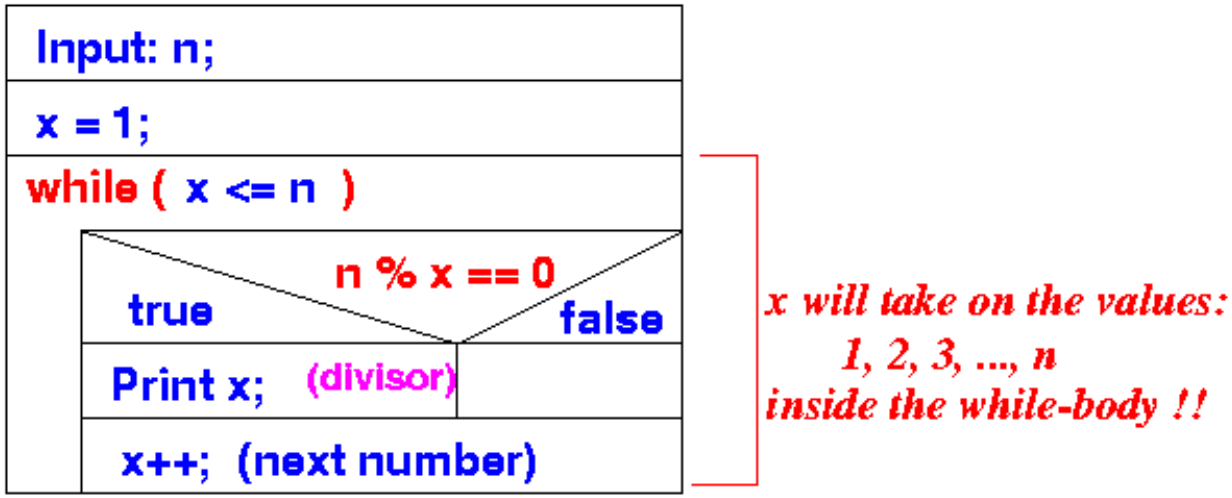
Programming example using the continue statement: find all divisors of a number

- Problem description:

- Write a C program that reads in an integer n...
- and prints all its divisors

Programming example using the continue statement: find all divisors of a number (cont.)

- Previously discussed solution:

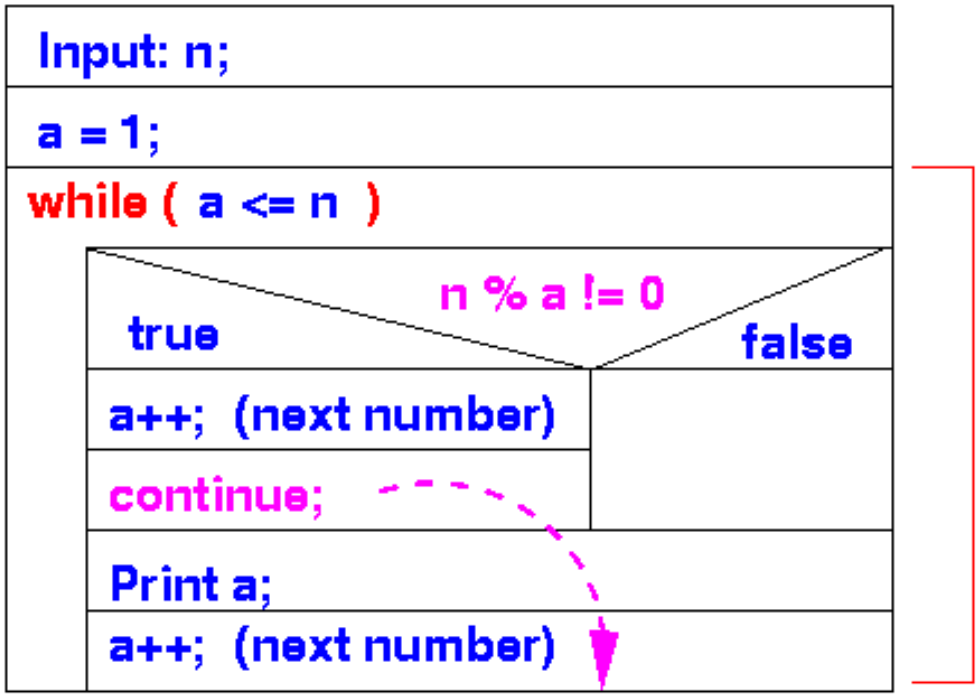


We try every number a = 1, 2, ..., n

For each number a, we check if n % a == 0.

Programming example using the continue statement: find all divisors of a number (cont.)

- We can re-write the *same* algorithm differently using a continue statement as follows:



n=6
a=1;
6%1!=0 False **Print 1** a=2
6%2!=0 False **Print 2** a=3
6%3!=0 False **Print 3** a=4
6%4!=0 True a=5 continue
6%5!=0 True a=6 continue
6%6!=0 True **Print 6**

a will take on the values:
1, 2, 3, ..., n
inside the while-body !!

Programming example using the continue statement: find all divisors of a number (cont.)

Notice that the *if*-condition has been changed to x % a != 0, meaning: a is not a divisor of x

When a is not a divisor of x, (the then-part), we increment a (to try next number) and jump to the *end of the while-loop* using the continue statement.

When x % a != 0 is false, the program will print a and increment a (to try next number)

Difference Between break and continue

break	continue
A `break` can appear in both `switch` and loop (`for`, `while`, `do`) statements.	A `continue` can appear only in loop (`for`, `while`, `do`) statements.
A `break` causes the `switch` or loop statements to terminate the moment it is executed. Loop or `switch` ends abruptly when break is encountered.	A `continue` doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if `continue` is encountered. The `continue` statement is used to skip statements in the loop that appear after the `continue`.

Difference Between break and continue

break	continue
The `break` statement can be used in both `switch` and loop statements.	The `continue` statement can appear only in loops. You will get an error if this appears in switch statement.
When a `break` statement is encountered, it terminates the block and gets the control out of the `switch` or loop.	When a `continue` statement is encountered, it gets the control to the next iteration of the loop.
A `break` causes the innermost enclosing loop or `switch` to be exited immediately.	A `continue` inside a loop nested within a `switch` causes the next loop iteration.

goto statement

- goto statement is used for unconditional jumping.
- We can move the control from any part of the program to any other part of the program with the help of goto statement.

Example: goto statement

```
void main()
{
    printf("Hello \n");
    goto abc;
    printf("Welcome \n");
    abc:
    printf("Good Morning");
}
```

Example: goto statement

```
void main()
{
    printf("Hello \n");
    goto abc;
    printf("Welcome \n");
    abc:
    printf("Good Morning");
}
```

Output:

```
Hello
Good Morning
```

Return statement

- Return statement is used to return a value from a function.
- The value is returned to the place where the function is called.

Problem Statement

Write a program to generate all prime numbers between 1 to 1000

Write a program to generate all prime numbers between 1 to 1000

Algorithm [Though Incomplete]

Strep 1 j=1;

Step 2: Repeat steps 3 to 6 while j <=1000 (i.e for(j=1;j<=1000 ;j++))

Step 3: num=j

Step 4: Check weather num is prime or not

Step 5: Print num if prime

Step 6: increment j and go to step 2

Step 7: End

Algorithm

Strep 1 j=1;

Step 2: Repeat steps 3 to 6 while j <=1000 (i.e for(j=1;j<=1000 ;j++))

Step 3: num=j

Step 4: set i=2;

Step 5: repeat steps 6 to 7 while i < =num-1

Step 6: if(num%i==0)

break and exit the loop

else

i=i+1;

Step 7: go to step 5

Step 8: if(i==num)

Print num is a prime number

Step 9: increment j and go to step 2

Step 10: End

Inner loop checking prime number