CSCI 2270: Data Structures

Lecture 03: C++ Review: File I/O, Arrays, and Structures

Ashutosh Trivedi

Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

C++: A quick review (contd.)

Streams: basic input-output mechanism

- *Streams* are the sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe).

Streams: basic input-output mechanism

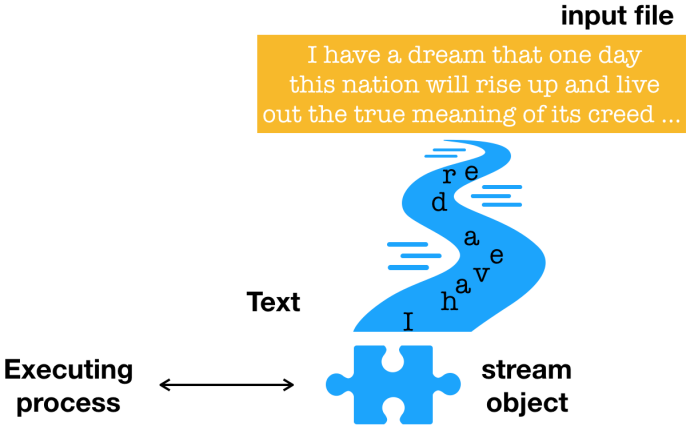
- *Streams* are the sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe).
- In *input operations*, data bytes flow from an input source (keyboard, file, network, strings, and other programs) to the program, and

Streams: basic input-output mechanism

- *Streams* are the sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe).
- In *input operations*, data bytes flow from an input source (keyboard, file, network, strings, and other programs) to the program, and
- in output operations, data bytes flow from the program to an output sink (such as console, file, network or another program).
- Streams act as an abstract interface between the program and the actual IO devices in such a way that frees a programmer from hardware concerns.

Streams: basic input-output mechanism

- *Streams* are the sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe).



Streams: basic input-output mechanism

- We have seen how to use the standard I/O stream by including the following pre-compiler directive:

```
#include <iostream>
```

Streams: basic input-output mechanism

- We have seen how to use the standard I/O stream by including the following pre-compiler directive:

```
#include <iostream>
```

- In order to output to screen, we merely use a statement like,

```
std::cout << " X = "<< X;
```

Streams: basic input-output mechanism

- We have seen how to use the standard I/O stream by including the following pre-compiler directive:

```
#include <iostream>
```

- In order to output to screen, we merely use a statement like,

```
std::cout << " X = "<< X;
```

- The “insertion operator” (<<) points in the direction of data flow.

Streams: basic input-output mechanism

- We have seen how to use the standard I/O stream by including the following pre-compiler directive:

```
#include <iostream>
```

- In order to output to screen, we merely use a statement like,

```
std::cout << " X = "<< X;
```

- The "insertion operator" (<<) points in the direction of data flow.
- Similarly, input stream reads from the keyboard into a variable.
- The extraction operator (>>) is "smart enough" to consider the type of the target variable when it determines how much to read from the input stream.

Input from the keyboard

```cpp
// program14.cpp
#include<iostream>
#include<string>
int main()
{
  std::cout << "Please input a string, an integer, a character, and a float (space separated): \n";
  std::string w;
  int x;
  char y;
  float z;
  std::cin >> w;
  std::cin >> x;
  std::cin >> y;
  std::cin >> z;
  std::cout << "w= " << w << " x= " <<  x << " y=" << y  <<  " z= " << z << std::endl;
  return 0;
}
```

File Output

```cpp
// program15.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main () {
  ofstream myfile;
  myfile.open("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

File Output: Parameters (Append)

```cpp
// program16.cpp
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile("example.txt", ios::binary | ios::app | ios::out);
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

Notice:

- The "constructor" instead of `open()`.
- Parameters passed while opening the file.

File Input

```cpp
// program17.cpp
#include <iostream>
#include <fstream>
int main (int argc, char *argv[]) {
  std::ifstream fin("addresses.txt");
  if (fin.is_open()) {
    std::cout << "File is open as fin stream\n";
    char c;
    fin >>  c;
    std::cout << "first char is " << c << " \n";
  }
  else std::cerr << "File addresses.txt not found!";
  fin.close(); // Don't forget to close!
  return 0;
}
```

File Input: Eat it line by line!

```cpp
// program18.cpp
#include <iostream>
#include <fstream>
#include <string>
int main (int argc, char *argv[]) {
  std::ifstream fin("addresses.txt");
  if (fin.is_open()) {
    std::string line;
    while (getline(fin, line)) {
      std::size_t found = line.find("TX");
      if (found!=std::string::npos) {
  std::cout << line << std::endl;
      }
    }
  }
  else std::cerr << "File addresses.txt not found!";
  fin.close(); // Don't forget to close!
  return 0;
}
```

String Streams: input and output to strings

```
    while (getline(fin, line)) {
      std::stringstream sin(line);
      std::string id, name, phone, email, street, zip, city, state, lat, lon;
      getline(sin, id, ',');
      getline(sin, name, ',');
      getline(sin, phone, ',');
      getline(sin, email, ',');
      getline(sin, street, ',');
      getline(sin, city, ',');
      getline(sin, state, ',');
      getline(sin, zip, ',');
      getline(sin, lat, ',');
      getline(sin, lon, ' ');
      std:: cout << name  << "lives in " << state << std::endl;
    }
  }
```

Pointer Variables

```
1  // program7.cpp
2  #include<iostream>
3  int main(int argc, char* argv[])
4  {
5    char ch= 'a';
6    char *cp; // cp is a pointer variable
7    cp = &ch; // cp points to the address of the ch
8    std::cout << "Size of a pointer to char: ";
9    std::cout << sizeof(char *) << std::endl;
10   std::cout << "Address of ch is = " << (void *) cp;
11   return 0;
12 }
```

1. What are the sizes of pointers to different types of objects?

Pointer Variables

```cpp
// program7.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  char ch= 'a';
  char *cp; // cp is a pointer variable
  cp = &ch; // cp points to the address of the ch
  std::cout << "Size of a pointer to char: ";
  std::cout << sizeof(char *) << std::endl;
  std::cout << "Address of ch is = " << (void *) cp;
  return 0;
}
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.

Pointer Variables

```
1  // program7.cpp
2  #include<iostream>
3  int main(int argc, char* argv[])
4  {
5    char ch= 'a';
6    char *cp; // cp is a pointer variable
7    cp = &ch; // cp points to the address of the ch
8    std::cout << "Size of a pointer to char: ";
9    std::cout << sizeof(char *) << std::endl;
10   std::cout << "Address of ch is = " << (void *) cp;
11   return 0;
12 }
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.
3. A pointer to variable of type T is:
   - 3.1 `T* p` or `T *p`
   - 3.2 bad practice: `int *p, q, r`.

Pointer Variables

```
1  // program7.cpp
2  #include<iostream>
3  int main(int argc, char* argv[])
4  {
5    char ch= 'a';
6    char *cp; // cp is a pointer variable
7    cp = &ch; // cp points to the address of the ch
8    std::cout << "Size of a pointer to char: ";
9    std::cout << sizeof(char *) << std::endl;
10   std::cout << "Address of ch is = " << (void *) cp;
11   return 0;
12 }
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.
3. A pointer to variable of type T is:
   - 3.1 `T* p` or `T *p`
   - 3.2 bad practice: `int *p, q, r`.
4. A pointer variable equal to 0 means it does not refer to an object. Use of **NULL** discouraged!

Arrays

- An array is a collection of elements of the same type.
- Given a variable of type T, and array of type $T[N]$ holds an array of N elements, each of type N.
- Each element of the array can be referenced by its index that is a number of 0 to $N - 1$.

Arrays

- An array is a collection of elements of the same type.
- Given a variable of type T, and array of type $T[N]$ holds an array of N elements, each of type N.
- Each element of the array can be referenced by its index that is a number of 0 to $N-1$.

```cpp
// program8.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  int ia[3]; //Array of 3 ints with garbage values
  std::cout << ia[1] << std::endl;
  float fa[] = {1, 2, 3}; //Array of 3 floats initialzed: size automatically computed
  std::cout << fa[2] << std::endl; // Read different values
  return 0;
}
```

Arrays (Statically Declared Arrays)

```cpp
// program8.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  int ia[3]; //Array of 3 ints with garbage values
  std::cout << ia[1] << std::endl;
  float fa[] = {1, 2, 3}; //Array of 3 floats initialzed: size automatically computed
  std::cout << fa[2] << std::endl; // Read different values
  return 0;
}
```

1. Static Array storage is contiguous.
2. Array bound must be a constant expression. If you need variable bounds, use a `vector`.
3. What happens when initialization and array size mismatch?
4. Multi-dimensional arrays (contiguous in row-order fashion!).

Structures (Our first data-structure!)

- A structure is useful for storing an aggregation of elements.
- Unlike an array, the elements of a structure may be of different types.
- Each element of field is referred by a given name.

Structures (Our first data-structure!)

- A structure is useful for storing an aggregation of elements.
- Unlike an array, the elements of a structure may be of different types.
- Each element of field is referred by a given name.

```
// program11.cpp
#include<iostream>
int main(int argc, char* argv[])  {
  struct address {
    long id; // unique ID: 1614011140000
    std::string name;  // Name: Ashutosh Trivedi
    std::string phone; //Phone number: (720) 707-9663
    std::string email; //ashutosh.trivedi@gmail.com
    std::string street; //4141 Spruce Street
    std::string city; // Philadelphia
    std::string state; //PA
    int zip;  // 19104
    float lat; //39.948610
    float lon; //-75.177830
  };
```

Structures (Our first data-structure!)

```cpp
// program11.cpp
#include<iostream>
int main(int argc, char* argv[])  {
  struct address {
    long id; // unique ID: 1614011140000
    std::string name;  // Name: Ashutosh Trivedi
    std::string phone; //Phone number: (720) 707-9663
    std::string email; //ashutosh.trivedi@gmail.com
    std::string street; //4141 Spruce Street
    std::string city; // Philadelphia
    std::string state; //PA
    int zip;  // 19104
    float lat; //39.948610
    float lon; //-75.177830
  };
  address myadress = {1614011140000, "Ashutosh Trivedi", "(720) 707-9663", "ashutosh.trivedi@gmail.com", "4141 Spruce Street","Philadelphia
        ","PA",19104,39.948610,-75.177830};
  std::cout << myadress.name << " lives in " << myadress.state << std::endl;
  return 0;
```

Structures (Constructors)

```cpp
// program20.cpp
#include<iostream>
int main(int argc, char* argv[])  {
  struct Address {
    long id; // unique ID: 16140111140000
    std::string name;  // Name: Ashutosh Trivedi
    std::string phone; //Phone number: (720) 707-9663
    std::string email; //ashutosh.trivedi@gmail.com
    std::string street; //4141 Spruce Street
    std::string city; // Philadelphia
    std::string state; //PA
    int zip;  // 19104
    float lat; //39.948610
    float lon; //-75.177830
    Address() {};
    Address(long _id, std::string _name, std::string _phone, std::string _email, std::string _street, std::string _city, std::string _state
          , int _zip,float _lat, float _lon) {
      id = _id;
      name = _name;
      phone = _phone;
      email = _email;
      street = _street;
      city = _city;
      state = _state;
      zip = _zip;
      lat = _lat;
      lon = _lon;
    }
```

Structures (Member functions)

```
1  struct Address {
2    long id; // unique ID: 1614011140000
3    std::string name;  // Name: Ashutosh Trivedi
4    std::string phone; //Phone number: (720) 707-9663
5    std::string email; //ashutosh.trivedi@gmail.com
6    std::string street; //4141 Spruce Street
7    std::string city; // Philadelphia
8    std::string state; //PA
9    int zip;  // 19104
10   float lat; //39.948610
11   float lon; //-75.177830
12   Address() {};
13   Address(long _id, std::string _name, std::string _phone, std::string _email, std::string _street, std::string _city, std::string _state
           , int _zip,float _lat, float _lon) {
14     id = _id;
15     name = _name;
16     phone = _phone;
17     email = _email;
18     street = _street;
19     city = _city;
20     state = _state;
21     zip = _zip;
22     lat = _lat;
23     lon = _lon;
24   }
25   void prettyPrint() {
26     std:: cout << name << std::endl;
27     std::cout << "    Id: \n        " << id << std::endl;
28     std::cout << "    Ph: \n        " << phone << std::endl;
29     std::cout << "    E-mail: \n        " << email << std::endl;
30     std::cout << "    Address: "<< std::endl;
31     std::cout << "        "<< street << ", " << city << ", " << state << "-" << zip << std::endl;
32     std::cout << "        Location: ("<< lat << ", " << lon << ")" << std::endl;
33     std::cout<< "_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/" << std::endl;
```

All together now!

- Read addresses.txt from the command-line.
- Define a structure corresponding to each record.
- Declare an array of such structures.
- Store the contents of the file into the structure.
- Pretty print the whole database.

Final Program - 1

```cpp
// final.cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
struct Address {
  long id; // unique ID: 1614011140000
  std::string name;  // Name: Ashutosh Trivedi
  std::string phone; //Phone number: (720) 707-9663
  std::string email; //ashutosh.trivedi@gmail.com
  std::string street; //4141 Spruce Street
  std::string city; // Philadelphia
  std::string state; //PA
  int zip;  // 19104
  float lat; //39.948610
  float lon; //-75.177830
  Address() {};
  Address(long _id, std::string _name, std::string _phone, std::string _email, std::string _street, std::
        string _city, std::string _state, int _zip,float _lat, float _lon) {
    id = _id;
    name = _name;
    phone = _phone;
    email = _email;
    street = _street;
    city = _city;
    state = _state;
    zip = _zip;
    lat = _lat;
    lon = _lon;
  }
```

Final Program - 2

```
void fill(std::string _id, std::string _name, std::string _phone, std::string _email, std::string _street,
      std::string _city, std::string _state, std::string _zip, std::string _lat, std::string _lon) {
  id = std::stol(_id);
  name = _name;
  phone = _phone;
  email = _email;
  street = _street;
  city = _city;
  state = _state;
  zip = std::stoi(_zip);
  lat = std::stof(_lat);
  lon = std::stof(_lon);
}
void prettyPrint() {
  std:: cout << name << std::endl;
  std::cout << "    Unique Identity Number: \n        " << id << std::endl;
  std::cout << "    Phone number: \n        +1 " << phone << std::endl;
  std::cout << "    E-mail: \n        " << email << std::endl;
  std::cout << "    Address: "<< std::endl;
  std::cout << "        "<< street << ", " << city << ", " << state << "-" << zip << std::endl;
  std::cout << "    Location:\n        ("<< lat << ", " << lon << ")" << std::endl;
  std::cout<< "_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/" <<
       std::endl;
}
};

int main (int argc, char *argv[]) {
  if (argc != 2) std::cerr << "Error: incorrect number of arguments \n";
  else {
    std::ifstream fin(argv[1]);
```

Final Program - 3

```
    if (fin.is_open()) {
      Address addressDB[100]; // Address database (array of structures)
      int size = 0;
      std::string line;
      while (getline(fin, line)) {
  std::stringstream sin(line);
  std::string id, name, phone, email, street, zip, city, state, lat, lon;
  getline(sin, id, ',');
  getline(sin, name, ',');
  getline(sin, phone, ',');
  getline(sin, email, ',');
  getline(sin, street, ',');
  getline(sin, city, ',');
  getline(sin, state, ',');
  getline(sin, zip, ',');
  getline(sin, lat, ',');
  getline(sin, lon, ' ');

  addressDB[size].fill(id, name, phone, email, street, city, state, zip, lat, lon);
  size++;
      }
      for (int i=0; i< size; i++) addressDB[i].prettyPrint();
  }
  else std::cerr << "File addresses.txt not found!";
  fin.close(); // Don't forget to close!
  return 0;
  }
}
```