Computer Programming 1

Basic concepts

Outline

- Basic concepts

What is a programming language? (recall)

- English is a natural language
  - It has words, symbols and (grammatical) rules
  - A programming language also has words, symbols and (grammatical) rules

Language = Syntax + Semantics

Syntax

- It describes ***composition and structure of a program***
- It is the set of grammatical rules
- It is a collection of rules that specifies the structure/form of the code

Semantic

- It describes the ***meaning of a program***
- It is a collection of rules that specifies the interpretation of the code and the meaning associated to words, symbols, and code

Sequence of words

- C++ words are composed of:
  - Letters:

    _ a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  - Digits:

    0 1 2 3 4 5 6 7 8 9
  - Special characters:

    ! ^ \& * ( ) - + = { } | ~ [ ] \ ; ’ : " < > ? , . /
- The spaces ca be composed of:
  - Space: “ ”
  - Tabular: “ ”
  - New line character: “
    ”

Comments

- Comments
  - On a single line

    // This comment is on a line
  - Grouped by “/*” and “*/” (also in more lines)

    /* This comment is
    over more lines*/

Identifiers

- The entities of a C+ program must have meaningful names
- In the simplest case, the names are identifiers freely chosen
  - There are some “reserved words” that cannot be used as names
- An identifier is a term initiating with a letter
  - 1var is not a valid identifier while var1 is valid

Naming conventions exist!

Note

- The character “_” is a letter
- C++ distinguishes between lower and upper case (it is case sensitive)
  - E.g., Fact is different from fact
- Camel case notation is often used (mix of lower and upper case)
  - E.g., name surname → NameSurname → nameSurname
  - Alternatively, you can do name_surname

Keywords

- The keywords of the language are symbols which meaning is pre-defined by the language and that cannot be changed or redefined.
- The use of such keywords is reserved

Main C++ Keywords:

asm auto **break case catch char class const continue default** delete **do double else enum** extern **float for** friend goto **if** inline **int long** new operator private protected public register **return short** signed sizeof **static struct switch** template this throw try **typedef** union **virtual void** volatile **while** . . .

Literal expression

- They denote constant values, often they are called only "literals“ (or "nameless constants", or "constant values")
- They can be:
  - Constant character (a single character)
  - Constant string (sequence of characters)
  - Integer numerical constants
  - Real numerical constants

Literal expression: constant characters

- They denote constant values
- A character constant is represented by enclosing the corresponding character between quotes
  - E.g., the constant 'n' represents the character *n*
- The control characters are represented by **special combinations** called escape sequences that start with an inverted slash (backslash '\')
  - i.e., escape sequences are used to represent special characters within character literals
  - Note: '\n' could have different behavior depending on the Operating System

Name	Shortly	Escape Sequence
New line	NL (LF)	\n
Orizontal tabular	HT	\t
Vertical tabular	VT	\v
Space back	BS	\b
Carriage return	CR	\r
Form feed	FF	\f
Acoustic signal	BEL	\a
Backslash	\	\\
Quotation mark	'	\'
Double quotation mark	"	\"

Example

- {../L03_00_EXAMPLE_BASE/escape.c}

https://en.cppreference.com/w/cpp/language/escape

Literal expression: constant string

- They denote constant values
- A constant string is a list of characters between double quotation marks
  - E.g., “Hello!”
- They can include spaces, non-alphanumeric characters, escape sequences
  - E.g., “Hello World!\n”

Literal expression: constant characters - examples

{../L03_00_EXAMPLE_BASE/escape.c}

```
using namespace std;
#include <iostream>

int main ()
{
  cout << "1. \\n means: " << "[\n]" << endl; //new line
  cout << "2. \\t means: " << "[\t]" << endl; //ad tab space
  cout << "3. \\v means: " << "[\v]" << endl; //vertical tab for printer (no monitor)
  cout << "4. \\b means: " << "[\b]" << endl; //backspace - it moves the cursor back one space, then writes a space to erase the character,  backspaces again, and writes the new character at the old position
  cout << "5. \\r means: " << "[\r]" << endl; //carriege return – it turns to the beginning of the current line
  cout << "6. \\nr means: " << "[\n\r]" << endl; //new line + carriege return
  cout << "7. \\f means: " << "[\f]" << endl; //increase of space in printer (no monitor)
  cout << "8. \\a means: " << "[\a]" << endl; //audio
  cout << "9. \\\\ means: " << "[\\]" << endl; //backslash
  cout << "10. \\\' means: " << "[\']" << endl; //quote character
  cout << "11. \\\" means: " << "[\"]" << endl; //double-quote character
  return 0;
}
```

```
$ g++ escape.cc
$ ./a.out
1. \n means: [

]
2. \t means: [        ]
3. \v means: [?]
4. \b means: ]
]. \r means: [
6. \nr means: [

]
7. \f means: [?]
8. \a means: []
9. \\ means: [\]
10. \' means: [']
11. \" means: ["]
```

Representation of numbers (1)

- An integer is a numerical literal (associated with a number), and it is represented by a sequence of digits which come interpreted as:
  - Decimal (base 10): a non-zero digit followed by zero or more decimal digits (default)
    - E.g., Decimal: 1, -9, 22
  - Octal (base 8): if the sequence starts with 0
    - E.g., Octal: 0, 010, 021, 077, 033
  - Hexadecimal (base 16): if the sequence starts with 0x
    - E.g., Hexadecimal: 0x7f, 0x2a, 0x521

Example

- {../L03_00_EXAMPLE_BASE/repr-numbers2.cc}

Summing up	
2 is equal to?	2
02 is equal to?	2
0x2 is equal to?	2
010 is equal to?	8
08 is equal to?	

Representation of numbers (2)

- A real number (floating-point literal) is represented by using the '.' (dot) to separate the integer part form the fractional part written in decimal form and the letter 'e' (or 'E') to separate the fixed-point part from the exponent
  - E.g., 0.0000234
  - E.g., -1235.6e-2 represents -12,356
  - The Anglo-Saxon (Scientific notation) is used for presenting real number
    - e/E is used to mean "10 to the power of…"

Example

- {../L03_00_EXAMPLE_BASE/repr-numbers.cc}

Representation of numbers (3) - example

```
using namespace std;
#include <iostream>

int main ()
{

 cout << "12 means: " << 12 << endl;
 cout << "012 means: " << 012 << endl;
 cout << "0X12 means: " << 0X12 << endl;
 cout << "-12.45 means: " << -12.45 << endl;
 cout << "-12.451e3 means: " << -12.451e3 << endl;
 cout << "-12.451e-3 means: " << -12.451e-3 << endl;

 return 0;
}
```

```
$ g++ repr-numbers.cc
$ ./a.out
12 means: 12
012 means: 10
0X12 means: 18
-12.45 means: -12.45
-12.451e3 means: -12451
-12.451e-3 means: -0.012451
```

{../L03_00_EXAMPLE_BASE/repr-numbers.cc}

Representation of numbers (4) - example

```
using namespace std;
#include <iostream>

int main ()
{

 cout << "12 means: " << 12 << endl;
 cout << "012 means: " << 012 << endl;
 cout << "0X12 means: " << 0X12 << endl;
 cout << "-12.45 means: " << -12.45 << endl;
 cout << "-12.451e3 means: " << -12.451e3 << endl;
 cout << "-12.451e-3 means: " << -12.451e-3 << endl;
 cout << endl;

  //cout << "08 means: " << 08 << endl; //compilation error
  cout << 4/3 << endl; //print 1
  cout << 4.0/3.0 << endl; //print 1.333
  cout << 4/3.0 << endl; //print 1.333

 return 0;
}
```

```
$ g++ repr-numbers.cc
$ ./a.out
12 means: 12
012 means: 10
0X12 means: 18
-12.45 means: -12.45
-12.451e3 means: -12451
-12.451e-3 means: -0.012451


1
1.33333
1.33333
```

{../L03_00_EXAMPLE_BASE/repr-numbers2.cc}

Representation of numbers (4) - example

```
using namespace std;
#include <iostream>

int main ()
{

 cout << "12 means: " << 12 << endl;
 cout << "012 means: " << 012 << endl;
 cout << "0X12 means: " << 0X12 << endl;
 cout << "-12.45 means: " << -12.45 << endl;
 cout << "-12.451e3 means: " << -12.451e3 << endl;
 cout << "-12.451e-3 means: " << -12.451e-3 << endl;
 cout << endl;

  //cout << "08 means: " << 08 << endl; //compilation error
  cout << 4/3 << endl; //print 1
  cout << 4.0/3.0 << endl; //print 1.333
  cout << 4/3.0 << endl; //print 1.333

 return 0;
}
```

```
$ g++ repr-numbers.cc
$ ./a.out
12 means: 12
012 means: 10
0X12 means: 18
-12.45 means: -12.45
-12.451e3 means: -12451
-12.451e-3 means: -0.012451

1
1.33333
1.33333
```

Notes

1.2 is equal to?	1.2
4/3 is equal to?	1
4.0/3.0 is equal to?	1.3
4/3.0 is equal to?	1.3

{../L03_00_EXAMPLE_BASE/repr-numbers2.cc}

Operators

- Some special characters, and their combination, are used as operators
- Operators are used to denote specific operations in the expressions
  - E.g., 2*3+4

Main C++ Operators

```
+ - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &=
|= << >> >>= <<= == != <= >= && || ++ -- , ->* ->
. * :: () [] ?:
```

Properties of operators

- The position with respect to its arguments:

  An operator is said:
  - *Prefix*: if it precedes the arguments
  - *Postfix*: if it follows the arguments
  - *Infix*: if it is among the arguments
- The number of arguments (arity)
- The precedence (or priority) in the execution order
  - E.g., 1+2*3 is computed as: 1+(2*3) and not as (1+2)*3
- Associativity: the order in which operators of the same priority are executed
  - The operators can be associative from right or from left

Separators

- Separators are punctuation symbols, which indicate the end of an instruction, separate elements of lists, group statements or expressions, etc.
- Some characters, such as the comma, can be both separators and operators, depending on the context
- The brackets must always be used in pairs, as in common mathematical use

Main C++ Separators

( ) , ; : { }