CS101 Computer Programming and Utilization

Milind Sohoni

May 19, 2006

The story so far ...

- We have seen various control flows.
- We have seen multi-dimensional arrays and the `char` data type.
- We saw the use of functions and calling methods.

This week...

Structs and File I/O

student.cpp

A struct is a composite data structure. This is useful in representing entities which have many attributes of distinct types. Here is a simple example:

```
struct student
{
  char name[7];
  char roll[9];
  int hostel;
}
```

Variables can be now defined of the type student.

Input a list of students and output those staying in hostels 10 or 11.

student.cpp

A struct is a composite data structure. This is useful in representing entities which have many attributes of distinct types. Here is a simple example:

```
struct student
{
  char name[7];
  char roll[9];
  int hostel;
}
```

Variables can be now defined of the type student.

Input a list of students and output those staying in hostels 10 or 11.

Let us first examine a simple code:

```
int main()
{
    student s;
    cin >> s.name;
    cin >> s.roll;
    cin >> s.hostel;
    printstudent(s);
    return 0;
}
```

student.cpp

A struct is a composite data structure. This is useful in representing entities which have many attributes of distinct types. Here is a simple example:

```
struct student
{
  char name[7];
  char roll[9];
  int hostel;
}
```

Variables can be now defined of the type student.

Input a list of students and output those staying in hostels 10 or 11.

Let us first examine a simple code:

```
int main()
{
    student s;
    cin >> s.name;
    cin >> s.roll;
    cin >> s.hostel;
    printstudent(s);
    return 0;
}
```

```
void printstudent(student s)
{
  printf("%6s  %10s  %4d",
        s.name,s.roll,s.hostel);
  cout << "\n";
}
```

student.cpp

A struct is a composite data structure. This is useful in representing entities which have many attributes of distinct types. Here is a simple example:

```
struct student
{
  char name[7];
  char roll[9];
  int hostel;
}
```

Variables can be now defined of the type `student`.

Input a list of students and output those staying in hostels 10 or 11.

student.cpp

A struct is a composite data structure. This is useful in representing entities which have many attributes of distinct types. Here is a simple example:

```
struct student
{
  char name[7];
  char roll[9];
  int hostel;
}
```

Variables can be now defined of the type student.

Input a list of students and output those staying in hostels 10 or 11.

```
void printstudent(student s)
{
  printf("%6s  %10s  %4d",s.name
  cout << "\n";
}
int main()
{
  int N; student s;
  cin>> N;
  for (int i=1;i<=N;i=i+1)
  {  cin >> s.name;
     cin >> s.roll;
     cin >> s.hostel;
    if ((s.hostel==10) ||(s.host
    printstudent(s);
  }
  return 0;
}
```

Input and Output

```
6
milind  00105003 1
sohoni 00000000 12
ranjit  01010101 10
nishak  10101010 11
ashita  11111111 3
vinita  22222222 4
```

```
ranjit     01010101     10
nishak     10101010     11
```

A small pointer:

- Both `name` and `roll` have one extra character than the length. This is typical of strings.
- The declared length of a character string should be one more than the required. This is because there is a character "\0" which marks the end of a string.

hostel.cpp

Write a program to read in a list of students from the file database.txt and answer queries of the following type:

- h 3 : Print all students in hostel 3.
- x : Exit.

hostel.cpp

Write a program to read in a list of students from the file database.txt and answer queries of the following type:

h 3 : Print all students in hostel 3.

x : Exit.

```
#include<fstream.h>
#include<iostream.h>
struct student
{ char name[7];
  char roll[9];
  int  hostel;
};
void printstudent(student s)
{  printf(s.name,s.roll,s.hostel);
   cout << "\n";
}
```

The structure of our program will be as follows:

- Part I: read in the file `database.txt` and load all the student names in an array of students.
- Part II: Run a while loop reading in options until the option x is observed.
- On option h 3, scan the student list for students in hostel 3.

The line

```
#include<fstream.h>
```

allows us to define:

- ifstream fin; declaring that a file called fin should be prepared for input.
- fin.open("database.txt"); says that this file is database.txt in the outside world, and should be opened for reading.
- fin.close(); closes database.txt is database.txt in the outside world, and should be opened for reading.

```
int main()
{
  student studentlist[100];
  char option;
  int done,i,N,hostelno;
  ifstream fin;
  fin.open("database.txt");
  student s;
  fin>> N;
  for (i=0;i<N;i=i+1)
  {
     fin >> s.name;
     fin >> s.roll;
     fin >> s.hostel;
     studentlist[i]=s;
  }
  cout << "read in database \n";
  fin.close();
```

The line

```
#include<fstream.h>
```

allows us to define:

- Note that with the program the file is called fin, while outside it is known as database.txt. We can replace `fin` by any name of our choice.
- fin is used for reading in data just as we would use `cin`.
- If we needed to output into a file, we would say:

```
ofstream ofilename;
```

The loop is standard programming.

```
int main()
{
  student studentlist[100];
  char option;
  int done,i,N,hostelno;
  ifstream fin;
  fin.open("database.txt");
  student s;
  fin>> N;
  for (i=0;i<N;i=i+1)
  {
     fin >> s.name;
     fin >> s.roll;
     fin >> s.hostel;
     studentlist[i]=s;
  }
  cout << "read in database \n";
  fin.close();
```

```
done=0;
while (done==0)
{
cout << "give your option\n";
cin >> option;
if (option=='x')
{
   done=1;
   cout << "done \n";
}
if (option=='h')
{
   cin >> hostelno;
   cout << ...
   for (i=0;i<N;i=i+1)
   {  s=studentlist[i];
      if (s.hostel==hostelno)
        printstudent(s);
   };
}
} // of while
}
```

By now, all the students have been stored in `studentlist[]`.

```
done=0;
while (done==0)
{
cout << "give your option\n";
cin >> option;
if (option=='x')
{
   done=1;
   cout << "done \n";
}
if (option=='h')
{
   cin >> hostelno;
   cout << ...
   for (i=0;i<N;i=i+1)
   {  s=studentlist[i];
      if (s.hostel==hostelno)
        printstudent(s);
   };
}
} // of while
}
```

By now, all the students have been stored in `studentlist[]`.

Notice the while done==0 loop and the two cases separately.

- On option x, done is set to 1 so that we exit the program.
- On option h, we read in the hostelno. We then scan the studentlist and output the results. We return to the while with done still zero.

Databases

What we have seen is something very close to what is also called a database. In other words, a database is in the form of stored tables which can be accessed through specialized programs.

- The first part of our code built the date base.
- The second part executed quries on the databse.

In most databases, the student list is stored on a central computer. Queries may be executed at different locations.

Write a program to read in a list of students from the file database.txt and answer queries of the following type:

h 3 : Print all students in hostel 3.

x : Exit.

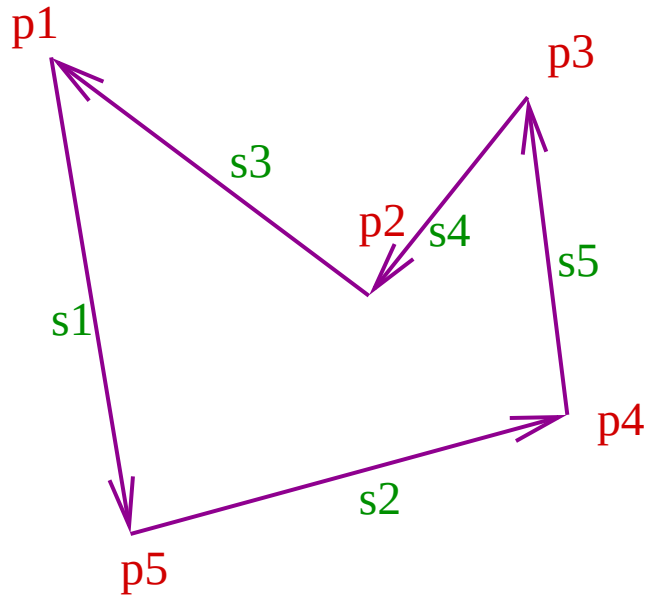
f fn : Route all subsequent output to file named fn.

o : Off the above feature, i.e., output just to cout.

2D line geometry

We define some struct for manipulating 2D objects:

```
struct point
{
  int x,y;
}
struct dsegment
{
   point start,end;
}
struct polygon
{
  dsegment sides[20];
  int N;
}
point p1,p2,p3,p4,p5;
dsegment s1,s2,s3,s4,s5;
polygon pp;
```




```
p1.x=1; p1.y=2;

s1.start=p1;
s1.end=p5;

pp.N=5;
pp.sides[0]=s1;
pp.sides[1]=s2;
...
pp.sides[4]=s3;
```

Here are some required functions:

- `int PointEqual(point p1,p2)` returns 1 if points are equal, else returns 0.
- `int SegEqual (dsegment s1,s2)` returns 1/-1 if segments are equal or opposite. Returns 0 otherwise.
- `int SegIntersect (dsegment s1,s2, int h, point& p)` returns h=1 if segments intersect internally, 0 if at an endpoint, and then returns the point in p. Returns -1 otherwise.

Here are some required functions:

- `int PointEqual(point p1,p2)` returns 1 if points are equal, else returns 0.
- `int SegEqual (dsegment s1,s2)` returns 1/-1 if segments are equal or opposite. Returns 0 otherwise.
- `int SegIntersect (dsegment s1,s2, int h, point& p)` returns h=1 if segments intersect internally, 0 if at an endpoint, and then returns the point in p. Returns -1 otherwise.

```
int SegEqual(dsegment s1,s2)
{
 if(PointEqual(s1.start,s2.start)==1)
  {
   if(PointEqual(s1.end,s2.end)==1)
      return (1)
    else
      return(0);
  };
 if(PointEqual(s1.start,s2.end)==1)
  {
   if(PointEqual(s1.end,s2.start)==1)
      return (-1)
    else
      return(0);
  };
  return(0);
}
```

Lots of Nice Problems

In other words:

- `structs` enable us to think abstractly about our problem.
- They help us in organizing our programs for modularity and maintainability.
- They enable us to develop code as a team.

Lots of Nice Problems

In other words:

- `structs` enable us to think abstractly about our problem.
- They help us in organizing our programs for modularity and maintainability.
- They enable us to develop code as a team.

Assignment

Write a program to check if a `polygon pp` is indeed a vaild non-intersecting, anti-clockwisely oriented polygon.

Write a program to check if a point p is in the interior of a valid polygon.