

Types and operators

UPC - Videogame Design & Development - Programming I

Contents

1. Variable names
2. Basic data types and sizes (review)
3. Literals and constants
4. Declaration of variables
5. Operators

Variable names

1. Can contain letters, numbers, and the underscore “_” symbol.
2. Cannot start with a number.
3. Uppercase and lowercase letters are different (C is case sensitive)
4. Cannot be reserved keywords of the language (int, float, if, else, etc).
5. Good naming is extremely important!

// Some valid names

```
int player1Score = 4500;
```

```
int player1_score = 4500;
```

```
int _player1Score = 4500;
```

// Some invalid names

```
int 1playerScore = 4500; // Wrong, initial character is a number
```

```
int float = 4500; // Wrong, variable name is a keyword of C
```

```
int player2-Score = 4500; // Wrong, it contains '-'
```

Variable names

1. Can contain letters, numbers, and the underscore “_” symbol.
2. Cannot start with a number.
3. Uppercase and lowercase letters are different (C is case sensitive)
4. Cannot be reserved keywords of the language (int, float, if, else, etc).
5. Good naming is extremely important!

```
// Looks ok
```

```
float distance = speed * time;
```

```
// Mmmm... not very self-explanatory
```

```
float d = s * t;
```

```
// Ok, but maybe this is too verbose...
```

```
float distanceTraversedDuringLastFrame = currentCarSpeed * timeOfTheLastFrame;
```

Variable naming is kind of subjective but it's good to strive to find self-explanatory, but not too large.

Not an easy task sometimes :-P

Basic data types

- **char**: Used to express a character (check [ASCII table](#)) or a small integer.
- **int**: Used to express an [integer](#) number.
- **float**: For [real](#) numbers. Computers' approximation is called floating point.
- **double**: Same as float, but twice its precision (and size).

The **sign** and size of char and int types can be modified with the keywords **signed/unsigned** and **short/long**, respectively. For instance:

```
short int totalDistance = 5984;    // a few thousands, but fits 2 bytes
long int numberOfPixels = 2073600; // 2 megapixels, 2 bytes are not enough
```

The **size** in memory of char and int types is **platform dependent** (you can use the operator [sizeof](#) to reveal the size in bytes of a type or a variable).

Common sizes of basic data types (platform dependent!)

```
/* Integral types (signed) */
char                /* size = 1 byte */
short int           /* size = 2 bytes */
int                 /* size = 2-4 bytes */
long int            /* size = 4-8 bytes */
long long int       /* size = 8 bytes */

/* Integral types (unsigned) */
unsigned char        /* size = 1 byte */
unsigned short int   /* size = 2 bytes */
unsigned int         /* size = 2-4 bytes */
unsigned long int    /* size = 4-8 bytes */
unsigned long long int /* size = 8 bytes */

/* Floating point types */
float               /* size = 4 bytes */
double              /* size = 8 bytes */

/* C-Strings */
const char*         /* size = 1 byte * #chars */
```

Exercise. Write a program to determine the **ranges of char, short int, int, long int, and float** variables, both **signed and unsigned**. Include their **size** in bytes.

To do this exercise, **include the header files `<limits.h>` and `<float.h>`**. You can check what they provide either by opening those two files in the editor, or having a look at the following sites:

- [`<limits.h>`](#)
- [`<float.h>`](#)

Literals

Literals are fixed values (or hard-coded values) that cannot change during the execution of the program. Let us see some examples of literals in C:

```
short int index    = 1234;    // Any number like this is an integer
long int longIndex = 1236549L; // the 'L' suffix forces the constant as long
double realNumber  = 3.141516; // any value with decimals is a double by default
double realNumber2 = 1e-5;    // 1 x 1/10^5 power is double by default
float realNumber3  = 3.141516f; // the 'f' suffix forces the constant as float
int anotherInt     = 034;    // the prefix '0' indicates the number is typed in octal
int anotherInt2    = 0x54;   // the prefix '0x' indicates the number is typed in hexadecimal
char letter        = 'a';    // a character between quotes ('a') gives the variable its ASCII value
char tab           = '\n';   // there are special characters such as '\n' (new line) or '\t' (tab)
const char *str    = "Hello!"; // string constants go between double quotes "" and end in '\0'
```

Constants

Constant expressions are: literals, definition constant values, and enumerated values:

```
// Integer numbers
short int index = 1234;           // A literal is constant (the number, not the variable!)

// Preprocessor definition
#define MAX_SIZE 1024             // This is the definition of a constant value

// Enumerated types
enum boolean { FALSE, TRUE };    // They start at zero (so FALSE:0 and TRUE: 1)
int hasFinished = FALSE;         // its constant values can be assigned to integers

// We can force enumerated values to start at different values
enum days { MON=1, TUE, WED, THU, FRI, SAT, SUN };
```


Variable declaration

- All variables must be declared before using them.
- In C, declarations must happen in the beginning of functions.
- It is usually a good idea to initialize variables during their declaration:

```
float x = 3.0f; // Good!
```

```
float y;          // Will compile, but why not give it an initial value? (e.g. 0.0f)
```

- Using the const modifier, we express the intention of not changing its value (forced by the compiler):

```
const short int index = 1234; // If you try to change the value of index after its  
                               // declaration, the compiler will fail and give an  
                               // error message
```

Arithmetic operators

Most of them are quite self explanatory:

```
int var1 = 1 + 1; // var1 equals 2
int var2 = 2 - 1; // var2 equals 1
int var3 = 2 * 3; // var3 equals 6
int var4 = 4 / 2; // var4 equals 2
int var5 = 5 / 2; // var5 equals 2 (the integer part of the division)
```

Modulo operator gives the remainder of a division. It uses the character '%' (cannot be used with float/double types, **only integral types**):

```
int var6 = 5 % 2; // var6 equals 1 (the remainder of the division)

// Modulo example
int x0 = 0 % 3; // x0 equals 0
int x1 = 1 % 3; // x1 equals 1
int x2 = 2 % 3; // x2 equals 2
int x3 = 3 % 3; // x3 equals 0
int x4 = 4 % 3; // x4 equals 1
// ...
```

Relational operators

Higher precedence

Unary operators:

- `!` (negation)

Relational operators:

- `<` (less than)
 - `<=` (less or equal than)
 - `>` (greater than)
 - `>=` (greater or equal than)
- } Same precedence

Equality operators:

- `==` (equality)
 - `!=` (inequality)
- } Same precedence

Logic operators:

- `&&` (logical and)
 - `||` (logical or)
- } `&&` has higher precedence than `||`

Lower precedence

// Basic examples

```
int val1 = !0; // evaluates to 1
```

```
int val2 = !1; // evaluates to 0
```

```
int val3 = 1 < 2; // evaluates to 1
```

```
int val4 = 1 > 2; // evaluates to 0
```

```
int val5 = 5 == 5; // evaluates to 1
```

// Let's combine some operators

```
int isAlive = health > 0 || !isZombie;
```

// Arithmetic operators have precedence

// over relational and logic operators

```
int isAlive = currentHealth - remainingDamage > 0;
```

Exercises

1. Write a program that defines the constants *BIRTH_YEAR* and *CURRENT_YEAR* (use the *#define* directive) and prints your age in days.
2. Write a program that defines a distance *MAX_DIST* in meters (use the *#define* directive) and prints the same distance in meters.
3. Write a program that gets two numbers from the console input (use the function *scanf_s()*), and prints the result of adding, subtracting, multiplying and dividing them (one line each result).
4. Write a program that defines the *PI* constant (use the *#define* directive), takes a number *n* from the console input (use the function *scanf_s()*), and computes the area of a circle of radius *n*.
5. Write a program that gets a number from the console input (use *scanf_s*), stores it into a variable called *days*, and prints this amount in years. Print also the remainder in days.