C++ Syntax

Novelties

New type

bool a=true, b=false

New headers format

#include <iostream>

using namespace std;

Input/Output

Output

Cout << “Enter a number(-1 = quit):”;

Input

Cin >> x;

Example

input-ok1.cpp

```cpp
#include <iostream>
using namespace std;
void TestInput(){
    int x = 0;
  do {
    cout << "Enter a number (-1 to quit): ";
    if(!(cin >> x)) {
      cout << "The input stream broke!" << endl;
      x = -1;
    }
    if(x != -1) cout << x << " was entered" << endl;
  } while(x != -1);
  cout << "Exit" << endl;
}
int main() {TestInput(); return 0;}
```

Example

input-ok2.cpp

```cpp
#include <iostream>
using namespace std;
void TestInput(){
  int x=0;
  do {
    cout << "Enter a number (-1 to quit): ";
    cin >> x;
    cin.clear();
    cin.ignore(10000,'\n');
    if(x != -1) cout << x << " was entered" << endl;
  }  while(x != -1);
  cout << "Exit" << endl;
}
int main() {TestInput(); return 0;}
```

Clean the wrong state of the stream

Clean the buffer

File I/O

Requires header: #include <fstream>

- Open file for reading: ifstream in(“file.txt”)
- Read from a file: in used in the same way as cin
- Open a file for writing: ofsream out(“file.txt”)
- Write in a file: out used in the same way as cout
- Read from a file, line by line: getline(in,s)

Example

fio.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
void FileIO() {
  string s;
  ifstream a("1.txt"); ofstream b("2.txt");
  while(getline(a,s)) {b << s << endl;  cout << s;}
}
int main () {FileIO();return 0;}
```

Object and Class

Object

An object has two main components:

- The data it contains, what is known to the object, its attributes or data members
- The behavior it has, what can be done by the object, its methods or function members.

Class and Instance

Class:

- Defines the family, type or nature of an object
- Equivalent of the type in “traditional programming”

Instance:

- Realization of an object from a given class
- Equivalent of a variable in “traditional programming”

A Note on Visibility

Private or Public:

- Private members can only be accessed by member functions within the class
- Users can only access public members

Benefits:

- Internal implementation can be easily adjusted without affecting the user code
- Accessing private attribute is forbidden: more secure

Example

circle-v0.h

```
1  class Circle {
2  /* user methods (and attributes)*/
3    public:
4      void move(float dx, float dy);
5      void zoom(float scale);
6      float area();
7  /* implementation attributes (and methods) */
8    private:
9      float x, y;
10     float r;
11 };
```

Example

Calling a method on an object: instance.method

main-v0.cpp

```cpp
#include <iostream>
#include "circle_v0.h"
using namespace std;
int main () {
  float s1, s2;
  Circle circ1, circ2;
  circ1.move(12,0);
  s1=circ1.area(); s2=circ2.area();
  cout << "area: " << s1 << endl;
  cout << "area: " << s2 << endl;
  circ1.zoom(2.5);
  s1=circ1.area();
  cout << "area: " << s1 << endl;
}
```

Implementation

Syntax: classname::methodname

circle-v0.cpp

```cpp
#include "circle_v0.h"
static const float PI=3.1415926535;
void Circle::move(float dx, float dy) {
  x += dx;
  y += dy;
}
void Circle::zoom(float scale) {
  r *= scale;
}
float Circle::area() {
  return PI * r * r;
}
```

Constructor and Destructor

Automatic construction and destruction of objects:

- Object not initialized by default (same as int i)
- Constructor: method that initializes an instance of an object
- Used for a proper default initialization
- Definition: no type, name must be classname
- Important note: can have more than one constructor
- Destructor: called just before the object is destroyed
- Used for clean up
- Definition: no type, name must be ~classname

Example

circle-v1.h

```
1  class Circle {
2  /* user methods (and attributes)*/
3    public:
4      Circle();
5      Circle(float r);
6      ~Circle();
7      void move(float dx, float dy);
8      void zoom(float scale);
9      float area();
10 /* implementation attributes (and methods) */
11   private:
12     float x, y;
13     float r;
14 };
```

Example

circle-v1.cpp

```cpp
#include "circle_v1.h"
static const float PI=3.1415926535;
Circle::Circle() {
  x=y=0.0; r=1.0;
}
Circle::Circle(float radius) {
  x=y=0.0; r=radius;
}
Circle::~Circle() {}
void Circle::move(float dx, float dy) {
  x += dx; y += dy;
}
void Circle::zoom(float scale) {
  r *= scale;
}
float Circle::area() {
  return PI * r * r;
}
```

Example

main-v1.cpp

```cpp
#include <iostream>
#include "circle_v1.h"
using namespace std;
int main () {
  float s1, s2;
  Circle circ1, circ2((float)3.1);
  circ1.move(12,0);
  s1=circ1.area(); s2=circ2.area();
  cout << "area: " << s1 << endl;
  cout << "area: " << s2 << endl;
  circ1.zoom(2.5);
// cout << circ1.r <<endl;
  s1=circ1.area();
  cout << "area: " << s1 << endl;
}
```

Overloading

Better definitions:

- Two constructor defined: circle() and circle(float)
- Proper one automatically selected

Another strategy is to set a default value in the specification.

```
1  Circle(float radius=1.0);
```