Introduction to Programming (in C++)

Data types and visibility

Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. Computer Science, UPC

Outline

- Data types
- Type conversion
- Visibility

Data types

- A data type specifies:
  - The set of values that data of that type can have (e.g. integer, real, character, Boolean, colour, Greek letter, city, etc.)
  - The type of operations that can be performed with the data. For example, two integer numbers can be added, the population of a city can be calculated, etc.

Basic data types in C++ (int)

- Integer (int). Represent the set of integer numbers.
  - In practice, computers have a limitation representing integer numbers.
  - For a 32-bit machine, **int** can represent the numbers in the interval $[-(2^{31}-1), 2^{31}-1]$.
    [-2147483648, 2147483647]
  - Arithmetic operators: +, -, *, /, %
    Integer division and remainder: 13 / 3 = 4, 13 % 3 = 1

Basic data types in C++ (double)

- Real (double). Represent the set of real numbers.

  – In practice, computers can only represent real numbers in a certain interval and with a certain accuracy.

  – IEEE 754-1985 standard, double-precision 64 bit:
    - Numbers closest to zero: $\pm 5 \times 10^{-324}$
    - Numbers furthest from zero: $\pm 1.797693134862315 \times 10^{308}$
    - Special representations for 0, $+\infty$ and $-\infty$
    - See http://en.wikipedia.org/wiki/IEEE_754-1985

  – Arithmetic operators: +, -, ∗, /
  Real division: 13.0 / 4.0 = 3.25

Basic data types in C++ (bool)

- Boolean (bool). Represent logic values.
  - Values: ***false*** and ***true***
  - Operators: ***not***, ***and***, ***or***.

x	***not*** *x*
false	*true*
true	*false*

x	*y*	*x* ***and*** *y*
false	*false*	*false*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

x	*y*	*x* ***or*** *y*
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*true*

Basic data types in C++ (bool)

- Properties of Boolean algebra
  - Commutativity:
    - a and b = b and a
    - a or b = b or a
  - Associativity:
    - (a and b) and c = a and (b and c)
    - (a or b) or c = a or (b or c)
  - Distributivity:
    - a and (b or c) = (a and b) or (a and c)
    - a or (b and c) = (a or b) and (a or c)
  - Double negation:
    - not (not a) = a
  - De Morgan’s law:
    - not (a and b) = (not a) or (not b)
    - not (a or b) = (not a) and (not b)

Basic data types in C++ (char)

- Character (char). Represent letters, digits, punctuation marks and control characters.

- Every character is represented by a code (integer number). There are various standard codes:
  - American Standard Code for Information Interchange (ASCII)
  - Unicode (wider than ASCII)

- Some characters are grouped by families (uppercase letters, lowercase letters and digits). Characters in a family have consecutive codes: 'a'…'z', 'A'…'Z', '0'…'9'

- Operators: given the integer encoding, arithmetic operators can be used, even though only addition and subtraction make sense, e.g. 'C'+1='D', 'm'+4='q', 'G'-1='F'.

Basic data types in C++ (char)

Bits												
b_7 →					0	0	0	0	1	1	1	1
b_6 →					0	0	1	1	0	0	1	1
b_5 →					0	1	0	1	0	1	0	1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FC	,	<	L	\	l	/
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII code

Basic data types in C++ (string)

- Strings (string). Represent sequences of characters.

- Examples
  - "Hello, world!", "This is a string", ":-)", "3.1416"
  - "" is the empty string (no characters)
  - 'A' is a *character*, "A" is a *string*

- Note: use **#include <string>** in the header of a program using strings.

Relational operators

- The values of most data types can be compared using relational operators:

  == != > >= < <=

- Relational operators return a Boolean value (*true* or *false*)

- Examples
  - **5 == 5** is *true*, **5 == 6** is *false*, **5 != 6** is *true*
  - **3.1416 <= 7** is *true*, **-5.99 >= 0.1** is *false*
  - **'J' <= 'K'** is *true*, **'a' == 'A'** is *false*
  - **"Obama" == "Bush"** is *false*, **"Bush" == "Bush"** is *true*, **"Bush" < "Obama"** is *true*, **"book" < "booking"** is *true*

    (relational operators use lexicographical order in strings)

Variable declarations

- A variable is declared as:

  type variable_name;

- Examples

  ```
  int population;
  double distance;
  string my_name;
  ```

- Several variables can be declared together:

  ```
  int age, children, cars;
  ```

- After its declaration, the value of a variable is undefined (unknown).

Expressions

- Expression: a combination of literals, variables, operators and functions that is evaluated and returns a value

- Examples:

```
a + 3*(i - 1)                    → int
sqrt(x)*log(4*n)                 → double
(i - 3) <= x                     → bool
(a != b) and (s <= "abc")        → bool
```

Expressions

- The operands used in expressions must be consistent with the operators.

```
int a, b, n;
…
(a <= b)  +  n
```

(Incorrect expression: semantic error)

bool   int

cannot add bool to int

Expressions

- Operators in expressions are evaluated according to certain rules of precedence

Unary	+, - , not
Multiplicative	∗ / %
Additive	+ -
Relational (inequalities)	> >= < <=
Relational (equalities)	== !=
Conjunction	and
Disjunction	or

- Example: $3 + 4*5 \;\; != \;\; (3 + 4)*5$
- Use parenthesis to change the precedence or when you are not sure about it.

TYPE CONVERSION

Type conversion

- Consider the following code:

```
int i = 5;
char a = 'B';
double x = 1.5;
i = i + x;
if (i) x = 5*a;
```

Type conversion

- In many programming languages, the compiler would report several type errors. Possibly:

```
int i = 5;
char a = ‘B’;
double x = 1.5;
i = i + x;
if (i) x = 5*a;
```

Type conversion

- In C++, there would be no errors in this fragment of code:

```
int i = 5;
char a = ‘B’;
double x = 1.5;
i = i + x;  // i gets the value 6
if (i) x = 5*a;
// the condition of the if statement
// would be true and x would get 5
// multiplied by the code of ‘B’
// converted into double
```

Type conversion

- As a general rule, using implicit type conversions is not considered to be a good practice because:
  - The code is less readable.
  - The code is less reliable, since unintentional errors may be introduced and they may be difficult to debug.

- Recommendation: to operate with different types, use explicit type conversions

  `char(i), int('a'), double(i)`

- Never use statements that depend on a particular encoding:
  - Wrong: `c == 65,  c == char(65),  int(c) == 65`
  - Correct: `c == 'A'`

Type conversion

- Arithmetic operations between integer and real values usually imply an implicit conversion into real values.

- Be careful:

```
int i=3, j=2;
double x;
x = i/j;            // x = 1.0
x = i/double(j);    // x = 1.5
x = double(i)/j;    // x = 1.5
x = double(i/j);    // x = 1.0
x = i/2;            // x = 1.0
x = i/2.0;          // x = 1.5
```

VISIBILITY

Visibility of variables

- Variables are only visible after their declaration and in the block they have been declared.

- Blocks can include other blocks. The variables of the outer blocks are visible, a priori, in the inner blocks.

- A variable declared in an inner block masks the variables with the same name declared in outer blocks.

Visibility of variables

```
{
    // a and b are not visible
    int a = 1, b = 20;
    // a and b are visible
    cout << a;                        // writes 1
    {
        // c is not visible, a and b are visible
        cout << a + b;                // writes 21
        int b = 3, c = 4;
        // b and c are visible,
        // but the outer b is not visible
        cout << b;                    // writes 3
        cout << c;                    // writes 4
    }
    // c is not visible
    cout << b;                        // writes 20
}
```