EXPRESSIONS in C

1. Overview

The learning objective of this lab is:

- To understand basic data types in C.
- To understand and construct expressions in C.
- To use operators in small programs.
- To understand the precedence of operators.

2. Brief theory reminder

2.1. Basic data types in C

Wikipedia: "In the C programming language, data types refers to an extensive system for declaring variables of different types. The language itself provides basic arithmetic types and syntax to build array and compound types. Several headers in the standard library contain definitions of support types, that have additional properties, such as exact size, guaranteed.". Table 1 Provides a summary of the basic data types.

Table 1. Summary of the basic data types in C.

Type	Explanation
`char`	smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned depending on the implementation.
`signed char`	same size as `char`, but guaranteed to be signed.
`unsigned char`	same size as `char`, but guaranteed to be unsigned.

`short`
`short int`
`signed short`
`signed short int`
	short signed integer type. At least 16 bits in size.

`unsigned short` `unsigned short int`	same as `short`, but unsigned.
`int` `signed int`	basic signed integer type. At least 16 bits in size.
`unsigned` `unsigned int`	same as `int`, but unsigned.

long long int signed long signed long int	*long* signed integer type. At least 32 bits in size.
unsigned long unsigned long int	same as `long`, but unsigned.
long long long long int signed long long signed long long int	*long long* signed integer type. At least 64 bits in size. Specified since the C99 version of the standard.
unsigned long long unsigned long long int	same as `long long`, but unsigned. Specified only in C99 version of the standard.
float	(single precision) floating-point type. Actual properties unspecified, however on most systems this is IEEE 754 single precision floating point format.
double	double precision floating-point type. Actual properties unspecified, however on most systems this is IEEE 754 double precision floating point format.
long double	extended precision floating-point type. Actual properties unspecified. Unlike types `float` and `double`, it can be either 80-bit floating point format, the non-IEEE "double-double" or IEEE 754 quadruple precision floating-point format if a higher precision format is provided, otherwise it is the same as double. See this page for details.

Properties of integer types

- `CHAR_BIT` – size of the `char` type in bits (at least 8 bits)
- `SCHAR_MIN`, `SHRT_MIN`, `INT_MIN`, `LONG_MIN`, `LLONG_MIN`(C99) – minimum possible value of signed integer types: `signedchar`, `signed short`, `signed int`, `signed long`, `signed long long`
- `SCHAR_MAX`, `SHRT_MAX`, `INT_MAX`, `LONG_MAX`, `LLONG_MAX`(C99) – maximum possible value of signed integer types: `signedchar`, `signed short`, `signed int`, `signed long`, `signed long long`
- `UCHAR_MAX`, `USHRT_MAX`, `UINT_MAX`, `ULONG_MAX`, `ULLONG_MAX`(C99) – maximum possible value of unsigned integer types:`unsigned char`, `unsigned short`, `unsigned int`, `unsigned long`, `unsigned long long`
- `CHAR_MIN` – minimum possible value of `char`
- `CHAR_MAX` – maximum possible value of `char`
- `MB_LEN_MAX` – maximum number of bytes in a multibyte character

Properties of floating-point types

- `FLT_MIN`, `DBL_MIN`, `LDBL_MIN` – minimum normalized positive value of `float`, `double`, `long double` respectively
- `FLT_TRUE_MIN`, `DBL_TRUE_MIN`, `LDBL_TRUE_MIN` (C11) – minimum positive value of `float`, `double`, `long double`respectively
- `FLT_MAX`, `DBL_MAX`, `LDBL_MAX` – maximum finite value of `float`, `double`, `long double` respectively
- `FLT_ROUNDS` – rounding mode for floating-point operations
- `FLT_EVAL_METHOD` – evaluation method of expressions involving different floating-point types (only available in C99)
- `FLT_RADIX` – radix of the exponent in the floating-point types
- `FLT_DIG`, `DBL_DIG`, `LDBL_DIG` – number of decimal digits that can be represented without losing precision by `float`,`double`, `long double` respectively
- `FLT_EPSILON`, `DBL_EPSILON`, `LDBL_EPSILON` – difference between 1.0 and the next representable value of `float`, `double`,`long double` respectively
- `FLT_MANT_DIG`, `DBL_MANT_DIG`, `LDBL_MANT_DIG` – number of `FLT_RADIX`-base digits in the floating-point significand for types `float`, `double`, `long double` respectively
- `FLT_MIN_EXP`, `DBL_MIN_EXP`, `LDBL_MIN_EXP` – minimum negative integer such that `FLT_RADIX` raised to a power one less than that number is a normalized `float`, `double`, `long double` respectively
- `FLT_MIN_10_EXP`, `DBL_MIN_10_EXP`, `LDBL_MIN_10_EXP` – minimum negative integer such that 10 raised to a power one less than that number is a normalized `float`, `double`, `long double` respectively
- `FLT_MAX_EXP`, `DBL_MAX_EXP`, `LDBL_MAX_EXP` – maximum positive integer such that `FLT_RADIX` raised to a power one more than that number is a normalized `float`, `double`, `long double` respectively
- `FLT_MAX_10_EXP`, `DBL_MAX_10_EXP`, `LDBL_MAX_10_EXP` – maximum positive integer such that 10 raised to a power one more than that number is a normalized `float`, `double`, `long double` respectively
- `DECIMAL_DIG` – minimum number of decimal digits needed to represent all the significant digits for `long double`.[4] The value is at least 10. (only available in C99)

2.2. Defining an expression

An **expression** is composed of a single operand or more **operands** glued together by **operators**.
An operand may be:

- A constant, e.g. 3, 20.4, 'd'.
- A symbolic constant, e.g. PI.
- A name for a scalar variable, e.g. a, b, x.
- A name for an array variable.
- A name for a structure.
- A name for a type.
- An indexed variable, e.g. a[i], a[7].

- A name for a function.
- A reference to an element of a structure.
- A function call.
- An expression surrounded by parenthesis.

An **operand** has a **value** and a **type** associated with it.
Operators may be unary or binary.
When evaluating an expression, the following are taken into account:
- Operator precedence.
- How operators of the same precedence associate.
- Default conversion rules.

2.3. Operators

The operators of C/C++ belong to one of the following categories:

- Arithmetic operators:
  - Unary operators: **+, –**
  - Binary multiplicative operators: ***, /, %**
  - Binary additive operators: +, –
- Relational operators: **<, <=, >, >=**
- Equality operators: **= =, !=**
- Logic operators: **!, &&, | |**
- Logic operators on bits: **~, <<, >>, &, ^, |**
- Assignment operators: **=, /=, *=, %=, +=, –=, <<=, >>=, &=, ^=, |=**
- Pre- and post increment operators : **++, —**
- Type cast operators: **(type)**
- Dimension operators: **sizeof**
- Referencing (address) operators: **&**
- Parenthesis operators: **( ), [ ]**
- Conditional operators: **?, :**
- Comma operator: **,**
- Dereferencing operator: *****
- Structure component access operator: **. , –>**

Note: C++ has a number of additional operators:

- Resolution operator: **::**
- An operator for the reference type: **&**
- An operator to allocate/deal locate the heap:
- Type constructor/destructor invocation operators: **new/delete**

Operator precedence, in the decreasing order of their priority, is given in the table below:

Priority	Operators
15-maximum	**( ) [ ] • ->**
14	**+(unary) -(unary) &(unary) *(unary) ++ -- (type) sizeof ! ~**
13	***(binary) / %**
12	**+(binary) -(binary)**
11	**<< >>**

<table>
<tr><td>10</td><td>< <= > >=</td></tr>
<tr><td>9</td><td>= = !=</td></tr>
<tr><td>8</td><td>&(binary)</td></tr>
<tr><td>7</td><td>^</td></tr>
<tr><td>6</td><td>|</td></tr>
<tr><td>5</td><td>&&</td></tr>
<tr><td>4</td><td>||</td></tr>
<tr><td>3</td><td>? :</td></tr>
<tr><td>2</td><td>= <<= >>= += -= *= /= %= &= ^= |=</td></tr>
<tr><td>1-minimm</td><td>,</td></tr>
</table>

Operators on the same row have identical precedence.

Operators are **associated** in the left-to-right order, except for the unary, conditional, and assignment operators which associate right-to-left.

2.4. Default conversion rules

The default conversion rules are stated as follows:

- If a binary operator is applied to operands of the same type, then the result will take the identical type of the operands.
- If a binary operator is applied to operands of the different types, then the operand of an inferior (more restrictive) type is converted by default to the superior type of the other operand, and the result gets the superior type.

Type priority, in decreasing order, is the following:

- long double (maximum priority);
- double;
- float;
- unsigned long;
- long;
- int (minimum priority).

3. Lab Tasks

3.1. Write a program to calculate the value ***z***= ***x**********y***, for variables ***x*** and ***y*** of the type double. Hint: look for function `pow`.

3.2. Explain the difference between real (floating point) division and the integer division.

3.3. Write a program that reads the value of an angle expressed in degrees and calculates the values for its sine, cosine and tangent.

3.4. Write a program that reads a positive integer number in the range [1600, 4900]. Knowing that that number represents a year, check whether that year is bissextile or not.

3.5. Using conditional expressions, write a program which reads a real value for ***x*** and then computes the value for the function:

$$f(x) = \begin{cases} x^2 - 7x + 4 & if \quad x < -2 \\ 0 & if \quad x = -2 \\ x^2 + 5x - 2 & if \quad x > -2 \end{cases}$$

3.6. Write a program which reads a real number ***x***, representing a measurement for an angle in Radians, and then converts it to degrees, minutes, and seconds.

3.7. Write a program to simulate the operation of a counting clock (indicating the hour, minute, and second).

3.8. Write a program to show the number of bytes the C/C++ primitive data types take in the computer memory.

3.9. Convert to binary, by computation, your birth year and the current year.
   a) Show how the two years are represented as data of type int.
   b) Show the effect the shift operations: left shift by 4 bits, right shift by 2 bits, and 1's complement applied to this data.
   c) Show the effect of the bitwise operations **&, ^, | ,** upon the provided data.
   d) Write a program to check the correctness your calculations.

3.10. Write a program which effects arithmetic operations upon two data items, one of which is an integer, and the other a real number. Execute this program using values which yield results outside the internal representation limits. What happens when limits are over(under)flowed?

3.11. Write a program which reads the integer numbers ***a, b, c, d*** and outputs the highest value of fractions ***a***/***b*** and ***c***/***d***.

3.12. Write a program to determine the relative position of a straight line in relationship with a given circle. Your program will read: the coordinates of the center of the circle, its radius, and the coordinates for two points located on the straight line. (See: Intersection_Euclidean_geometry)

3.13. Write a program which reads the planar coordinates of the vertices of a triangle, and then describes the relative position (i.e. above, below, left, right, inside, on border) of a point in the same plane, given by its coordinates, in relationship with this triangle. Hint: start by checking whther the point is on one of the 3 straight line segments. Then check whether the point is interior (see Point in Polygon Test). Otherwise the point is outside the triangle; figure where, by looking at x and y coordinates: above, below, left, right.

3.14. Write a program to calculate the ideal weight of a person using the following formulas:

W_male = 50 + 0.75*(height-150) + (age-20)/4;
W_female= W_male - 10;

Sex, height (in centimeters), and age (in years) are to be read.

3.15. Write a program to convert Cartesian coordinates of a given point to polar coordinates.

Hints. The two polar coordinates r and θ can be converted to the Cartesian coordinates x and y by using the trigonometric functions sine and cosine:
$x = r\cos\theta$

$y = r \sin \theta,$
while the two Cartesian coordinates x and y can be converted to polar coordinate r by
$r = \sqrt{x^2 + y^2}$ (by a simple application of the Pythagorean theorem).
To determine the angular coordinate θ, the following two ideas must be considered:
For $r = 0$, θ can be set to any real value.
For $r \neq 0$, to get a unique representation for θ, it must be limited to an interval of size 2π. Conventional choices for such an interval are $[0, 2\pi)$ and $(-\pi, \pi]$.
To obtain θ in the interval $[0, 2\pi)$, the following may be used (**arctan** denotes the inverse of the tangent function):

$$\theta = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) + 2\pi & \text{if } x > 0 \text{ and } y < 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ \frac{3\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

To obtain θ in the interval $(-\pi, \pi]$, the following may be used:

$$\theta = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

One may avoid having to keep track of the numerator and denominator signs by use of the **atan2** function, which has separate arguments for the numerator and the denominator.