C++ IO Streams

Eden Burton <ronald.burton@senecacollege.ca>
github repository:
(https://github.com/Seneca-OOP244/SCD-Notes)

25/09/2015

Streams

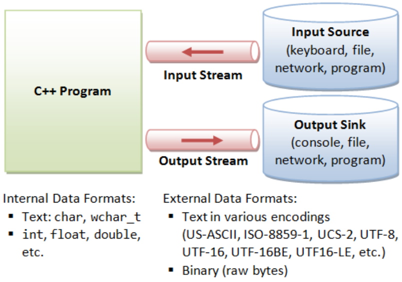
```
#include <stdio.h>

int main(...) {
   int grade; double avg;
   ...
   printf ("Grade:␣%d␣Avg:␣%f\n", grade, avg)
}
```

```
#include <iostream>
using namespace std;
int main(...) {
   int grade; double avg;
   ...
   cout <<  "Grade:␣" <<  grade
        << "␣Avg:␣" <<  avg  <<  endl;
}
```

Streams

– a sequence of characters


C++ Program
Input Stream
Input Source (keyboard, file, network, program)
Output Stream
Output Sink (console, file, network, program)
Internal Data Formats:
▪ Text: char, wchar_t
▪ int, float, double, etc.
External Data Formats:
▪ Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
▪ Binary (raw bytes)

– a input object, stream characters → types in system memory

– a output object, types in system memory → stream characters

1

[1] https://www3.ntu.edu.sg/home/ehchua/programming/cpp/images/IOstreams.png

Streams - Output Streams

- `ostream` type
- takes data from our program, put it into stream characters
- standard objects
  - cout, stream to stdout device
  - cerr, stream to stderr device

```
    cout << identifier;
    cout << identifier [ << identifier];
```

Streams - Output Stream Format

The predefined in and cout identifiers are actually objects

- can be operated on via its member functions

```
int main(...) {
    ...
    cout <<  "1234" <<  endl;
    cout.width(10);
    cout << 12 << endl;  }
```

- or via manipulators

```
int main(...) {
    ...
    cout <<  setw(10) <<  12 << endl;}
```

Streams - Input Streams

- `istream` type
- takes stream characters, stuffs it into program
- standard objects
  - cin, stream from std device

```
    cin >> identifier;
```

- skips leading whitespace
- whitespace as delimiter for numeric and string data types
- adds null byte right after string in memory

Dynamic Memory - A Quick Peek

- static memory, memory allocated for application by o/s
- dynamic memory, memory requested by application
  - memory is managed by the developer explictly

```
void createAStudent() {

    Student staticHarry;
    Student * dynamicHarry = new Student();
    ...
    staticHarry.display();
    dynamicHarry->display();
    ...
    delete dynamicHarry;
    dynamicHarry = nullptr;
    ...
}
```

Passing Arguments To Functions

type identifier(type[, ...], type = value)

- pass-by-value, argument is a copy of the variable
- pass-by-address, argument is a pointer to variable
- pass-by-reference, argument is an alias of the variable

```
// pass-by-value
void swap ( char a, char b );

// pass-by-address
void swap ( char *a, char *b );

// pass-by-reference
void swap ( char &a, char &b );
```

Member Functions

"...recall that a structure (or class) is composed of data and member functions used to modify it..."

```
class Box {
   double length;
   double breadth;
   double height;
   double volume;

   double getVolume();
   double setHeight(double h);

 };
```

More on Member Functions

Member Function Classifications

- accessor methods, answer question about object state without modifying it
- mutator methods, they modify object state
- special, create, assign and destroy objects

```
// declaration usually put in the header file
class Box {
   double length;
   double breadth;
   double height;
   double volume;

   double getVolume() const;    // accessor
   double setHeight(double h);  // mutator
};
```

Privacy

Accessibility Labels

- private: - prevents external access by clients
- public: - allows client access

```
struct Student {
     int no;
     char grade[14];
     void display() const;
 };

 class Student {
     int no;
     char grade[14];
     void display() const;
 };
```

- `struct` makes members public by default
- **`class`** makes members private by default

More Privacy

– labels set viability until another label changes it

```
struct Student {
  private:
    int no;
    char grade[14];
  public:
    void display() const;
};
...
int main() {
   Student st;
   st.display()   // ok
   st.no;      //error, cannot access
}
```