CS1100
Introduction to Programming

Structures

Structures

- Collection of one or more variables, possibly of different types, grouped together under a single name for easy handling.
- For example - a structure which represents a point in a two dimensional 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

- Declaring structure variables

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

```
struct point point1, point2, pointB,
HelloVar1234; // Un-initialized
struct point point3 = {3, 2};  // Initialized
```


Extra Q

Write the function prototype to match the function call in main()

```
_______ Diwali ( _____________________);

int main(){
  double d1;
  int j, arr[100];
  float *fptr;
  fptr = Diwali(j, &d1, arr);
}
```

Point in 2D → 2 Integers

- Two ways of declaring structure variables

```
struct point{
    int x;
    int y;
};
struct point point1, point2; // two variables.
```

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


Marks and Names

```
struct student{
 int rollno;
 char *name;
};

struct student s1, s2;
struct student s3 = {79, "Ram" };
```

name could itself be a struct made up of first name, middle name and last name…
Nested structures are allowed

Defining New Types

- '***typedef***' keyword 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;
```

- This is equivalent to: `struct point point1, point2;`

Marks and Names

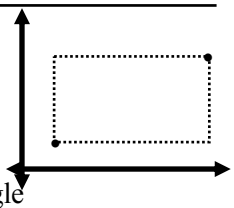
```
typedef struct student{
 int rollno;
 char *name;
} Student_t;

Student_t s1, s2;
Student_t s3 = {79, "Ram" };
```

name could itself be a struct made up of first name, middle name and last name…
Nested structures are allowed

A Rectangle

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



- Accessing points in the rectangle

```
rect1.pt1.x = 4; rect1.pt1.y = 5;
rect1.pt2.x = 3; rect.pt2.y = 0;
```

Or

```
rect1.pt1 = {4, 5}; rect1.pt2 = {3,0};
```


Operations on Structures

- Structures may be copied by assignment statement
  - point1 = point2; // both are of type point
- The address of the structure (use &) can be passed to functions and can be returned by functions
  - one can pass an entire structure
  - one can pass some components of a structure
  - one can pass a pointer to a structure
- Structures may not be compared
  - if (point1 == point2) // not allowed in C

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.x = x;
        temp.y = y;
        return temp;
  }
```

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

Structure1 = Structure2

- Structures can be assigned using the assignment operator

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

After this, newPoint.x will be 4, etc.

Arrays of Structures

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

```
struct point {
    int x;
    int y;
} pointArray3[ ] = {
        {1, 2},
        {2, 3},
        {3, 4}
    };
```

```
typedef struct point pointType;
```

```
pointType pointArray2[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);
```


Accessing Member Values, contd.

- Reading into elements of Structures

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

A Screen and its Centre Point

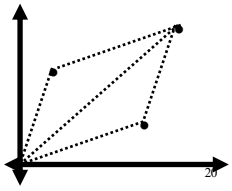
```
struct rectangle screen;
struct point middle;
struct point makePoint(int, int);

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

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

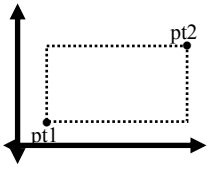
Note that the local changes to p1 would not affect the point p1; *pass by value*



Point Inside a Rectangle?

- /* isPtInRect: return 1 if point p is in rectangle r, else return 0 */

```
int isPtInRect(struct point p, struct rectangle r){
  return (p.x >= r.pt1.x) && (p.x < r.pt2.x) &&
         (p.y >= r.pt1.y) && (p.y < r.pt2.y);
}
```



A Canonical Rectangle

```
#define min(a, b) ((a<b)?a:b)  /*Macro definitions*/
#define max(a, b) ((a>b)?a:b)
struct rectangle canonRect(struct rect r){
            /*canonicalize coordinates of rectangle*/
  struct rectangle 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;
}
```

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 jayanthi, 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];     // a shelf holds 100 books
```


Using Structures

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

```
typedef struct {
double real;
        double imag;
} Complex;

Complex Sum (Complex m, Complex n);
Complex Product (Complex m, Complex n);
```


Using Complex Type

Dot (.) Notation:

Accessing components of a structure

```
int 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);
}
```

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

The brackets are necessary. Otherwise it is taken as *(ptr.x)

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
- If we have
  - *struct* rectangle r, *rp = &r;
  - The following forms are equivalent

    r.pt1.x (r.pt1).x

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

    (*rp).x

Recall: Precedence & Associativity of Operators

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

Bitwise operators

Table from K & R book 2nd edn. page 53

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
- Define a structure(s) for the flight information
- Write a function to read in the flight info for all flights
- 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 FlightInfo agency1[MAX_FLIGHTS];
void ReadInfo(int numFlights, struct FlightInfo flightList[ ]){
for (i=1; i< numFlights; i++){
        printf("Enter Flight Number %d", i);
        scanf(" %s", flightList[i].flightNo);
        printf("Enter Origin (3 letter code): ");
        scanf(" %s", flightList[i].origin);
        printf("Enter Destination(3 letter code): ");
        scanf(" %s", flightList[i].destination);
        printf("Enter Departure Time (hh:mm): ");
        scanf(" %d%d", &flightList[i].depTime.hour,

&flightList[i].depTime.minute);          printf("Enter Arrival Time (hh:mm): ");
        scanf(" %d%d", &flightList[i].arrTime.hour,
                                &flightList[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 );
    }
}
```