Tutorial LaTeX

Petru Rusu

Contents

1 The starting

1.1 Low level vs high Level

when you write a programs in any language, this program have to be converted in machine language, after the CPU will execute the task. There are two type of language: low level and high level.

- **Low level:** is really close to the hardware and we should have a good knowledge about CPU.
  For example we have to know that when we sum 1 + 1 we create a space in RAM that occupies 20 bits, where 4 is for declare the +, 1 byte is for memorize 1 and the other byte is for memorize the last 1, it is the arrays. They usually use a compiler

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

- **High level:** allow developers to write more abstract and readable code, away from specific machine or hardware details. This fosters better code understanding and facilitates software maintenance and updates. but usually they are slower than low level code. They usually use a interpreter

1.1.1 compiler vs interpreter

- **Compiler:** C code→ Compiler→ machine code.
  The code first is translated, and after is executed.
  But it means that is most difficult to correct errors because will show us all errors after the compiling. we do not need about the source code. the compiler do not stay in memory when we execute the code.

- **Interpreter:** code→ interpreter→ out put
  The while the code is translated the machine execute, so the machine execute line per line. Do not produce object code.
  It means that is most easier to correct errors because when it encounters an error it will stop compilation and show it to us one line at a time. and we need about the source code. Interpreter have to stay in memory because is he that execute the code.

1.2 What is Compilation?

Is the process of converting the source code into object code. It is done with the help of the compiler. The compiler checks for errors.

1.2.1 How Does It Work?

- Processor: Removes the comments and expands the library with the code. `hello.c` → `hello.i` (expanded code), it means that replaces all the line that start with # with the value, and remove #
- Compiler: Converts this code into assembly code pre-processed code, and check for syntax errors
  assembly code `hello.i` → `hello.s` (assembly code)
- Assembler: Converts the assembly code into machine code by using an assembler. `hello.s` → `hello.o`
- Linker: Its main work is to combine the object code of the library file with the object code of our program. It links the object code of these files to our program.

2 variables in c

A variable is a name for the memory location, is used for store data.

2.1 constants

#define pi 3.14 is a symbolic constant, is used for define a fix constant.
In C we have 2 type of constants:

- the first is the numeric:
  - integer numbers, that can be represented in different bases:
    - decimal → base10 = 0-9, by default
    - octal → base8 = 0-7, add a 0 before (05)
    - hexadecimal → base16 = 0-15, add 0x before (0xf)
  - float numbers [decimal part, decimal point, fractional part].
    - 12.54
- the second is the

  characters
  - the first is the single character constant
    - are like 'a', 'b', '2'
    - 5 ≠ '5'
    - we can represent a number like a char following the ASCII rule

```
printf("%d", 'a');//represent a char in number
printf("%c", 97);//represent a number in a char
```

  - string constants:

* are written inside double quotes: "i am a string"
* 'a' $\neq$ "a"
* at the finish of every string put a \0

2.2 how to create them

The syntax for declare the variable is: `type variable-list`; after that we have to initialize a variable saying what data it have to contain: `variable = 10;`

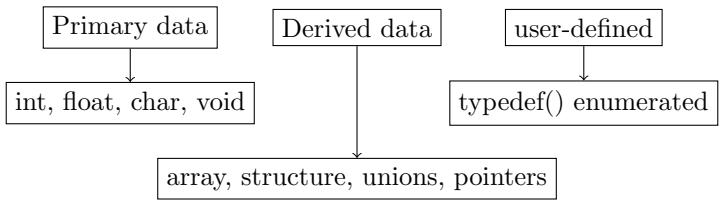
```
int main()
{
    int a;  // declaration
    a = 10; // initialization

    // we can also do this
    int a = 100, b = 200;

    const int c = 10;
    c = 12; // it is a error, we cannot change a constant
}
```

2.3 rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whites-pace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

2.4 Data Type in C

the c language is a typed language; it means that you have to write what type and what size of data the variable have to contain. datatype are divided in:



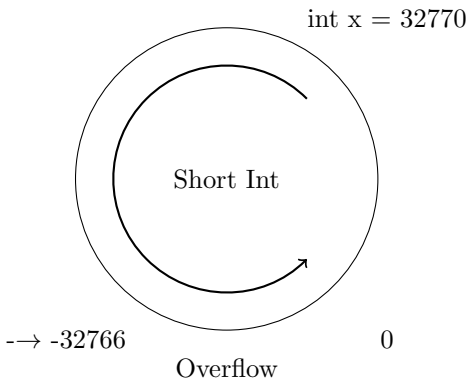
2.4.1 Primary data:

the value of this data type can change, it depend if u are using a 32 bit machine, a 16 bit machine, a 64 bit machine

Name of data	**Range of number**	**bytes**	**Symbol**
char	-128 to 127	1	%c
unsigned char	0 to 255	1	%c
signed char	-128 to 127	1	%d
short int	-32768 to 32767	2	%h
unsigned short int	0 to 65535	2	%h
int	-2147483648 to 2147483647	4	%d
unsigned int	0 to 4294967295	4	%u
long int	-2147483648 to 2147483647	4	%ld
unsigned long int	0 to 4294967295	4	%lu
long long int	-9.223×10^{18} to 9.223×10^{18}	8	%lld
unsigned long long int	0 to 18446744073709551615	8	%llu
float	6-9 digits	4	%f
double	15-17 digits	8	%lf
long double	18-19 digits	16	%Lf
void	represents absence	0	—

2.4.2 What happens if we take a value that is over the range?

For example take the short int data type and say that: `short x = 32768`, it will not get us a error, but will get us the position of the next value, like a wheel



3 Operators

expressions are a set of operands and operators

```
1  int main()
2  {
3    int a = 5    +      5; // --> expressions
4    /*     | |    |      |
5           |-+-operator  |
6             |           |
7             |      operand
8         operand
9
10   ----------------------------------
11   but there are 3 type of operators: */
12
13   return 0;
14 }
```

3.1 type of operands:

1 Unary: only 1 operand are used

2 binary: 2 operands used

3 Tetrary: 3 operands used

3.1.1 Unary

- unary minus = -10, -5, -2.3
- increment or decrement operators = ++, —; ++a and a++ are differents
- logical not = !
- the address of = &, for take the memory address
- for take the saze of something = sizeof()