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).
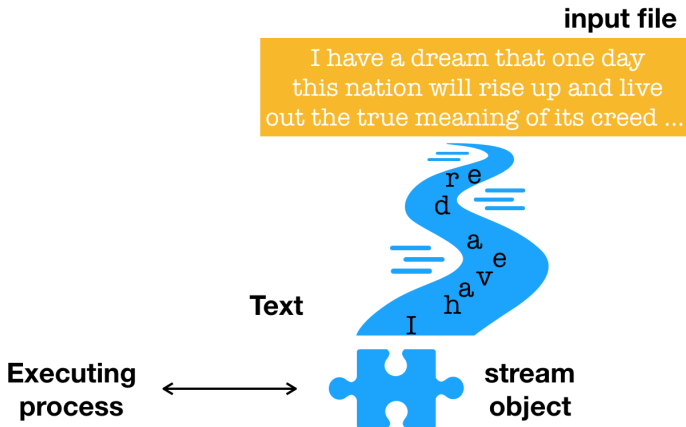
Streams: basic input-output mechanism

– *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).

input file

I have a dream that one day this nation will rise up and live out the true meaning of its creed ...

Text

Executing process ⟷ **stream object**

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
1   // program14.cpp
2   #include<iostream>
3   #include<string>
4   int main()
5   {
6     std::cout << "Please input a string, an integer, a character, and a float (space separated): \n";
7     std::string w;
8     int x;
9     char y;
10    float z;
11    std::cin >> w;
12    std::cin >> x;
13    std::cin >> y;
14    std::cin >> z;
15    std::cout << "w= " << w << " x= " <<  x << " y=" << y  <<  " z= " << z << std::endl;
16    return 0;
17  }
```

File Output

```
1   // program15.cpp
2   #include <iostream>
3   #include <fstream>
4   using namespace std;
5   int main () {
6     ofstream myfile;
7     myfile.open("example.txt");
8     myfile << "Writing this to a file.\n";
9     myfile.close();
10    return 0;
11  }
```

File Output: Parameters (Append)

```
1  // program16.cpp
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main () {
7    ofstream myfile("example.txt", ios::binary | ios::app | ios::out);
8    myfile << "Writing this to a file.\n";
9    myfile.close();
10   return 0;
11 }
```

Notice:

– The "constructor" instead of `open()`.

– Parameters passed while opening the file.

File Input

```
1   // program17.cpp
2   #include <iostream>
3   #include <fstream>
4   int main (int argc, char *argv[]) {
5     std::ifstream fin("addresses.txt");
6     if (fin.is_open()) {
7       std::cout << "File is open as fin stream\n";
8       char c;
9       fin >> c;
10      std::cout << "first char is " << c << " \n";
11    }
12    else std::cerr << "File addresses.txt not found!";
13    fin.close(); // Don't forget to close!
14    return 0;
15  }
```

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

```
1      while (getline(fin, line)) {
2        std::stringstream sin(line);
3        std::string id, name, phone, email, street, zip, city, state, lat, lon;
4        getline(sin, id, ',');
5        getline(sin, name, ',');
6        getline(sin, phone, ',');
7        getline(sin, email, ',');
8        getline(sin, street, ',');
9        getline(sin, city, ',');
10       getline(sin, state, ',');
11       getline(sin, zip, ',');
12       getline(sin, lat, ',');
13       getline(sin, lon, ' ');
14       std:: cout << name  << "lives in " << state << std::endl;
15     }
16   }
```

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

```
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.

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.
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.

```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
  };
```

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
1    // program20.cpp
2    #include<iostream>
3    int main(int argc, char* argv[]) {
4      struct Address {
5        long id; // unique ID: 1614011140000
6        std::string name;  // Name: Ashutosh Trivedi
7        std::string phone; //Phone number: (720) 707-9663
8        std::string email; //ashutosh.trivedi@gmail.com
9        std::string street; //4141 Spruce Street
10       std::string city; // Philadelphia
11       std::string state; //PA
12       int zip;  // 19104
13       float lat; //39.948610
14       float lon; //-75.177830
15       Address() {};
16       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) {
17         id = _id;
18         name = _name;
19         phone = _phone;
20         email = _email;
21         street = _street;
22         city = _city;
23         state = _state;
24         zip = _zip;
25         lat = _lat;
26         lon = _lon;
27       }
```

Structures (Member functions)

```cpp
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
1   // final.cpp
2   #include <iostream>
3   #include <fstream>
4   #include <sstream>
5   #include <string>
6   struct Address {
7     long id; // unique ID: 1614011140000
8     std::string name;  // Name: Ashutosh Trivedi
9     std::string phone; //Phone number: (720) 707-9663
10    std::string email; //ashutosh.trivedi@gmail.com
11    std::string street; //4141 Spruce Street
12    std::string city; // Philadelphia
13    std::string state; //PA
14    int zip;  // 19104
15    float lat; //39.948610
16    float lon; //-75.177830
17    Address() {};
18    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) {
19      id = _id;
20      name = _name;
21      phone = _phone;
22      email = _email;
23      street = _street;
24      city = _city;
25      state = _state;
26      zip = _zip;
27      lat = _lat;
28      lon = _lon;
29    }
```

Final Program - 2

```cpp
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

```
1       if (fin.is_open()) {
2          Address addressDB[100]; // Address database (array of structures)
3          int size = 0;
4          std::string line;
5          while (getline(fin, line)) {
6      std::stringstream sin(line);
7      std::string id, name, phone, email, street, zip, city, state, lat, lon;
8      getline(sin, id, ',');
9      getline(sin, name, ',');
10     getline(sin, phone, ',');
11     getline(sin, email, ',');
12     getline(sin, street, ',');
13     getline(sin, city, ',');
14     getline(sin, state, ',');
15     getline(sin, zip, ',');
16     getline(sin, lat, ',');
17     getline(sin, lon, ' ');
18
19     addressDB[size].fill(id, name, phone, email, street, city, state, zip, lat, lon);
20     size++;
21          }
22          for (int i=0; i< size; i++) addressDB[i].prettyPrint();
23     }
24     else std::cerr << "File addresses.txt not found!";
25     fin.close(); // Don't forget to close!
26     return 0;
27     }
28  }
```