

Today: Structures, defining your own types

Computer Programming 143 – Lecture 22

C Structures I

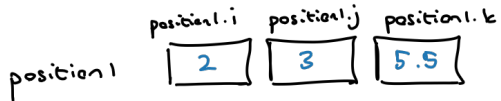
Faculty of Engineering



Defining a struct (your own type)

```
struct <name>  
{  
    <members>  
}; // often forget
```

struct vector force1 = {0.5, 5, -0.5};



10.2 Structure Definitions II

Example II

```
struct student {  
    char name[ 30 ];  
    int age;  
    long studentNumber;  
};
```

Example III

```
struct node {  
    struct node *leftPtr;  
    struct node *rightPtr;  
};
```

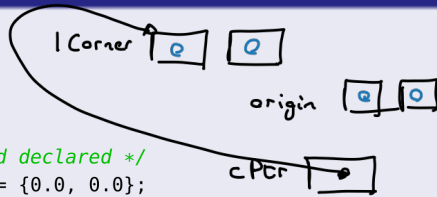
struct information

- A struct cannot contain a member that is an instance of itself
- Can contain a member that is a pointer to the same structure type (link lists, etc. – Chap. 12)
- A structure definition does not reserve space in memory
 - Instead creates a new data type used to define structure variables

10.2 Structure Definitions III

Valid Operations

```
struct coord {  
    double x;  
    double y;  
};  
/* two variables of type coord declared */  
struct coord lCorner, origin = {0.0, 0.0};  
struct coord *cPtr;
```



- 1 Assigning a structure variable to a structure variable of the same type
`lCorner = origin;`
- 2 Taking the address (&) of a structure variable
`cPtr = &lCorner;`
- 3 Accessing the members of a structure variable (next sections)
- 4 Using the `sizeof` operator to determine the size of a structure variable

10.3 Initialising Structures

Example

```
struct coord origin = {0.0, 0.0};
```

Example

```
struct person {  
    char firstName[ 20 ];  
    char lastName[ 20 ];  
    int age;  
    char gender;  
};  
  
struct person DEls = {"Danie", "Els", 22, 'M'};
```



Els Danie ✓ · 2nd

Lecturer Mechanical Engineering at University of Stellenbosch
City of Cape Town, Western Cape, South Africa · [Contact info](#)

466 connections



Steve, Corne and 13 other mutual connections



Connect



Message



University of Stellenbosch



University of Stellenbosch

10.4 Accessing Members of Structures

- Dot operator (.) used to access members by means of the *structure name*
- Arrow operator (->) used to access members by means of a *pointer to the structure* (no spaces - >)

```
struct coord {  
    double x;  
    double y;  
};  
  
struct coord c, *cPtr;  
  
cPtr = &c;  
  
/* The following assignments are all equivalent */  
    c.x = 10.5;           c.y = 12.3;  
(*cPtr).x = 10.5;       (*cPtr).y = 12.3;  
cPtr->x = 10.5;          cPtr->y = 12.3;
```

Shorthanding struct declarations

Defining a structure variable is more clunky than standard variables.

For instance:

```
int a;
```

vs

```
struct coord a;  
Coordinate a;
```

Something that helps us create shorthands:

typedef

10.6 typedef

typedef

- Creates synonyms (aliases) for previously defined data types
- Use typedef to create shorter type names
- Example:

Option 1

```
/*Option 1*/  
struct coord {  
    double x;  
    double y;  
};  
typedef struct coord Point;
```

Option 2

```
/*Option 2*/  
typedef struct {  
    double x;  
    double y;  
} Point;  
Point pa, pb;
```

Point pa, pb;

- Defines a new type name Point as a synonym for type struct coord
- typedef does not create a new data type, only an alias

Today

C Structures

- Introduction to structures
- Structure definitions
- Initialising structures
- Accessing structure members
- Using structures with functions
- typedef

Main :

point A [10] [11]

point B [1] [2]

point Sum [11] [13]

add Coordinates

point 1 [10] [11]

point 2 [1] [2]

result [11] [13]