Flow control

UPC - Videogame Design & Development - Programming I

Contents

Statements and blocks

A **statement** is, for instance, a line of code ending in the character **;**

For example:

```
// These are examples of statements
float time = 34.4f;
time += 7;
int frameCount = 30;
++frameCount;
```

A **block** is a set of statements grouped together within braces **{ }**

For example:

```
// This is a block of statements
{
   float time = 34.4f;
   time += 7;
   int frameCount = 30;
   ++frameCount;
}
```

If / else (conditional statement)

The **if** statement is used to express decisions. Syntax (not valid C):

```
if (condition)
        Statement / block
```

```
if (condition)
        Statement / block
else
        Statement / block
```

```
if (condition1)
        Statement / block
else if (condition2)
        Statement / block
else if (condition3)
        Statement / block
. . .
else
        Statement / block
```

- We can add as many **else if (condition)** blocks as we want.
- An optional **else** block (only one, and without a condition between parentheses) can be added at the end of the construction.

If / else (conditional statement)

Be careful not using blocks in these kind of constructions.

```
if (distance < EXPLOSION_RADIUS)
   health -= EXPLOSION_DAMAGE;
```

✓

```
if (distance < EXPLOSION_RADIUS) {
   health -= EXPLOSION_DAMAGE;
}
```

✓

```
if (distance < EXPLOSION_RADIUS)
   health -= EXPLOSION_DAMAGE;
   ragePoints += 5;
```

✗

Here, the statement **ragePoints += 5;** is out of the conditional statement, so it will be always executed

```
if (distance < EXPLOSION_RADIUS) {
   health -= EXPLOSION_DAMAGE;
   ragePoints += 5;
}
```

✓

If / else (conditional statement)

Example 1:

```
int number;
printf("Introduce a number: ");
scanf_s("%d", &number);

if (number == 0)
{
   printf("You have introduced the number zero.\n");
}
else if (number < 0)
{
   printf("You have introduced a negative
number.\n");
}
else
{
   printf("You have introduced a positive
number.\n");
}
```

Example 2:

```
int number;
printf("Introduce a number: ");
scanf_s("%d", &number);

if (number == 0) {
   printf("You have introduced the number zero.\n");
} else if (number == 1) {
   printf("You have introduced the number one.\n");
} else if (number == 10) {
   printf("You have introduced the number ten.\n");
} else if (number < 0) {
   printf("You have introduced a negative
number.\n");
} else {
   printf("You have introduced a positive
number.\n");
}
```

Switch (conditional statement)

- A multi-way decision that tests whether an expression matches on of a number or constant integer (char is an integer type too).
- It evaluates the expression in **switch**, and branches accordingly executing the code in the **case** with the matching value.
- The case labeled **default** is executed if none of other cases are satisfied.
- The **break** statement causes an immediate exit from the switch. If not used, code in the next **case** within the switch continues executing.

```
switch (expression)
{
        case value1:
                Statements
                break;
        case value2:
                Statements
                break;
        case value3:
                Statements
                break;
        . . .
        default:
                Statements
}
```

Switch (conditional statement)

Example 1:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
   int num;
   printf( "Type a number " );
   scanf_s( "%d", &num );
   switch( num ){
       case 1:
           printf( "It’s 1\n" );
           break;
       case 2:
           printf( "It’s 2\n" );
           break;
       default:
           printf( "It’s not 1 or 2\n" );
   }
   system("pause");
   return 0;
}
```

What happens if we miss a break?

Example 2:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
   int num;
   printf( "Type a number " );
   scanf_s( "%d", &num );
   switch( num ){
       case 1:
           printf( "It’s 1\n" );
           /* Here we are missing a break. */
       case 2:
           printf( "It’s 2\n" );
           break;
       default:
           printf( "It’s not 1 or 2\n" );
   }
   system("pause");
   return 0;
}
```

Exercises about conditional statements

NOTE: One of these exercises can be solved using the **switch** statement. Identify which one it is and use **switch** there instead of **if else**.

1. Write a program that gets a number *health* from 0 to 100. Clamp the number to 0 if it is less than 0, and to 100 if it is greater than 100. Print on the console a message to tell if the health level is low (< 20), low-medium (< 40), medium (< 60), medium-high (<80) or high (>= 80).
2. Write a program that gets a number and prints a message to say if it is an odd number or an even number. Remember that even numbers can be evenly divided by 2 without a remainder. Think about which operator is adequate to know this.
3. Write a program that takes three numbers and prints a message with those three numbers in increasing order (e.g. Input numbers: 9 2 6, output message: 2 6 9).
4. Write a program that takes an unsigned integer number *option* and prints which option it is: 0 - start game, 1 - load game, 2 - save game, 3 - settings, 4 - exit. If the option is not valid, it prints a message notifying that the option number introduced is not valid.

While (loop statement)

- In a **while** statement, the inner statement or block is repeated while the condition between parentheses is satisfied.
- When the condition between parentheses is not satisfied, (that is, evaluates to 0) the program stops repeating the inner statement or block, and continues executing after the **while** statement.
- Syntax (not valid C):

```
while (condition)
        Statement / block
```

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

int main()
{
   int i = 0;

   while (i < 3)
   {
       printf("Iteration %d\n", i);
       ++i;
   }

   system("pause");
   return 0;
}
```

For (loop statement)

- A **for** statement is another loop construction with the following syntax (not valid C):

  for (expr1**;** expr2**;** expr3**)**

  Statement / block

- It is equivalent to using **while** like this:

  expr1;
  while (expr2)
  {
  Statement / block
  expr3
  }

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

int main()
{
    int i;

    for (i = 0; i < 3; ++i)
    {
        printf("Iteration %d\n", i);
    }

    system("pause");
    return 0;
}
```

Do until (loop statement)

- A **do while** statement is another loop construction
- It **always executes the inner statement or block the first time**, no matter the result of the condition.
- At the end it evaluates a condition and if it is fulfilled, the inner statement or block is repeated again .
- Syntax (not valid C):

```
do
        Statement / block
while (condition);
```

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


int main()
{
   int i = 0;


   do
   {
       printf("Iteration  %d\n", i);
       ++i;
   } while (i < 3);


   system("pause");
   return 0;
}
```

Continue and break

C provides two keywords that modify the behavior of loop statements:

- **Continue**: It skips the remaining statements of the current iteration and evaluates the loop condition to see if it needs to iterate again.
- **Break**: It immediately exits the innermost loop statement and continues executing code after it.

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

int main()
{
   for (int i = 0; i < 10; ++i)
   {
       if (i == 5) {
           continue; // Test also break here
       }
       printf("Iteration %d", i);
   }

   system("pause");
   return 0;
}
```

Goto and labels

- C provides the infinitely-abusable **goto** statement, and labels to jump to.
- **goto** is never necessary, and in practice it is almost always easy to write code without it.
- A label has the same form as a variable name, and is followed by colon.
- The scope of a label is the entire function.

NOTE: No example code here. You can investigate on its usage if you are curious. However, it is recommended not to use this C feature, as misusing it can lead to what many people call "spaghetti code", which is difficult to maintain.

Exercises about loops

1. Write a program that computes the factorial of a given value. The formula of the factorial is this one: n! = 1 * 2 * 3 * ... * (n-1) * n.
2. Create a program that prints the next sequence:

   0, 4, 2, 6, 4, 8, 6, 10, 8, 12, 10, 14, 12, 16, 14, 18, 16, 20, 18, 22, 20, 24

   The sequence starts at 0 and finishes at 24. The first iteration adds 4, the second subtracts 2, the third adds 4, and so on.
3. Create a program that prints the first 10 numbers of the Fibonacci sequence:

   First= 1, Second= 1, Third=First+Second, Fourth=Second+Third, etc.

   The sequence printed will be: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
4. Create a program that determines if a number is perfect. A number is perfect when its value is equal to the sum of all its divisors except itself. For example, 28 is perfect because the sum of its divisors (1, 2, 4, 7, 14) is equal to itself (1+2+4+7+14 = 28).