Structures in C

EECS 2031

Summer 2014

Przemyslaw Pawluk

May 26, 2014

What we will discuss today

- Structures
- Pointers and Structs
- More complex types
- Self-referential Structures
- Homework

Table of Contents

Array

What is a Structure?

A `struct` (structure) in C is a **complex data type declaration** that defines a **physically grouped list of variables** to be **placed under one name in a block of memory**.

- allows the different variables to be accessed via a single pointer,
- can contain many other complex and simple data type in an association,
- is a natural organizing type for records like the mixed data types in lists of directory entries reading a hard drive

Example

Syntax

```
struct name{
    type1 name1;
    type2 name2;
    ...
    typeN nameN;
};
```

Declaration examples

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

Example

Point

Syntax

```
struct point { //structure tag
    int x; // member
    int y; // member
};
```

Now `struct point` is a valid type.

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

Note:

The same member names may occur in different structures.

Using structures

Accessing members

Members are accessed using operator "."

e.g. `structure-name.member`

Example

```
printf("%d,%d", pt.x, pt.y);

dist = sqrt((double)pt.x * pt.x +
    (double)pt.y * pt.y);
```

Note: Structures cannot be assigned.

```
struct point pt1, pt2;
pt1.x = 0; pt1.y = 0;
pt2 = pt1; /* WRONG !!! */
```

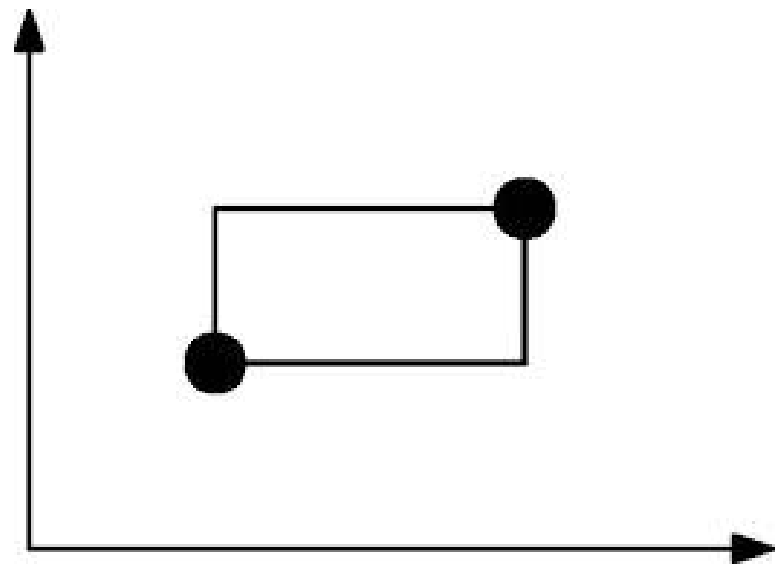
Nested structures

Structures can be nested

You can build more complex structures using simpler by nesting (e.g. Rectangle is defined by two points)

Example

```
struct rect {
    struct point pt1;
    struct point pt2;
};
struct rect screen;
screen.pt1.x = 1;
screen.pt1.y = 1;
screen.pt2.x = 3;
screen.pt2.y = 2;
```



Structures and Functions

You can return 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 rect screen;
struct point middle;
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);
```

Structures and Functions

Passed by value

Structure parameters are passed by values like int, char, float, etc.
A copy of the structure is sent to the function.
No changes to original struct are visible outside

Example

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

Table of Contents

Pointers to Structures

Syntax

```
struct type * varName;
```

Example

```
struct point *pp;
```

Why?

Useful when 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.

Note

`*pp.x` means `*(pp.x)`, which is illegal (why?)

Pointers to Strictures

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 );
}
```

Pointers to Structures

Shorthand

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

Example

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

Note

Both `.` and `->` associate from left to right

Table of Contents

Arrays of Structures

Syntax

```
struct type varName[size];
```

Example

```
struct dimension {
    float width;
    float height;
 };

struct dimension chairs[2];

struct dimension *tables;

tables = malloc(20 * sizeof(struct dimension));
```

Initialization

How?

You can use a syntax similar to the one used for array initialization

Example

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

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

Dynamic allocation of structures

Structs can be allocated dynamically

One can use `malloc` or `calloc` to allocate the memory

What is the total structure size?

Use the `sizeof()` operator to get the correct structure size.

Example

```
tables = malloc(20 * sizeof(struct dimension));
```

Note

$$\texttt{sizeof(S)} \neq \sum_i \texttt{sizeof(S->xi)}$$

Dynamic allocation of structures

Structs can be allocated dynamically

One can use `malloc` or `calloc` to allocate the memory

What is the total structure size?

Use the `sizeof()` operator to get the correct structure size.

Example

```
tables = malloc(20 * sizeof(struct dimension));
```

Note

$$\textbf{sizeof(S)} \neq \sum_i \textbf{sizeof(}\texttt{S->xi}\textbf{)}$$

Table of Contents

Linked List

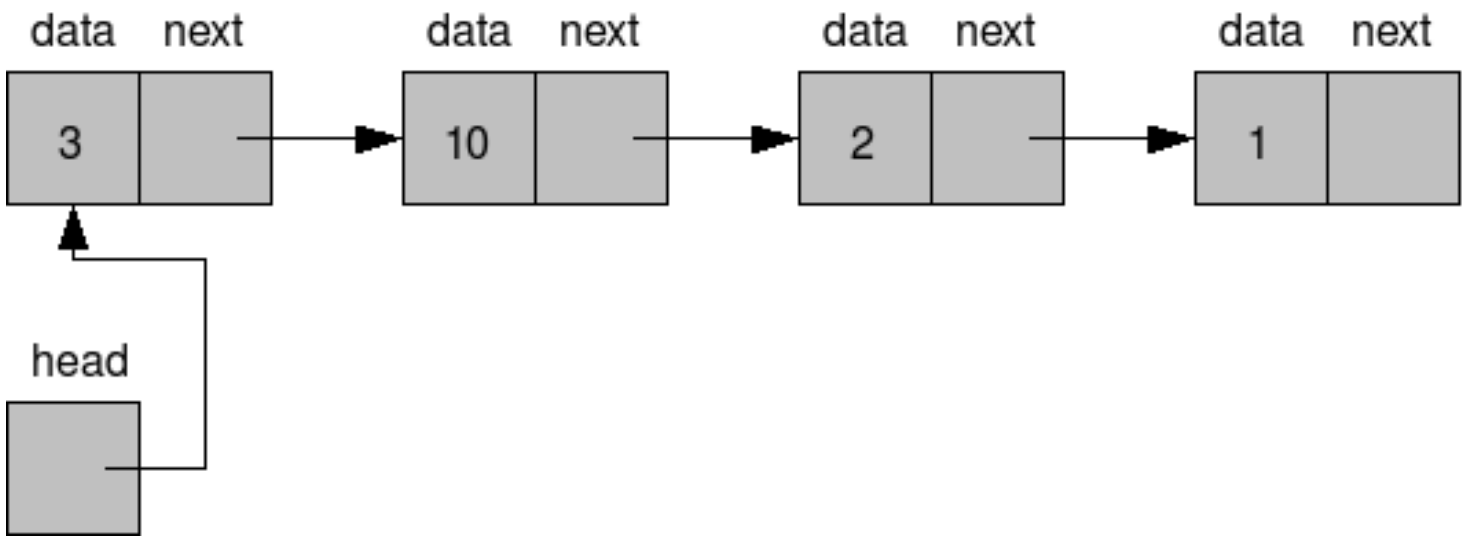
- Pointer head points to the first element
- Last element pointer is NULL
- We also learn how to dynamically allocate a structure.
- Add elements
  - at the end
  - at the beginning
  - in the middle

Linked List

Example

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

struct list *head;
```



typedef

Why?

For creating new data type **names**

Example

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

typedef and struct

We can define a type to simplify the code e.g., `mynewtype` is a type in C just like `int` or `float`.

Example

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

mynewtype a, b, c, x;
```

Table of Contents

Homework

Create a simple C program that:

- Reads integers from the stdin
- Puts them into a binary tree
  - if new element is smaller add it as a left leaf
  - if new element is bigger add it as a right leaf
- Travers the tree in-order (left-parent-right) and prints the values of all nodes