Readin from and Writint to Standart I/O

BIL104E: Introduction to Scientific and Engineering Computing

Lecture 4

- Doing the same thing over and over
- More operators

Introduction

One of the very important feature of the C language: looping.

Three statements in C are designed for looping:
(repetition control statements)

- The `for` statement
- The `while` statement
- The `do-while` statement

The following sections explore these statements.

The Essentials of Repetition

continue statement

- Used for skipping the remainder of the body of a repetition structure and proceeding with the next iteration of the loop.

Loop

– Group of instructions computer executes repeatedly while some condition remains true

- **Counter-controlled repetition**

– Definite repetition: know how many times loop will execute

– Control variable used to count repetitions

- **Sentinel-controlled repetition**

– Indefinite repetition

– Used when number of repetitions not known

– Sentinel value indicates "end of data"

Essentials of Counter-Controlled Repetition

Counter-controlled repetition requires

- The name of a control variable (or loop counter)
- The initial value of the control variable
- A condition that tests for the final value of the control variable (i.e., whether looping should continue)
- An increment (or decrement) by which the control variable is modified each time through the loop

The for Repetition Structure

The general form of the for statement is

```
for (expression1; expression2; expression3)
{
statement1;
statement2;
. . .
}
```

The for Repetition Structure

Example:

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

Another example:

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
printf( "%d\n", j + i );
```

More example:

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

The for Repetition Structure

```c
/* Converting 0 through 15 to hex numbers */
#include <stdio.h>

main()
{
   int i;

   printf("Hex(uppercase) Hex(lowercase) Decimal\n");
   for (i=0; i<16; i++){
       printf("%X %x %d\n", i, i, i);
   }
   return 0;
}
```

```
Hex(uppercase)    Hex(lowercase)    Decimal
0                 0                 0
1                 1                 1
2                 2                 2
3                 3                 3
4                 4                 4
5                 5                 5
6                 6                 6
7                 1                 7
8                 2                 8
9                 3                 9
A                 a                 10
B                 b                 11
C                 c                 12
.                 .                 .
```

The `for` Repetition Structure – more info

- **Arithmetic expressions in `for` statement :**
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If **`x`** equals **`2`** and **`y`** equals **`10`**

```
    for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
    for ( j = 2; j <= 80; j += 5 )
```

- **Notes about the for structure:**
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially **`false`**
    - The body of the **`for`** structure is not performed
    - Control proceeds with the next statement after the **`for`** structure
  - Control variable
    - Often printed or used inside for body, but not necessary

The `while` Repetition Structure

while loop (Repeat statements as long as condition is true)

```
initialization;
while ( loopContinuationTest ) {
statement(s);
increment;
}
```

- Example: (printing numbers from 0 to 10

```
i = 0;
while(i<=10){
printf( "%d\n",i );
i++;
}
```

The while Repetition Structure

while loop (Repeat statements as long as condition is true)

```
initialization;
while ( loopContinuationTest ) {
statement(s);
increment;
}
```

– Example: (printing numbers from 0 to 10

```
i = 0;
while(i<=10){
printf( "%d\n",i );
i++;
}
```

The while Repetition Structure

The while statement is also used for looping. Unlike the situation with the for statement, there is only one expression field in the while statement.

The general form of the while statement is:

```
while (expression) {
   statement1;
   statement2;
   . . .
}
```

The while Repetition Structure

```c
/* Using a while loop */
#include <stdio.h>

main()
{
   int c;

   c = ` `;
   printf("Enter a character:\n(enter x to exit)\n");
   while (c != `x') {
        c = getc(stdin);
        putchar(c);
    }
    printf("\nOut of the while loop. Bye!\n");
    return 0;
 }
```

```
Enter a character:

(enter x to exit)
H
H
i
i
x
x
Out of the while loop. Bye!
```

The `do / while` Repetition Structure

Another statement used for looping, do-while, which puts the expressions at the bottom of the loop :

```
do {
    statement1;
    statement2;
    . . .
} while (expression);
```

- Similar to the **`while`** structure
- Condition for repetition tested after the body of the loop is performed

The do / while Repetition Structure

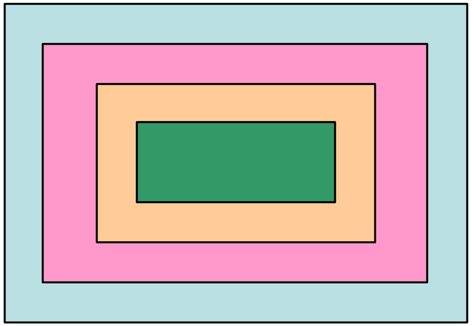
```c
/* Using a do-while loop */
#include <stdio.h>

main()
{
int i;

i = 65;
do {
printf("The numeric value of %c is %d.\n", i, i);
i++;
} while (i<72);
return 0;
}
```

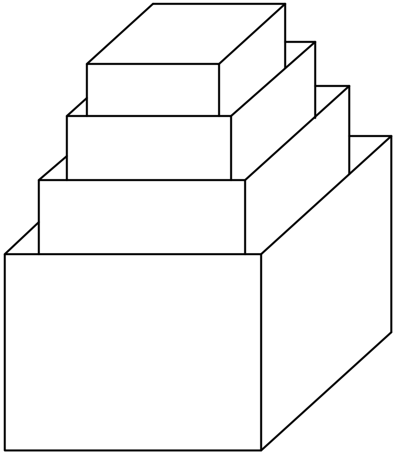
```
The numeric value of A is 65.
The numeric value of B is 66.
The numeric value of C is 67.
The numeric value of D is 68.
The numeric value of E is 69.
The numeric value of F is 70.
The numeric value of G is 71.
```

The Repetition Structure - Nested Loops

- You can put a loop inside another one to make nested loops.

- The computer will run the inner loop first before it resumes the looping for the outer loop.





The **break** and **continue** Statements

break

– Causes immediate exit from a **while**, **for**, **do/while** or **switch** structure

– Program execution continues with the first statement after the structure

– Common uses of the **break** statement

- Escape early from a loop
- Skip the remainder of a **switch** structure

The **break** and **continue** Statements

Continue

– Skips the remaining statements in the body of a while, for or **do/while** structure

- Proceeds with the next iteration of the loop

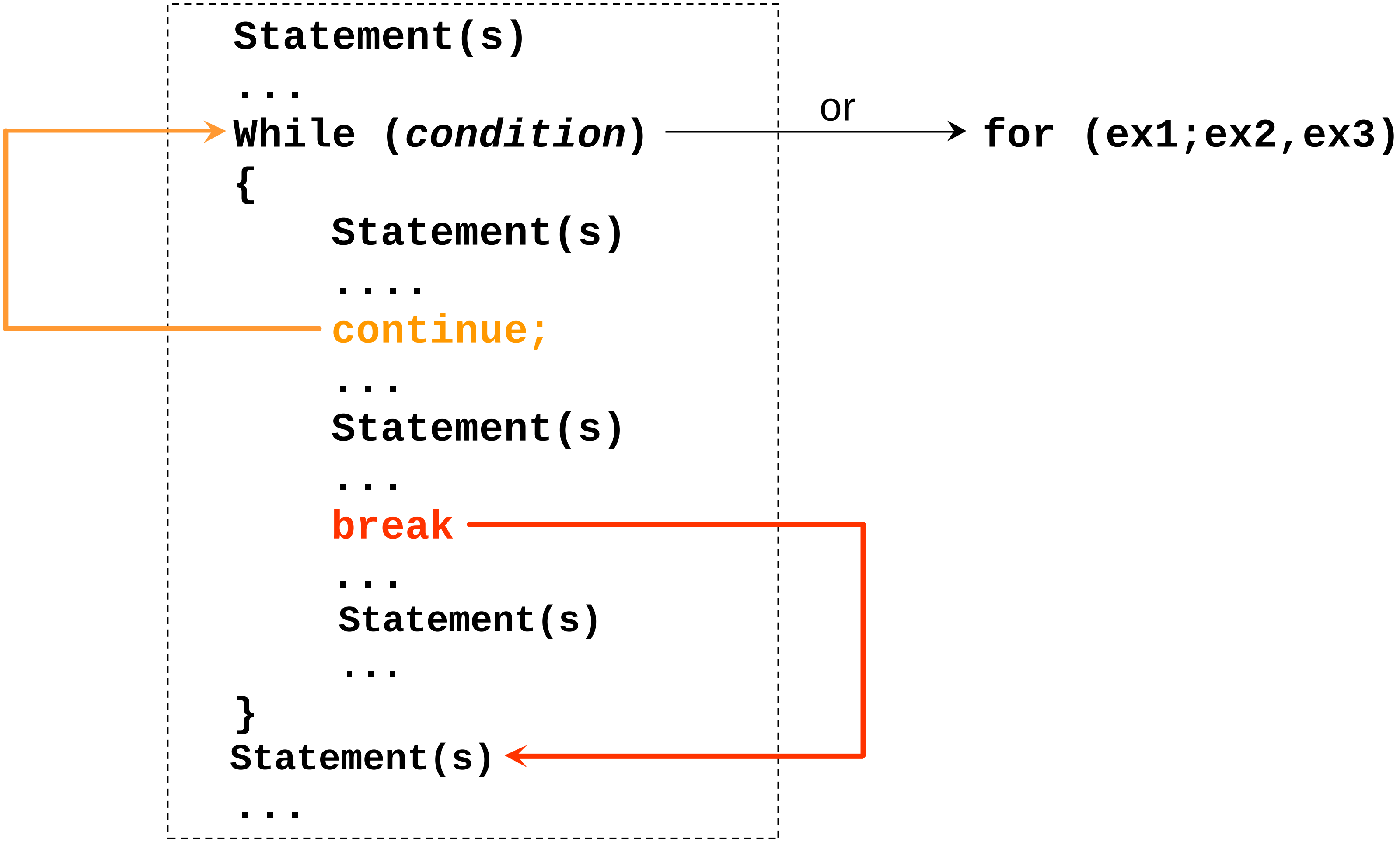
– **while** and **do/while**

- Loop-continuation test is evaluated immediately after the continue statement is executed

– **for**

- Increment expression is executed, then the loop-continuation test is evaluated.

The **break** and **continue** Statements



```
Statement(s)
...
While (condition)
{
    Statement(s)
    ....
    continue;
    ...
    Statement(s)
    ...
    break
    ...
    Statement(s)
    ...
}
Statement(s)
...
```

The **break** and **continue** Statements

```
   /*  Using the continue statement in a for structure */

#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         continue;               /* skip remaining code in loop only if x == 5 */

      printf( "%d ", x );
   }

   printf( "\nUsed continue to skip printing the value : 5\n" );
   return 0;
}
```

```
Program output:
Used continue to skip printing the value : 5
```

The break and continue Statements

```c
   /*  Using the continue statement in a for structure */

#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         continue;               /* skip remaining code in loop only if x == 5 */

      printf( "%d ", x );
   }

   printf( "\nUsed continue to skip printing the value : 5\n" );
   return 0;
}
```

```
Program output:
Used continue to skip printing the value : 5
```

Measuring Data Sizes

you can measure the data type size by using the **sizeof** operator :

```
/* Using the sizeof operator */
#include <stdio.h>

main()
{
char ch = ` `;
int int_num = 0;
float flt_num = 0.0f;
double dbl_num = 0.0;

printf("The size of char is: %d-byte\n", sizeof(char));
 printf("The size of ch is: %d-byte\n", sizeof ch );
  printf("The size of int is: %d-byte\n", sizeof(int));
   printf("The size of int_num is: %d-byte\n", sizeof int_num);
   printf("The size of float is: %d-byte\n", sizeof(float));
  printf("The size of flt_num is: %d-byte\n", sizeof flt_num);
 printf("The size of double is: %d-byte\n", sizeof(double));
printf("The size of dbl_num is: %d-byte\n", sizeof dbl_num);
return 0;
}
```

```
Program Output:

The size of char is: 1-byte
The size of ch is: 1-byte
The size of int is: 2-byte
The size of int_num is: 2-byte
The size of float is: 4-byte
The size of flt_num is: 4-byte
The size of double is: 8-byte
The size of dbl_num is: 8-byte
```

Operators

The assignment, Mathematical, Relational, Logical operators

Logical Operators ?

Operator	**Symbol**	**Example**
AND	&&	*exp1* && *exp2*
OR	||	*exp1* || *exp2*
NOT	!	!*exp1*

True → 1

False → 0

Operators – AND / OR / NOT

- **&&** (logical AND)
  - Returns **true** if both conditions are **true**
- **||** (logical OR)
  - Returns **true** if either of its conditions are **true**
- **!** (logical NOT, logical negation)
  - Reverses the truth/falsity of its condition
  - Unary operator, has one operand
- Useful as conditions in loops

Expression	Result
`true && false`	**`false`**
`true || false`	**`true`**
`!false`	**`true`**

Operators – AND / OR / NOT

More example

<table>
<tr><th>Expression</th><th>What It Evaluates To</th></tr>
<tr><td>(<i>exp1</i> &amp;&amp; <i>exp2</i>)</td><td>True (1) only if both <i>exp1</i> and <i>exp2</i> are true; false (0) otherwise</td></tr>
<tr><td>(<i>exp1</i> || <i>exp2</i>)</td><td>True (1) if either <i>exp1</i> or <i>exp2</i> is true; false (0) only if both are false</td></tr>
<tr><td>(!<i>exp1</i>)</td><td>False (0) if <i>exp1</i> is true; true (1) if <i>exp1</i> is false</td></tr>
</table>

Operators

The assignment, Mathematical, Relational, Logical operators

Relational Operators ?

==, <, >, >=, <=, !=

True → 1

False → 0

Equality (==) and Assignment (=) Operators

== Eoperator is an equality operator but **=** is an assignment operator.

```
y = 5 + 3 ;
```

~~y == 5 + 3;~~

Example using == :

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

- Checks paycode, if it is 4 then a bonus is awarded

Equality (==) and Assignment (=) Operators

Example, replacing == with =:

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

- This sets **`paycode`** to **`4`**
- **`4`** is nonzero, so expression is **`true`**, and bonus awarded no matter what the **`paycode`** was
  - Logic error, not a syntax error

The other Operators

C's relational operators are used to compare expressions, asking questions such as, ***"Is x greater than 100?"*** or ***"Is y equal to 0?"*** An expression containing a relational operator evaluates to either **true** (**1**) or **false** (**0**). C's six relational operators are listed in Table:

Operator	Symbol	Question Asked	Example
Equal	==	Is operand 1 equal to operand 2?	x == y
Greater than	>	Is operand 1 greater than operand 2?	x > y
Less than	<	Is operand 1 less than operand 2?	x < y
Greater than or equal to	>=	Is operand 1 greater than or equal to operand 2?	x >= y
Less than or equal to	<=	Is operand 1 less than or equal to operand 2?	x <= y
Not equal	!=	Is operand 1 not equal to operand 2?	x != y

The other Operators

Examples:

Expression	**How It Reads**	**What It Evaluates To**
5 == 1	Is 5 equal to 1?	0 (false)
5 > 1	Is 5 greater than 1?	1 (true)
5 != 1	Is 5 not equal to 1?	1 (true)
(5 + 10) == (3 * 5)	Is (5 + 10) equal to (3 * 5)?	1 (true)