Introduction to Programming (in C++)

Data types and visibility

Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. Computer Science, UPC
• 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: ±5 × 10⁻³²⁴
    • Numbers furthest from zero: ±1.7976931348623157 × 10³⁰⁸
    • Special representations for 0, +∞ and -∞

  – 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*.
Basic data types in C++ (bool)

• Properties of Boolean algebra
  – **Commutativity:**
    • a \(\text{and}\) b = b \(\text{and}\) a
    • a \(\text{or}\) b = b \(\text{or}\) a
  – **Associativity:**
    • (a \(\text{and}\) b) \(\text{and}\) c = a \(\text{and}\) (b \(\text{and}\) c)
    • (a \(\text{or}\) b) \(\text{or}\) c = a \(\text{or}\) (b \(\text{or}\) c)
  – **Distributivity:**
    • a \(\text{and}\) (b \(\text{or}\) c) = (a \(\text{and}\) b) \(\text{or}\) (a \(\text{and}\) c)
    • a \(\text{or}\) (b \(\text{and}\) c) = (a \(\text{or}\) b) \(\text{and}\) (a \(\text{or}\) c)
  – **Double negation:**
    • not (not a) = a
  – **De Morgan’s law:**
    • not (a \(\text{and}\) b) = (not a) \(\text{or}\) (not b)
    • not (a \(\text{or}\) b) = (not a) \(\text{and}\) (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)

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

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:

\[
\begin{align*}
\text{==} & \quad \text{!=} & \quad > & \quad >= & \quad < & \quad <= \\
\end{align*}
\]

• 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:
  
  \texttt{type \ variable\_name;}

• Examples
  
  \texttt{int population;}
  \texttt{double distance;}
  \texttt{string my\_name;}

• Several variables can be declared together:
  
  \texttt{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:
    
    \[
    a + 3\times(i - 1) \rightarrow \text{int}
    \]
    \[
    \sqrt{x} \times \log(4\times n) \rightarrow \text{double}
    \]
    \[
    (i - 3) \leq x \rightarrow \text{bool}
    \]
    \[
    (a \neq b) \text{ and } (s \leq "abc") \rightarrow \text{bool}
    \]
Expressions

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

```c
int a, b, n;
...
(a <= b) + n  // (Incorrect expression: semantic error)
```

Cannot add bool to int
Expressions

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>+, -, not</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>* / %</td>
</tr>
<tr>
<td>Additive</td>
<td>+ -</td>
</tr>
<tr>
<td>Relational (inequalities)</td>
<td>&gt; &gt;= &lt; &lt;=</td>
</tr>
<tr>
<td>Relational (equalities)</td>
<td>== !=</td>
</tr>
<tr>
<td>Conjunction</td>
<td>and</td>
</tr>
<tr>
<td>Disjunction</td>
<td>or</td>
</tr>
</tbody>
</table>

• 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:

```c
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:

```plaintext
int i = 5;
char a = 'B';
double x = 1.5;
i = i + x;
if (i) x = 5*a;
```
In C++, there would be no errors in this fragment of code:

```c++
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
  \[
  \text{char}(i), \quad \text{int}('a'), \quad \text{double}(i)
  \]

• Never use statements that depend on a particular encoding:
  – Wrong: \( c == 65, \quad c == \text{char}(65), \quad \text{int}(c) == 65 \)
  – Correct: \( c == 'A' \)
Type conversion

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

• Be careful:

```cpp
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

```cpp
{
    // 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
}
```