IC150
Computation for Engineers

Lecture 21

Structures

Timothy Gonsalves

Marks and Names

- We kept Marks in an integer array and Names in a corresponding two dimensional array
  - could keep Names in an array of pointers
  - memory reserved as per requirements
- The connection between the two arrays was via the shared index
- Can we keep them in the same array?
  - one option – keep first three characters for marks and convert them while processing
  - a better option – use structures

```
099Usha
014Pravin
```


Structures

- Collection of one or more variables, possibly of different types, grouped together under a single name for easy handling.
- Eg. a structure which represents a point in a 2-d plane

```
struct point
 {
    int x;
    int y;
 };
```

A mechanism for defining compound data types

By itself it reserves no storage

Point in 2D → 2 integers

- Different ways of declaring structure variables

```
struct point{
      int x;
      int y;
 } point1, point2;
```

```
struct point point1, point2;
struct point point1 = { 3 , 2 };
```

Marks and Names

1.
```
struct student{
   char *name;
   int mark;
}s1, s2;
```

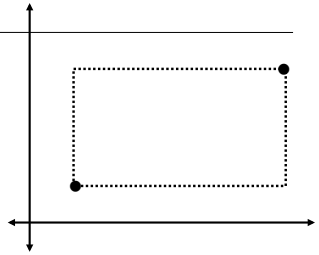
name could itself be a struct made up of first name middle name last name…

2.
```
   struct student s1, s2;
```

3.
```
   struct student s1 = { "Ramesh" , 79 };
```


A rectangle

```
struct rectangle{
        struct point pt1;
        struct point pt2;
   }rect1;
```



- Accessing points in the rectangle

```
rect1.pt1.x = 4;
rect1.pt1.y = 5;
```

Or

```
rect1.pt1 = { 4, 5 };
```


Defining new types

- 'typedef' is used for creating new data types, for example

```
typedef int Age;
Age myAge = 99;
```

- typedef and Structures

```
typedef struct point PointType;
PointType point1, point2;
```

- Is equivalent to `struct point point1, point2;`

Operations on structures

- Structures can be copied by assignment statement
  - `p1 = p2` equivalent to `p1.x = p2.x; p1.y = p2.y`
- &structure is the address of the structure
- Can be passed to functions and can be returned by functions
  - one can pass an entire structure
  - one can pass components
  - one can pass a pointer to a structure
- Structures cannot be compared

Functions and structures

- Structure as function argument

```
int IsOrigin(PointType pt)
  {
    if( pt.x ==0 && pt.y ==0)
              return 1;
    else
              return 0;
  }
```


Structures and functions

- Structure as return type

```
PointType MakePoint(int x, int y)
  {
     PointType *temp;
     temp = malloc(sizeof(PointType));
    temp->x = x;
    temp->y = y;

    return *temp;
  }
```

Observe there is no confusion between the two occurrences of x and y

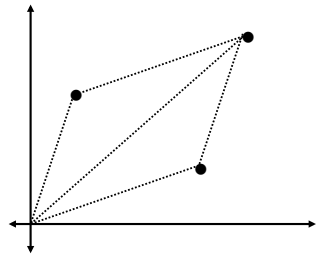
A screen and its centre point

```
struct rect screen;
struct point middle;
struct point MakePoint(int, int);

screen.pt1 = MakePoint(0, 0);
screen.pt2 = MakePoint(XMAX, YMAX);
middle = MakePoint(
  (screen.pt1.x + screen.pt2.x)/2,
   (screen.pt1.y + screen.pt2.y)/2);
```


adding two points

```
/* AddPoint: add two points */
struct point AddPoint(struct point p1,
  struct point p2)
{
     p1.x += p2.x;
     p1.y += p2.y;
     return p1;
}
```



IC150
Computation for Engineers

Lecture 22

Structures continued

Timothy Gonsalves

Point inside a rectangle?

```
/* IsPtInRect: return 1 if point p is in rectangle r, else
   return 0 */
int IsPtInRect(struct point p,
                    struct rect r)
{
  return p.x >= r.pt1.x && p.x < r.pt2.x
                    && p.y >= r.pt1.y &
                       p.y < r.pt2.y;
}
```

assume pt1 comes “before” pt2

a canonical rectangle

```
#define min(a, b) ((a < b) ? a : b)
#define max(a, b) ((a > b) ? a : b)

struct rect CanonRect(struct rect r)
  /* canonicalize coordinates of rectangle */
{
  struct rect temp;

  temp.pt1.x = min(r.pt1.x, r.pt2.x);
  temp.pt1.y = min(r.pt1.y, r.pt2.y);
  temp.pt2.x = max(r.pt1.x, r.pt2.x);
  temp.pt2.y = max(r.pt1.y, r.pt2.y);
  return temp;
}
```


Arrays of structures

```
struct point{
   int x;
   int y;
} pointArray[10];
```

```
struct point {
  int x;
  int y;
} pointArray[] = {
          { 1,2 },
          { 2,3 },
          { 3,4 }
      };
```

```
pointType pointArray[10];
```

Accessing member values

- Assigning values to structure elements

```
pointArray[0].x = 1;
pointArray[0].y = 2;
```

OR

```
pointArray[i].x = 5;
pointArray[i].y = 5;
```

- Printing elements of Structures

```
printf("(%d, %d )", pointArray[0].x,
                    pointArray[0].y);
```


structure1 = structure2

- Structures can be assigned using the assignment operator

```
struct point newPoint;
newPoint = MakePoint(4,4);
```


Structures (review)

- Often needed requirement
  - keeping related data items as one unit
  - for instance
- Name, Roll No, Gender, Hostel, Room No of a Student
  - logically belong together - need to be grouped
- the items may be of different types
  - name -- string
  - rollNo – integer
  - gender -- single character
  - hostel – string
  - roomNo -- integer

Example Structure Definition

```
typedef struct student{
 char name[30];
 int rollNo;
 char gender;
 char hostel[8];
 int roomNo;
} StudentType;
```

Components can be of any type - even struct

Observe the semi-colons

Creates - a new data type called *StudentType*

a composite type with 5 components

Can be used in type declarations of variables

```
StudentType jeyanthi, vikas, mahesh;
```

Another Definition

```
typedef struct book
{
   char title[20];
   char authors[30];
   int accNo;
   char subject[25];
} BookType;
BookType cText;       // a C textbook
BookType shelf[100];  // shelf holds 100 books
```


IC150

Lecture 23

Structures: Complex arithmetic

Timothy Gonsalves

Using Structures

Let us create a type for complex numbers and a few operations on complex numbers

```
typedef struct {
  float real;
  float imag;
} Complex;
Complex Sum(Complex m, Complex n);
Complex Product(Complex m, Complex n);
```


Using Complex Type

Dot Notation:

Accessing components of a structure

```
main(){
  Complex a,b,c,d;
  scanf("%f %f", &a.real, &a.imag);
  scanf("%f %f", &b.real, &b.imag);
  c = Sum(a,b);
  d = Product(a,b);
  printf("Sum of a and b is %f + i%f\n",
  c.real, c.imag);
  printf("Product of a and b is %f + i%f
  \n", d.real, d.imag);
}
```

Implementation of Sum and Product

```
Complex Sum(Complex m, Complex n){
  Complex p;
  p.real = m.real + n.real;
  p.imag = m.imag = n.imag;
  return (p);
}
```

```
Complex Product(Complex m, Complex n){
  Complex p;
  p.real = (m.real * n.real) –
                          (m.imag * n. imag);
  p.imag = (m.real * n.imag) +
                          (m.imag * n. real);
  return (p);
}
```


Pointers to structures

```
PointType point1, *ptr;
point1 = MakePoint(3,4);
ptr    = &point1;
printf("(%d,%d)", (*ptr).x,
                  (*ptr).y);
```

the paren are necessary

OR

```
printf("(%d,%d)", ptr->x, ptr->y);
```

equivalent short form

- The operator **->** (minus sign followed by greater than symbol) is used to access members of structures when pointers are used.

Precedence and association

- Both **.** and **->** associate left to right. They are at top of precedence hierarchy. The following forms are equivalent

```
struct rect r, *rp = r;
```

```
r.pt1.x
rp -> pt1.x
(r.pt1).x
(rp -> pt1).x
```


Examples

Given the declaration

```
struct {
    int len; char *str;
} *p;
```

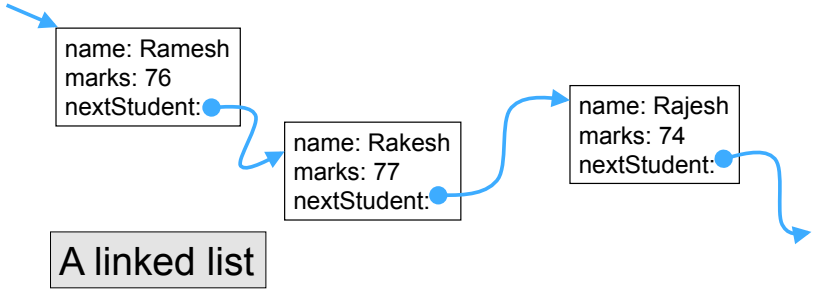
- ++p-> len increments len not p
  - implied paranthesis ++(p-> len)
- (++p)-> len increments p before accessing len
- p++-> len increments p afterwards
- *p->str fetches whatever str points to
- *p->str++ increments str after accessing whatever it points to (just like *s++)
- (*p->str)++ inceremens whatever str points to
- *p++-> str increments p after accessing whatever str points to

Dynamic data structures

- How does one cater to an uncertain and changing amount of memory requirements?
  - for example if the number of students writing an online surprise exam is unknown
- One way is dynamic tables / arrays
  - declare an array of some size N
  - if it seems to be getting full declare an array of twice the size and copy all elements into it
- The other is to ask for memory for a structure one at a time and *link* them together

Self referential structures

- The structure contains a pointer to a structure *of the same type*
  - this is in addition to the other data it stores
  - let student contain marks and name



Homework Exercise

- Write a program to take a filename as a command line argument, open the file and count the frequencies of the different words in the file.
- Given a "-n" option it should print the words preceded by their counts in an increasing order of frequency, one word per line.
- Otherwise it should print the words in alphabetic order

Exercise

- Suppose we have a travel agency which stores information about each flight:
  - Flight Number
  - Originating airport code (3 letters)
  - Destination airport code (3 letters)
  - Departure Time
  - Arrival Time
- User enters the origin and destination airport codes … get a list of matching flights

1. Define a structure(s) for the flight information
2. Write a function to read in the flight info for all flights
3. Write a function to find the info for a given origin and destination

Solution: FlightInfo structure

- We will start with a structure which represents flight information

```
struct FlightInfo
  {
      String flightNo;
      String origin;
      String destination;
      Time depTime;
      Time arrTime;
  };
```


String and Time types

- But 'C' does not have any 'String' or 'Time' data types. We can define them!

```
typedef char* String;
```

//Don't forget to allocate memory using malloc!!!!

OR

```
typedef char[10] String;
```

//But this will allocate more memory than actually required.

```
struct TimeData
{
  int hour;
  int minute;
};
typedef struct TimeData Time;
```


Reading in the data

```
struct FltInfo agency1[MAX_FLIGHTS];
void ReadInfo(int numFlights, struct FltInfo fltList[])
{
  for(i=1; i< numFlights; i++)  {
    printf("Enter Flight Number %d", i);
    scanf(" %s", fltList[i].flightNo);
    printf("Enter Origin (3 letter code): ");
    scanf(" %s", fltList[i].origin);
    printf("Enter Destination(3 letter code): ");
    scanf(" %s", fltList[i].destination);
    printf("Enter Departure Time (hh:mm) : ");
    scanf(" %d%d", &fltList[i].depTime.hour,
                   &fltList[i].depTime.minute);
    printf("Enter Arrival Time (hh:mm) : ");
    scanf(" %d%d", &fltList[i].arrTime.hour,
                   &fltList[i].arrTime.minute);
  }
}
```



```
void RetrieveFlight(struct FlightInfo flightList[], int numFlights)
{
   String userOrigin, userDest;
   printf("\nEnter the origin and destination airport codes: ");
   scanf(" %s, %s", userOrigin, userDest);

  for (int i=0;  i < numFlights; i++)
    if ((strcmp(flightList[i].origin, userOrigin) ==0 ) &&
          (strcmp(flightList[i].destination, userDest) == 0))
       printf("\nFlight Number: %s \n", flightList[i].flightNo);
   printf("Departure Time: %d: %d\n",
       flightList[i].depTime.hour,  flightList[i].depTime.minute );
  printf("Arrival Time: %d: %d \n", flightList[i].
       arrTime.hour,  flightList[i].arrTime.minute );
}
```

Storing and accessing elements

- Linked list are accessed by following the pointers
  - linear time complexity
- Search trees are traversed by comparing values at nodes
  - logarithmic time complexity for balanced trees
- Array elements are accessed by using the index
  - constant time complexity
  - *index value should be known (else search)*
- Can we store names/strings in arrays?
  - and find them in constant time
  - Yes, in Hash tables (average complexity)

Computer Solutions

Problem Solving

main purpose of using computers

Steps

clear specification, understanding of the problem

remove unnecessary details and retain only the required parameters, constraints of the problem

"abstraction" - better insight, helps in thinking

find the method of solving the problem

"algorithm design" - "efficiency"

express the solution in a programming language

References

GN87: Peter Grogono and Sharon Nelson, *Problem Solving and Computer Programming*, Narosa, 1987

D96 : R G Dromey, *How to solve it by computer*, Prentice-Hall India, 1996