CS 101: Computer Programming and Utilization

Shivaram Kalyanakrishnan
(Abhiram Ranade's slides, borrowed and edited)
Lecture 20

Where are we in the course…

You have learned enough to write essentially any program.

- Basic control statements
- Functions
- Arrays

Next lectures: Language features that will make writing programs more convenient, safer, modular.

- These terms will become clearer soon.

Today's Lecture

- Structures
- Structure initialisation, assignment
- Structures and functions

A difficulty

- A large program will have lots of variables.
- Just managing all the variables is tiring.
  - “Lots of papers strewn over the table”
  - We can bring some neatness by putting related papers into files.
  - Can we do something like that with variables?
- The solution: “structures”

Structures – high level idea

Most entities we deal with in programming have lots of attributes.

- If our program is about simulating movement of stars
  - Each star has a position, velocity, mass, ...
- If our program is about managing books in a library
  - Each book has author, library number, who has borrowed it, ...
- **Key idea**: collect together all information about an entity into a group/supervariable = structure

Defining a structure type

- General form

```
struct structure-type{
    member1-type member1-name;
    member2-type member2-name;
    …
};          // Don't forget the semicolon!
```

- Example

```
struct Book{
    char title[50];
    double price;
};
```

- A structure-type is a user-defined data type, just as `int`, `char`, `double` are primitive data types.
- Structure-type and member names can be any identifiers.

Creating structures of a type defined earlier

- To create a structure of structure type `Book`, just write:

```
Book p, q;
```

- This creates two structures: `p, q of` type `Book`.
- Each created structure has all members defined in structure type definition.
- Member x of structure y can be accessed by writing `y.x`

```
p.price = 399;
// stores 399 into p.price.
cout << p.title;
// prints the name of the book p
```

Structures: overview

- Structure = collection of variables
- Members
  - Structure = super variable, denotes the memory used for all members.
  - Each structure has a name, the name refers to the super variable, i.e. entire collection.
  - Each structure has a type: the type defines what variables there will be in the collection.

Structure types

- You can define a structure type for each type of entity that you want to represent on the computer.
  - "Programmer defined type"
- Example: To represent books, you can define a `Book` structure type.
- When you define a structure type, you must say what variables each structure of that type will contain.
- Example: In a structure to represent books, you may wish to have variables to store the name of the book, its price, …

Today's Lecture

- Structures
- Structure initialisation, assignment
- Structures and functions

Initialising structures during creation

```
struct Book{char title[50]; double
price; };
Book b = {“On Education”, 399};
```

- Stores “On Education” in `b.title` (null terminated as usual) and 399 into `b.price`.
- A value must be given for initializing each member.
- You can make a structure unmodifiable by adding the keyword `const`:

```
const Book c = {“The Outsider”, 250};
```

One structure can contain another

```
struct Point{
   double x,y;
};
struct Disk{
   Point center;    // contains Point
   double radius;
};
Disk d;
d.radius = 10;
d.center.x = 15;
// sets the x member of center member of
d
```

Assignment

- One structure can be assigned to another.
  - All members of right hand side copied into corresponding members on the left.
  - Structure name stands for entire collection unlike array name which stands for address.
  - A structure can be thought of as a (super) variable.

```
book b = {“On Education”, 399};
book c;
c = b;     // all members copied.
cout << c.price << endl;
// will print 399.
```

Today's Lecture

- Structures
- Structure initialisation, assignment
- Structures and functions

Structures and functions

- Structures can be passed to functions by value
  - members are copied
- Structures can also be passed by reference.
  - Same structure is used in called function
- Structures can also be returned.
  - All data members are copied back to a temporary structure in the calling program

Passing by value

```
struct Point{double x, y;};
Point midpoint(Point a,
Point b){
  Point mp;
  mp.x = (a.x + b.x)/2;
  mp.y = (a.y + b.y)/2;
  return mp;
}

int main(){
  Point p={10,20},
q={50,60};
  Point r = midpoint(p,q);
  cout << r.x << endl;
  cout << midpoint(p,r).x <<
endl;
}
```

- Call `midpoint(p,q)` : `p`,`q` copied to parameters `a`,`b`.
- `midpoint` creates local structure `mp`.
- The value of `mp` is returned:

A nameless temporary structure of type `Point` is created in the activation frame of `main`.

`mp` is copied into the temporary structure

- The temporary structure is copied into structure `r`.
- `r.x` is printed.
- We can use the “.” operator on temporary structures, as in the second call.

Passing by reference

```
struct Point{double x, y;};
Point midpoint(const Point &a,
               const Point &b)
{
    Point mp;
    mp.x = (a.x + b.x)/2;
    mp.y = (a.y + b.y)/2;
    return mp;
}
int main(){
    Point p={10,20}, q={50,60};
    Point r = midpoint(p,q);
    cout << r.x << endl;
}
```

- In execution of `midpoint(p,q)` parameters `a,b` refer to variables `p,q` of `main`.
- There is no copying of `p,q`.
- Saves execution time if the structures being passed are large.
- The rest of the execution is as before.
- Normally, reference parameters are expected to be variables.
- `const` says that `a,b` will not be modified inside function.
- Enables `const` structures to be passed as arguments.

`midpoint(midpoint(.,.,.,.),..)`