C++ Basics

Variables, Identifiers, Assignments, Input/Output

Variable

A variable can hold a number or a data of other types, it always holds something. A variable has a name and the data held in variable is called value. Variables are implemented as memory locations and assigned certain memory addresses. The exact address depends on the computer and compiler.

We think as though the memory locations are actually labeled with variable names.

Identifiers

- Name of a variable (or any other item you define in a program) is called identifier.
- An identifier must start with a letter or underscore symbol (_), the rest of the characters should be letters, digits or underscores.
- The following are valid identifiers:
  - `x`, `x1`, `_abc`, `sum`, `State`, `average`
- The following are not legal identifiers. Why?
  - `13`, `3X`, `%change`, `data-1`, `my.identifier`, `a(3)`
- C++ is case sensitive: `MyVar` and `myvar` are different identifiers.

What are good identifiers?

- Careful selection of identifiers makes your program clearer.
- Identifiers should be:
  - Short enough to be reasonable to type (single word is norm).
  - Standard abbreviations are fine (but only standard abbreviations).
  - Long enough to be understandable.
  - Two styles of identifiers:
    - C-style - terse, use abbreviations and underscores to separate the words, never use capital letters for variables.
    - Pascal-style - if multiple words: capitalize, don’t use underscores.
- Pick style and use consistently. I use C-style.

Examples

<table>
<thead>
<tr>
<th>Pascal-style</th>
<th>C-style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>min</td>
</tr>
<tr>
<td>Temperature</td>
<td>temperature</td>
</tr>
<tr>
<td>CameraAngle</td>
<td>camera_angle</td>
</tr>
<tr>
<td>CurrentNumberPoints</td>
<td>cur_point_nbr</td>
</tr>
</tbody>
</table>

Keywords

- Keywords are identifiers reserved as part of the language.
- They cannot be used by the programmer to name things.
- They consist of lowercase letters only.
- They have special meaning to the compiler.

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto, double, int, return</td>
</tr>
<tr>
<td>break, delete, for, continue</td>
</tr>
<tr>
<td>case, catch, char, class</td>
</tr>
<tr>
<td>const, continue, default</td>
</tr>
<tr>
<td>delete, do, else, enum, if</td>
</tr>
<tr>
<td>add, break, char, class</td>
</tr>
<tr>
<td>const, continue, default</td>
</tr>
<tr>
<td>delete, do, else, enum, if</td>
</tr>
<tr>
<td>add, break, char, class</td>
</tr>
</tbody>
</table>

Keywords (cont.)

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm, do, if, return, typedef</td>
</tr>
<tr>
<td>auto, double, int, short</td>
</tr>
<tr>
<td>bool, dynamic_cast, long</td>
</tr>
<tr>
<td>case, else, mutable, static</td>
</tr>
<tr>
<td>catch, enum, namespace, static_cast</td>
</tr>
<tr>
<td>char, char, class, const</td>
</tr>
<tr>
<td>case, catch, class, const</td>
</tr>
<tr>
<td>char, char, class, const</td>
</tr>
<tr>
<td>case, catch, class, const</td>
</tr>
<tr>
<td>char, char, class, const</td>
</tr>
</tbody>
</table>
Variable Declarations

- Every variable in C++ program needs to be declared.
- Declaration tells the compiler (and eventually the computer) what kind of data is going to be stored in the variable.
- The kind of data stored in variable is called its type.
- An object definition specifies:
  - Type
  - Name
- A common definition form: Type Id, Id, ..., Id
- Two commonly used types are:
  - int - whole positive or negative numbers: 1, 2, -1, 0, -288, etc.
  - double - positive or negative numbers with fractional part: 1.75, -0.55
- Example declarations:
  ```
  int number_of_bars;
  double weight, total_weight;
  ```

Where to declare

- The variables should be declared as close to the place where they are used as possible.
- If the variable will be used in several unrelated locations, declare it at the beginning of the program:
  ```
  int main() {
    // right here
  }
  ```
- Note that variable contains a value after it is declared. The value is usually arbitrary.

Assignment

- Assignment statement is an order to the computer to set the value of the variable on the left hand side of the equation to what is written on the right hand side.
- It looks like a math equation but it is not!
- Example:
  ```
  number_of_bars = 37;
  total_weight = one_weight;
  total_weight = one_weight * number_of_bars;
  number_of_bars = number_of_bars + 3;
  ```

Output

- To do input/output you have to insert `#include <iostream>` at the beginning of your program.
- C++ uses streams for input or output.
- Stream - is a sequence of data to be read (input stream) or a sequence of data generated by the program to be output (output stream).
- Variable values as well as strings of text can be output to the screen using `cout` (console output):
  ```
  cout << number_of_bars;
  cout << " candy bars\n";
  ```
- The `\n` at the end of the string serves the same purpose as `endl`.
- Arithmetic expressions can be used with the output statement:
  ```
  cout << "The total cost is $" << (price + tax);
  ```
- Escape sequence starts with a backslash (`\`). It is actually just one special character.
- Useful escape sequences:
  ```
  \n
  - new-line
  \t
  - horizontal tab
  \a
  - alert
  \v
  - backslash
  \n
  - double quote
  \";
  ```
- What does this statement print?
  ```
  cout << "\" this is a \c very cryptic \" statement \"\n";
  ```

More Output

- The data can be stringed together:
  ```
  cout << number_of_bars << " candy bars\n";
  ```
- Symbol `\n` at the end of the string serves the same purpose as `endl`.
- Arithmetic expressions can be used with the output statement:
  ```
  cout << "The total cost is $" << (price + tax);
  ```

Escape Sequences

- Certain sequences of symbols make special meaning to the computer. They are called escape sequences.
- Escape sequence starts with a backslash (`\`). It is actually just one special character.
- Useful escape sequences:
  ```
  \n
  - new-line
  \t
  - horizontal tab
  \a
  - alert
  \v
  - backslash
  \n
  - double quote
  \";
  ```
- What does this statement print?
  ```
  cout << "\" this is a \c very cryptic \" statement \"\n";
  ```
<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Name</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>newline</td>
<td>NL</td>
<td>\n</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>HT</td>
<td>\t</td>
</tr>
<tr>
<td>backspace</td>
<td>BS</td>
<td>\b</td>
</tr>
<tr>
<td>form feed</td>
<td>FF</td>
<td>\f</td>
</tr>
<tr>
<td>alert or bell</td>
<td>BEL</td>
<td>\a</td>
</tr>
<tr>
<td>carriage return</td>
<td>CR</td>
<td>\r</td>
</tr>
<tr>
<td>vertical tab</td>
<td>VT</td>
<td>\v</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>single quote</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
<td>*</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

**Input**

- `cin:` (stands for Console Input) is used to fill the values of variables with the input from the user of the program.
- when the program reaches the input statement it just pauses until the user types something and presses <return>.
- therefore it is beneficial to precede the input statement with some explanatory output:
  - `cout << "Enter the number of candy bars\n";
  - `cout << "and weight in ounces,\n";
  - `cin >> number_of_bars >> one_weight;
- `>>` is called extraction operator.
- note how input statements (similar to output statements) can be stacked.
- input tokens (numbers in our example) should be separated by (any amount of) whitespace (spaces, tabs, newlines).
- the values typed are inserted into variables when <return> is pressed, if more values needed - program waits, if extra typed - they are used in next input statements if needed.