Programming I

Arithmetic

Dimitrios Michail

Dept. of Informatics and Telematics
Harokopio University of Athens

Second Program

```c
/* Second Simple Program: add 2 numbers */

#include <stdio.h>

int main() {
    int integer1, integer2;
    int sum;

    printf("Enter first integer\n");
    scanf("%d", &integer1);

    printf("Enter second integer\n");
    scanf("%d", &integer2);

    sum = integer1 + integer2;
    printf("Sum is %d\n", sum);

    return 0;
}
```

Function `scanf()`

Defined in `stdio.h` does the opposite job from `printf()`.

Reads from the user some data. The execution of the following line:

```
scanf("%d", &integer1);
```

blocks the program until the user types an number (integer) and presses ENTER.

Function `scanf()`

specifier	Input	Example
d	read an integer as a signed decimal	392
c	character	a
f	decimal floating point	392.65
x	hexadecimal without sign	7fa
o	octal	610
s	string	sample
(space)	reads whitespace	
lf	reads a `double`	1.333
Lf	reads a `long double`	1.333

Arithmetic in C

Most programs in C execute some mathematical calculations using constants and variables, e.g.

```
int x, y;

x = 1;
y = x + 100;
```

Binary Arithmetic Operators

	operator	algebraic expression	C
addition	+	$x + 7$	`x + 7`
subtraction	-	$p - c$	`p - c`
multiplication	*	bm	`b * m`
division	/	x/y or $x \div y$	`x / y`
remainder	%	$r \mod s$	`r % s`

Parentheses are used in C just like they are used in algebraic expressions, e.g.

```
a = b * ( c + d );
```

Operators Precedence

The way that expressions are calculated depends on the precedence of the operators:

1. **Parentheses:** ()
   Calculated first, from left to right. Nested parentheses are calculated first.
2. **multiplication, division and remainder:** $*$, $/$, or %
   Calculated second from left to right.
3. **addition, subtraction:** $+$ or $-$
   If there are many, calculated from left to right.
4. **assignment:** $=$
   From right to left.

Examples

$$m = \frac{a + b + c + d + e}{5}$$

in C language

```
m = (a+b+c+d+e)/5;
```

Examples

$$y = ax^2 + bx + c$$

in C language

```
y = a * x * x + b * x + c
```

Operators Precedence

$$z = p * r \% q + w / x - y$$

$$\textcircled{6}\ \textcircled{1}\ \textcircled{2}\ \textcircled{4}\ \textcircled{3}\ \textcircled{5}$$

Unary Arithmetic Operators

- `+`

  e.g.

  ```
  y = +5;
  ```

- `-`

  e.g.

  ```
  x = -y;
  ```

- `++`

  e.g.

  ```
  x = ++y
  ```

  or

  ```
  x = y++
  ```

- `--`

  e.g.

  ```
  x = --y
  ```

  or

  ```
  x = y--
  ```

Unary arithmetic operators have higher precedence that other arithmetic (except parentheses) and are calculated from right to left.

Operators ++ --

Operator ++ increases a variable by 1 and -- decreases a variable by 1.

```
int main() {
    int x;

    x = 1;
    printf("%d\n", x);  /* prints 1 */
    x++;
    printf("%d\n", x);  /* prints 2 */
    ++x;
    printf("%d\n", x);  /* prints 3 */
    x--;
    printf("%d\n", x);  /* prints 2 */
    --x;
    printf("%d\n", x);  /* prints 1 */
}
```

Operators ++ --

From a programmer's point of view, expressions `x++` and `++x` differ only when the value of the expression is used (e.g. stored in another variable, as a parameter to a function call, etc.).

In this case the variable `x` is increased **after** (`x++`) or **before** (`--x`) the expression is assigned a value.

Operators ++ --

```
int main() {
    int x, y;

    x = 1;
    y = x++;
    printf("x=%d, y=%d\n", x, y);     /* prints x=2, y=1 */
    y = ++x;
    printf("x=%d, y=%d\n", x, y);     /* prints x=3, y=3 */
    y = x--;
    printf("x=%d, y=%d\n", x, y);     /* prints x=2, y=3 */
    y = --x;
    printf("x=%d, y=%d\n", x, y);     /* prints x=1, y=1 */
    y = y + x--;
    printf("x=%d, y=%d\n", x, y);     /* prints x=0, y=2 */
    y = y + ++x;
    printf("x=%d, y=%d\n", x, y);     /* prints x=1, y=3 */
}
```

Operators ++ --

What does the following program print?

```
int main() {
    int x, y;

    x = 3;
    y = x++;

    printf("%d %d\n", x++, ++y);
    printf("%d %d\n", ++x, ++y);
    printf("%d %d\n", y++ + ++x, --y + --x);
    printf("%d %d\n", ++y + --x, --y - ++x);
}
```

Operators Precedence

The way that expressions are calculated depends on the precedence of the operators:

1. **Parentheses:** (), `expr++` or `expr--`
   Calculated from left to right. Nested parentheses are calculated first.
2. **unary operators:**
   $+$, $-$, `++expr` or `--expr` (prefix) Calculated from right to left.
3. **multiplication, division and remainder:** $*$, $/$, or $\%$
   Calculated from left to right.
4. **addition, subtraction:** $+$ or $-$
   If there are many, calculated from left to right.
5. **assignment:** $=$
   From right to left.

Assignment Operator

The assignment operator = needs particular attention as it is calculated from right to left.

Moreover the expression `x = 5` returns the value that what assigned.

```
int main() {
    int x,y;

    y = x = 5;
}
```

In the program above, first `x=5` is executed and returns the value `5`. Afterwards the assignment `y = 5` is executed.

Other Assignment Operators

C provides various shortcut assignment operators.

operator	C statment	equivalent statement in C
+=	`c += 7`	`c = c + 7`
-=	`d -= 4`	`d = d - 4`
*=	`e *= 5`	`e = e * 5`
/=	`f /= 3`	`f = f / 3`
%=	`g %= 9`	`g = g % 9`

Operators Precedence

The way that expressions are calculated depends on the precedence of the operators:

1. **Parentheses:** (), `expr++` or `expr--`
   Calculated from left to right. Nested parentheses are calculated first.
2. **unary operators:**
   $+$, $-$, `++expr` or `--expr` (prefix) Calculated from right to left.
3. **multiplication, division and remainder:** $*$, $/$, or $\%$
   Calculated from left to right.
4. **addition, subtraction:** $+$ or $-$
   If there are many, calculated from left to right.
5. **assignment:** $=$, $+=$, $-=$, $*=$, $/=$, $\%=$
   From right to left.