Enums & Structs

Outline

- Enum definition and Examples
- Struct basic examples
- Struct data type definition
- Struct and pointers

Enums

Enums

An enumeration is a user-defined data type that consists of a list of integer constants. To define an enumeration, keyword `enum` is used.

Enums

An enumeration is a user-defined data type that consists of a list of integer constants. To define an enumeration, keyword `enum` is used.

Example:

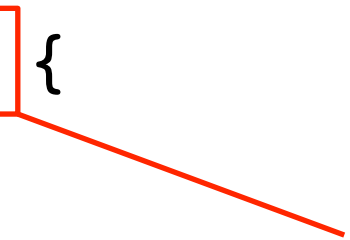
```
enum flag {
   const1,
   const2,
   …,
   constN
};
```

Enums

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword `enum` is used.

Example:

```
enum flag {
    const1,
    const2,
    …,
    constN
};
```



Enums

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword `enum` is used.

Example:

```
enum flag {
    const1,
    const2,
    …,
    constN
};
```

Values of type flag.

By default

const1 =0

const2= 1

…

Enums

You can change default values of enum elements during declaration (if necessary).

Example:

```
enum color {
      RED, BLU, GREEN, YELLOW, PINK, ORANGE
};
```

Enums

You can change default values of enum elements during declaration (if necessary).

Example:

```
enum color {
      RED, BLU, GREEN, YELLOW, PINK, ORANGE
};
       0      1     2        3         4       5
```

When to Use Enums?

- When you need a predefined list of values which do not represent some kind of numeric or textual data
- When a variable can only take one out a small set of possible values.

When to Use Enums?

- When you need a predefined list of values which do not represent some kind of numeric or textual data
- When a variable can only take one out a small set of possible values.

Example

```
enum week { mon, tue, wed, thu, fri, sat, sun};

int main(){
     enum week today;
     today = fri;
     printf("Day (integer format): %d\n", today + 1);
     switch(today){
          case(mon):
               printf("Day (string format): %s\n", "Monday");
               break;
          case(tue):
                    printf("Day (string format): %s\n", "Tuesday");
               break;
          case(wed):
               printf("Day (string format): %s\n", "Wednesday");
               break;

          . . .

          default:
               break;

     }
     return 0;
}
```

Structs

Structure

Definition

A structure is a collection of related variables (of possibly different types) grouped together under a single name

Structure

Definition

A structure is a collection of related variables (of possibly different types) grouped together under a single name.

Examples:

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

Structure

Definition

A structure is a collection of related variables (of possibly different types) grouped together under a single name.

Examples:

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

Notice ';' at the end

Structure

Definition

A structure is a collection of related variables (of possibly different types) grouped together under a single name.

Examples:

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

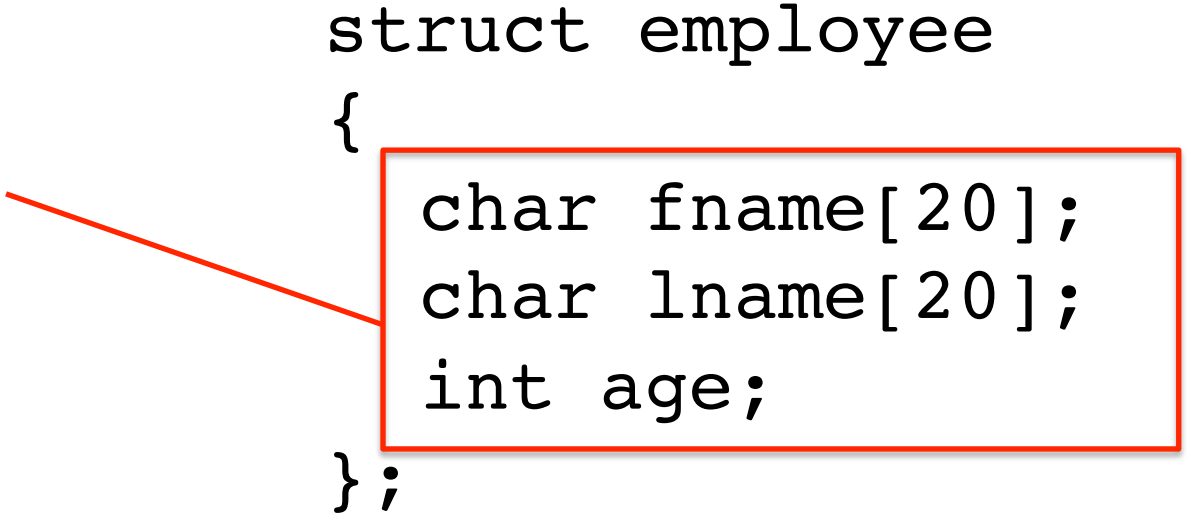
```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
```

Structure

Definition

A structure is a collection of related variables (of possibly different types) grouped together under a single name.

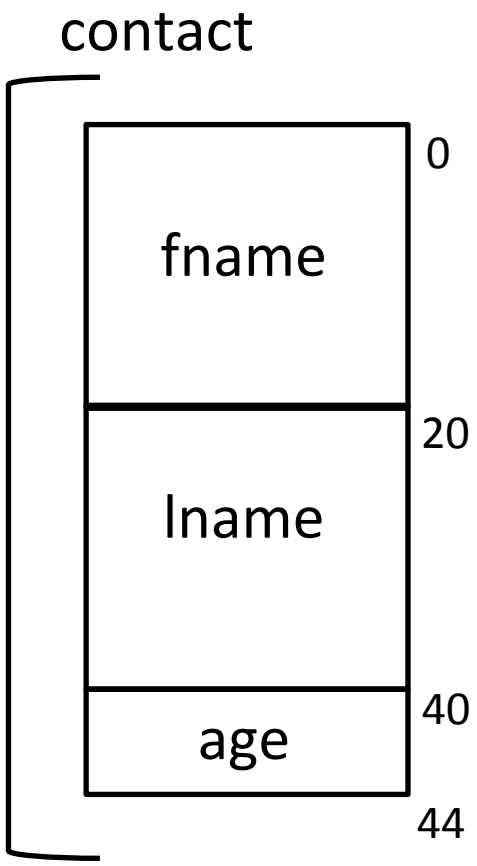
members pf different types



```
struct employee
{
    char fname[20];
    char lname[20];
    int age;
};
```

How is a struct represented in the RAM?

```
struct contact
{
   char fname[20];
   char lname[20];
   int age;
};
```



Structure - Characteristics

- **struct** defines a new datatype

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
```

Structure - Characteristics

- **struct** defines a new datatype

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
. . .
struct employee alice, bob;
```

…What happens if you use typedef

```
typedef struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
. . .
```

The use of typedef when defining a struct allows you to declare variables alice and bob without using ‘struct’

```
employee alice, bob;
```

Structure - Characteristics

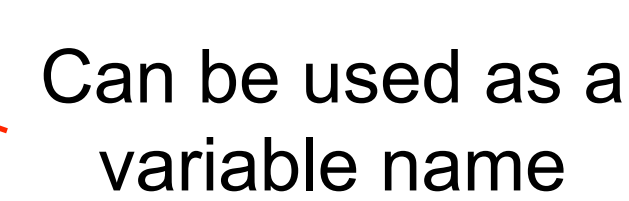
- **struct** can have a name… but it is optional

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
} alice;
```

Structure - Characteristics

- **struct** can have a name… but it is optional

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
} alice;
```



Structure - Characteristics

- **struct** can have a name… but it is optional

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
} alice;

. . .
printf("%s", alice.fname);
printf("%s", alice.lname);
```

Structure - Characteristics

- Initialization is done by specifying values of every member.

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
```

Structure - Characteristics

- Initialization is done by specifying values of every member.

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
struct employee alice={“alice”,”murphy”,19};
```

Structure - Characteristics

- Initialization is done by specifying values of every member.

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
struct employee alice={“alice”,”murphy”,19};
```

Assignment operator copies every member of the structure

Structure - Characteristics

- Individual members can be accessed using '.' operator

```
struct employee
{
   char fname[20];
   char lname[20];
   int age;
};
struct employee alice={“alice”,”murphy”,19};
. . .
printf(“Full name %s %s\n”, alice.fname, alice.lname);
printf(“Age: %d\n”, alice.age);
```

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct point{
    int x;
    int y;
};

struct line{
    struct point p1;
    struct point p2;
};

. . .
```

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct point{
    int x;
    int y;
};

struct line{
    struct point p1;
    struct point p2;
};

. . .
```

Members of a structure can be themselves structures (**nested structures**)

Example (Program line.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct point{
    int x;
    int y;
};

struct line{
    struct point p1;
    struct point p2;
};

. . .
```

```
int main(){
    struct line myLine;
    int  distX,distY;
    double segment;

    myLine.p1.x=10;
    myLine.p1.y=10;
    myLine.p2.x=50;
    myLine.p2.y=30;

    distX = abs(myLine.p1.x - myLine.p2.x);
    distY = abs(myLine.p1.y - myLine.p2.y);

    segment = pow(distX,2) + pow(distY,2);

    printf("The Line Segment is %.
2lf\n",segment);
}
```

Example (Program line.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct point{
    int x;
    int y;
};

struct line{
    struct point p1;
    struct point p2;
};

. . .
```

```
int main(){
    struct line myLine;
    int  distX,distY;
    double segment;

    myLine.p1.x=10;
    myLine.p1.y=10;
    myLine.p2.x=50;
    myLine.p2.y=30;

    distX = abs(myLine.p1.x - myLine.p2.x);
    distY = abs(myLine.p1.y - myLine.p2.y);

    segment = pow(distX,2) + pow(distY,2);

    printf("The Line Segment is %.
2lf\n",segment);
}
```

If structure is nested, multiple '.' are required

Arrays of Structures

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

int x[10];
struct point rectangle[4];
```

Arrays of Structures

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

int x[10];
struct point rectangle[4];

x={1,2,3,4,5,6,7,8,9,10};
rectangle = {0,5,6,5,2,3,4,6};
```

Arrays of Structures

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

int x[10];
struct point rectangle[4];

x={1,2,3,4,5,6,7,8,9,10};
rectangle = {0,5,6,5,2,3,4,6};
rectangle = {{0,5},{6,5},{2,3},{4,6}};
```

Arrays of Structures

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

int x[10];
struct point rectangle[4];

x={1,2,3,4,5,6,7,8,9,10};
rectangle = {0,5,6,5,2,3,4,6};
rectangle = {{0,5},{6,5},{2,3},{4,6}};
```

Equivalent ways to initialise variable rectangle

Example (Program rectangle.c)

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

int main(){
    struct point rectangle[4];

    for(int i=0; i<4;i++){
        printf("Insert 2 points:\n");
        scanf( "%d %d",&rectangle[i].x, &rectangle[i].y);
    }

    printf("Rectangle points\n");
    for(int i=0; i<4;i++){
        printf("Point %d {%d,%d}\n",
            i,rectangle[i].x, rectangle[i].y);
        }
}
```

Structure Pointers

Structure Pointers

- Passing structures by reference can sometimes be inefficient
- For large structures it is more efficient to pass pointers

Structure Pointers

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

void foo(point * pp){
   . . .
}

point pt;
```

Structure Pointers

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

void foo(point * pp){
   . . .
}

point pt;

foo(&pt);
```

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;
```

What is the value of p.x?

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

pp->x = 10;      /*Changes p.x */

int y = pp->y;
```

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

pp->x = 10;

int y = pp->y;
```

What is the value of y?

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

pp->x = 10;

int y = pp->y;  /* Same as y = p.y */
```

What is the value of y?

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

(*pp).x = 10;  /* Same as pp->x = 10 */

int y = (*pp).y;
```

What is the value of y?

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

(*pp).x = 10;  /* Same as pp->x = 10 */

int y = (*pp).y;  /* Same as int y = pp->y */
```

What is the value of y?

Example

- Members can be accessed from structure pointers using '->' operator.

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

point p={5,20};
point *pp = &p;

(*pp).x = 10;  /* Same as pp->x = 10 */

int y = (*pp).y;  /* Same as int y = pp->y */
```

What is the value of y?

To Recap

- Use a struct to define squares of the board
  - Note that they can contain a stack of game pieces

- Use a struct to define the game players. Each player should be characterized by:
  - Name
  - Color
  - Number of adversary pieces captured
  - Number of his/her own pieces, that can be placed on the board

- Use enums to specify the colour associated with each player and his/her pieces.