# Pointers: Part 2
# 9th Mar 12

# Structures Revisited

- Recall that a structure can be declared as:

```
struct  stud  {
            int   roll;
            char  dept_code[25];
            float  cgpa;
          };
struct  stud  a, b, c;
```

- And the individual structure elements can be accessed as:

```
a.roll ,  b.roll ,  c.cgpa , etc.
```

# Arrays of Structures

- We can define an array of structure records as

```
struct   stud   class[100] ;
```

- The structure elements of the individual records can be accessed as:

```
class[i].roll
class[20].dept_code
class[k++].cgpa
```

# Example: Sorting by Roll Numbers

```
#include <stdio.h>
struct   stud    {
     int    roll;
     char  dept_code[25];
     float  cgpa;
 };

int main()
{
     struc  stud  class[100], t;
     int  j, k, n;

     scanf  ("%d", &n);
                    /* no. of students */
```

```
for  (k=0; k<n; k++)
        scanf ("%d %s %f", &class[k].roll,
              class[k].dept_code, &class[k].cgpa);
    for  (j=0; j<n-1; j++)
        for  (k=j+1; k<n; k++)
          {
              if  (class[j].roll > class[k].roll)  {
                        t = class[j] ;
                        class[j] = class[k] ;
                        class[k] = t
               }
          }
   <<<< PRINT THE RECORDS >>>>
}
```

# Pointers and Structures

- You may recall that the name of an array stands for the address of its zero-th element.
  - Also true for the names of arrays of structure variables.
- Consider the declaration:

```
struct  stud  {
            int    roll;
            char  dept_code[25];
            float  cgpa;
          }  class[100],  *ptr ;
```

- The name class represents the address of the zero-th element of the structure array.
- ptr is a pointer to data objects of the type struct stud.

- The assignment

```
ptr = class ;
```

will assign the address of class[0] to ptr.

- When the pointer ptr is incremented by one (ptr++) :
  - The value of ptr is actually increased by sizeof(stud).
  - It is made to point to the next record.

- Once ptr points to a structure variable, the members can be accessed as:

```
ptr –> roll ;
ptr –> dept_code ;
ptr –> cgpa ;
```

  – The symbol "–>" is called the arrow operator.

# Example

```
#include <stdio.h>
typedef struct {
        float real;
        float imag;
       } _COMPLEX;
```

```
swap_ref(_COMPLEX *a, _COMPLEX *b)
{
  _COMPLEX tmp;
   tmp=*a;
   *a=*b;
   *b=tmp;
}
```

```
print(_COMPLEX *a)
{
  printf("(%f,%f)\n",a->real,a->imag);
}
```

```
int main()
{
 _COMPLEX x={10.0,3.0}, y={-20.0,4.0};

  print(&x); print(&y);
  swap_ref(&x,&y);
  print(&x); print(&y);
}
```

```
(10.000000,3.000000)
(-20.000000,4.000000)
(-20.000000,4.000000)
(10.000000,3.000000)
```

# A Warning

- When using structure pointers, we should take care of operator precedence.
  - Member operator "." has higher precedence than "*".
    - ptr –> roll and (*ptr).roll mean the same thing.
    - *ptr.roll will lead to error.
  - The operator "–>" enjoys the highest priority among operators.
    - ++ptr –> roll will increment roll, not ptr.
    - (++ptr) –> roll will do the intended thing.

# Structures and Functions

- A structure can be passed as argument to a function.
- A function can also return a structure.
- The process shall be illustrated with the help of an example.
  - A function to add two complex numbers.

# Example: complex number addition

```
#include <stdio.h>
struct complex {
                    float re;
                    float im;
                  };

int main() {
   struct complex a, b, c;
   scanf ("%f %f", &a.re, &a.im);
   scanf ("%f %f", &b.re, &b.im);
   c = add (a, b) ;
   printf ("\n %f %f", c.re, c.im);
}
```

```
struct complex add ( struct complex x,
                     struct complex y) {
   struct complex t;
   t.re = x.re + y.re ;
   t.im = x.im + y.im ;
   return (t) ;
}
```

# Example: Alternative way using pointers

```
#include <stdio.h>
struct complex {
                    float re;
                    float im;
                  };

int main() {
   struct complex a, b, c;
   scanf ("%f %f", &a.re, &a.im);
   scanf ("%f %f", &b.re, &b.im);
   add (&a, &b, &c) ;
   printf ("\n %f %f", c,re, c.im);
}
```

```
void add (struct complex *x,
          struct complex *y,
          struct complex *t)
{
    t->re = x->re + y->re ;
    t->im = x->im + y->im ;
}
```

```
#include <stdio.h>
struct foo  {  // a global definition, the struct foo is known
    int a, b, c;          // in all of these functions
};
//  function prototypes
void inp (struct foo *);
void outp (struct foo);

int main( )   {
        struct foo x;   // declare x to be a foo
        inp(&x);     // get its input, passing a pointer to foo
        outp(x);  //send x to outp, requires 2 copying actions
}
```

```
void inp(struct foo *x)
{
        scanf("%d%d%d", &x->a, &x->b, &x->c);
}

void outp(struct foo x)
{
        printf("%d %d %d\n", x.a, x.b, x.c);
}
```

# Nested structs

- In order to provide modularity, it is common to use already-defined structs as members of additional structs

```
struct point {
    int x;
    int y;
 }
struct rectangle {
    struct point pt1;
    struct point pt2;
}
```

**If we have**

```
struct rectangle r;
```

**Then we can reference**

**r.pt1.x, r.pt1.y,**

**r.pt2.x and r.pt2.y**

**Now consider the following**

```
struct rectangle r, *rp;
rp = &r;
```

**Then the following are all equivalent**

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

**But not `rp->pt1->x` (since pt1 is not a pointer to a point)**

# Arrays of structs

- To declare an array of structs (once you have defined the struct):

```
struct rectangle rects[10];
```

  - rects now is a group of 10 structures (that consist each of two points)
  - You can initialize the array as normal where each struct is initialized as a { } list as in {5, 3} for a point or {{5, 3}, {8, 2}} for a rectangle

# Example

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

struct rectangle {
    struct point p1;
    struct point p2;
};
```

```
void printRect(struct rectangle r)
{
        printf("<%d, %d> to <%d, %d>\n",
r.p1.x, r.p1.y, r.p2.x, r.p2.y);
}

int main( )
{
    int i;
    struct rectangle rects[ ] =
     {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};
                              // 2 rectangles
    for(i=0;i<2;i++) printRect(rects[i]);
}
```