Structures

EECS 2031

Basics of Structures (6.1)

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

keyword **`struct`** introduces a structure declaration.

`point`: *structure tag*

`x, y`: *members*

The same member names may occur in different structures.

- Now **`struct point`** is a valid type.
- Defining **`struct`** variables:

```
struct point pt;
struct point
  maxpt = {320, 200};
```

- A **`struct`** declaration defines a type.

  `struct { ... } a, b, c;`

  or **`struct point a,b,c;`**

  is syntactically analogous to

  `int a, b, c;`

Using Structures

- Members are accessed using operator "."

```
    structure-name.member
struct point pt;
printf("%d,%d", pt.x, pt.y);
double sqrt(double);  /* function prototype */
double dist = sqrt((double)pt.x * pt.x +
                   (double)pt.y * pt.y);
```

3

Structure Name Space

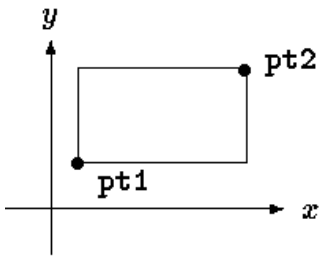
- Structure and members names have their own name space separate from variables and functions.

```
struct point point;
struct point {
    int x;
    int y;
} x;
/* Both are valid but not recommended. */
```

4

Nested Structures

```
struct rectng {
  struct point pt1;
  struct point pt2;
};
struct rectng screen;
screen.pt1.x = 1;
screen.pt1.y = 2;
screen.pt2.x = 8;
screen.pt2.y = 7;
```



5

Structures and Functions (6.2)

- Returning a structure from a function.

```
/* makepoint: make a point from x and y components */
struct point makepoint( int x, int y ) {
   struct point temp;
   temp.x = x;
   temp.y = y;
   return temp;
}
struct rectng 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 );
```

6

Structures and Functions (cont.)

- Passing structure arguments to functions: structure parameters are passed by values like int, char, float, etc. A copy of the structure is sent to the function.

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

- *Note*: the components in p1 are incremented rather than using an explicit temporary variable to emphasize that structure parameters are passed by value like any others (no changes to original struct).

7

Pointers to Structures

- If a large structure is to be passed to a function, it is generally more efficient to pass a pointer than to copy the whole structure.

```
struct point *pp;
struct point origin = makepoint( 0, 0 );
pp = &origin;
printf("origin is (%d,%d)\n", (*pp).x, (*pp).y);
```

- Note: *pp.x means *(pp.x), which is illegal (why?)

8

Precedence and Associativity

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

9

Pointers to Structures: Example

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

main() {
   struct point a, b, c;
   /* Input or initialize structures a and b */
   c = addpoint( &a, &b );
}
```

10

Pointers to Structures: Shorthand

- `(*pp).x` can be written as `pp->x`

```
printf("origin is (%d,%d)\n", pp->x, pp->y);

struct rect r, *rp = &r;
r.pt1.x = 1;
rp->pt1.x = 2;
(r.pt1).x = 3;
(rp->pt1).x = 4;
```

- Note: Both . and -> associate from left to right.

11

malloc()

```
pointer = (type *) malloc
          (no_of_elements * sizeof (element))
double *pd;
pd = (double *) malloc(sizeof(double));
int *pi;
pi = (int *) malloc(sizeof(int));
free ( pi );
pi = (int *) malloc(10 * sizeof(int));
char *str;
str = (char *) malloc(MAXSIZE *
                      sizeof(char));
```

12

Arrays of Structures (6.3)

```
struct dimension {
   float width;
   float height;
};
struct dimension chairs[12];
struct dimension *tables;
tables = (struct dimension *) malloc
 (20 * sizeoff(struct dimension));
```

13

Initializing Structures

```
struct dimension sofa = {2.0, 3.0};

struct dimension chairs[] = {
                    {1.4, 2.0},
                    {0.3, 1.0},
                    {2.3, 2.0} };
```

14

Arrays of Structures: Example

```
struct key {
   char *word;
   int count;
};

struct key keytab[NKEYS];
struct key *p;
for (p = keytab;
    p < keytab + NKEYS; p++)
   printf("%4d %s\n",
       p->count, p->word);
```

```
struct key {
   char *word;
   int count;
} keytab[] = {
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
};
```

15

Pointers to Structures (6.4)

```
struct key keytab[NKEYS];
struct key *p;
for (p = keytab; p < keytab + NKEYS; p++)
   printf("%4d %s\n", p->count, p->word);
```

- p++ increments p by the correct amount (i.e., structure size) to get the next element of the array of structures.

```
struct {
   char c;  /* one byte */
   int i;   /* four bytes */
};
```

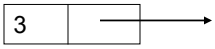
- What is the total structure size?
- Use the **`sizeof`** operator to get the correct structure size.

16

Self-referential Structures (6.5)

Example: (singly) linked list

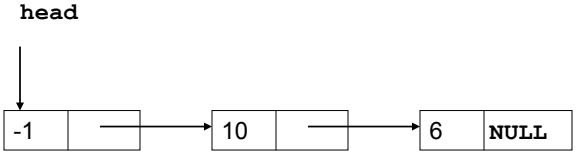
```
struct list  {
    int data;
    struct list *next;
 };
```



17

Linked List

- Pointer **`head`** points to the first element
- Last element pointer is **`NULL`**
- Example (next slide): build a linked list with **`data`** being non-negative integers, then search for a number.
  - Insertion at the end (rear) of the list.
- We also learn how to dynamically allocate a structure.



18

Linked List Implementation

```
#include <stdio.h>
#include <stdlib.h>
main() {
    struct list {
            int data;
            struct list *next;
    } *head, *p, *last;
    int i;

    /* Create a dummy node, which
       simplifies insertion and deletion */
    head = (struct list *) malloc
            ( sizeof( struct list ) );
    head—>data = -1;
    head—>next = NULL;
    last = head;
    scanf( "%d", &i );  /* input 1st element */
    while( i >= 0 ) {
            p = (struct list  *)
                  malloc( sizeof( struct list ) );
            p—>data = i;
            p—>next = NULL;
            last—>next = p;
            last = p;
            scanf( "%d", &i );
    } /* while */

    printf("Enter the number to search for  ");
    scanf( "%d", &i );
    for( p = head;  p != NULL; p = p—>next )
            if( p—>data == i )
              printf( "FOUND %d \n", i );
} /* main */
```

19

Pointers to Structures: More Examples

- The operators . and -> along with ( ) and [ ] have the highest precedence and thus bind very tightly.

```
int main() {
  struct string {
    int len;
    char *str;
  } *p;
 p = ( struct string * ) malloc( 3 * sizeof( struct string ) );
 if( !p ) {
    printf( "malloc error!\n" );
    exit ( 1 );
 }
 p->str = ( char * ) malloc( 5 * sizeof( char ) );
 if( !( p->str ) ) {
    printf( "malloc error!\n" );
    exit ( 2 );
 }
 p->str[0] = 'A'; p->str[1] = 'D'; p->str[2] = 'M'; p->str[3] = '\0';
 p->len = 3;
 return( 0 );
}
```

20

Example 1

```
++p->len          ⇔ ++(p->len)
(++p)->len
(p++)->len        ⇔ p++->len
```

21

Example 2

```
printf( "%s\n", p->str );
printf( "%c\n", *p->str );
printf( "%c\n", *p->str++ );
printf( "%c\n", *p->str );
(*p->str)++;
printf( "%c\n", *p->str );
printf( "%c\n", *p++->str );
printf( "%d\n", p->len );
```

22

`typedef` (6.7)

- For creating new data type names

```
typedef int Length;
Length len, maxlen;
Length *lengths[];

typedef char *String;
String p, lineptr[MAXLINES];
p = (String) malloc(100);
int strcmp(String, String);
```

23

`typedef` with `struct`

- We can define a new type and use it later

```
typedef struct {
      int x,y;
      float z;
   } mynewtype;
mynewtype a, b, c, x;
```

- Now, `mynewtype` is a type in C just like `int` or `float`.

24

Self-referential Structures: More Examples

- Binary trees (6.5)
- Hash tables (6.6)

To be covered later if time permits.

25

Reminders

- Lab test 1 (Oct. 20 and 23)
- Midterm (Oct. 30)
- Next lecture: Pointers (part 2)

26