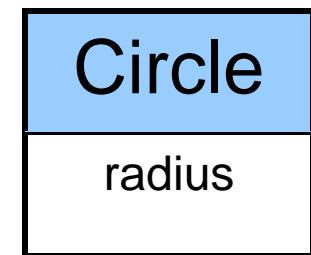
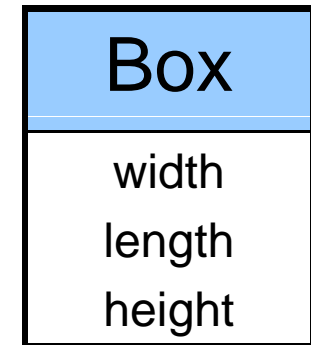


structs

Aggregating associated data
into a single variable

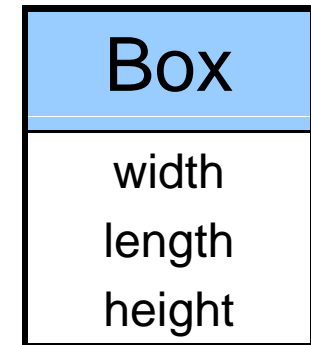
```
int main()
{
    Box mybox;
    Circle c;

    mybox.width = 10;
    mybox.length = 30;
    mybox.height = 10;
    c.radius = 10;
}
```



The idea

I want to describe a box. I need variables for the width, length, and height.



I can use three variables, but wouldn't it be better if I had a single variable to describe a box?

That variable can have three parts, the width, length, and height.

Structs

A `struct` (short for structure) in C is a grouping of variables together into a single type

- Similar to structs in Matlab

```
struct nameOfStruct
```

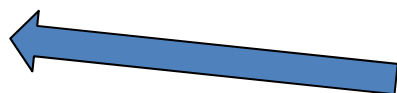
```
{
```

```
    type member;
```

```
    type member;
```

```
    ...
```

```
} ;
```

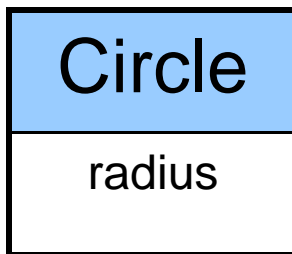
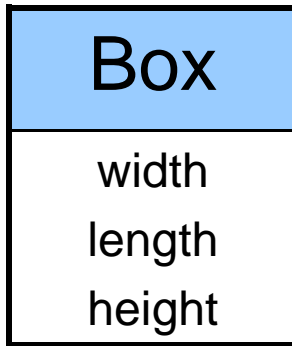


Note the semicolon at the end.

To declare a variable:

```
struct nameOfStruct variable_name;
```

Example



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

```
struct Box
```

```
{
```

```
    int width;
```

```
    int length;
```

```
    int height;
```

```
};
```

```
struct Circle
```

```
{
```

```
    double radius;
```

```
};
```

```
int main()
```

```
{
```

```
    struct Box b;
```

```
    struct Circle c;
```

```
}
```

**Data
structure
definition**

**You can
declare
variables**

Example

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

```
struct Box
```

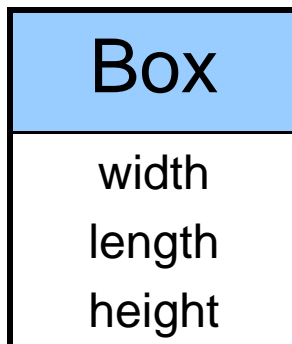
```
{
```

```
    int width;
```

```
    int length;
```

```
    int height;
```

```
};
```



```
int main() {
```

```
    struct Box b;
```

```
    b.width = 10;
```

```
    b.length = 30;
```

```
    b.height = 10;
```

```
}
```

**You can
assign values
to each
member**

We use a period “.” to get to the elements of a struct.

If x is a struct, x.width is an element in a struct.

Another Example

```
struct bankRecordStruct
```

```
{  
    char name[50];  
    float balance;  
};
```

You can use mixed data types within the struct (int, float, char [])

```
struct bankRecordStruct billsAcc;
```

Accessing values

```
struct bankRecordStruct
```

```
{  
    char name[50];  
    float balance;  
};
```

Access values in a
struct using a period:
“.”

```
struct bankRecordStruct billsAcc;
```

```
printf(“My balance is: %f\n”, billsAcc.balance);
```

```
float bal = billsAcc.balance;
```

Assign Values using Scanf()

```
struct BankRecord
{
    char name[50];
    float balance;
};

int main()
{
    struct BankRecord newAcc; /* create new bank record */

    printf("Enter account name: ");
    scanf("%50s", newAcc.name);
    printf("Enter account balance: ");
    scanf("%d", &newAcc.balance);
}
```


Copy via =

You can set two struct type variables equal to each other and each element will be copied

```
struct Box { int width, length, height; };

int main()
{
    struct Box b, c;
    b.width = 5; b.length=1; b.height = 2;
    c = b;          // copies all elements of b to c
    printf("%d %d %d\n", c.width, c.length, c.height);
}
```

Passing Struct to a function

- You can pass a struct to a function. All the elements are copied
- If an element is a pointer, the pointer is copied **but** **not** what it points to!

```
int myFunction(struct Person p)
{
...
}
```

Using Structs in Functions

Write a program that

- Prompts the user to enter the dimensions of a 3D box and a circle
- Prints the volume of the box and area of the circle

Sample run:

```
Enter the box dimensions (width,length,height): 1 2 3
Enter the radius of the circle: 0.8
```

```
Box volume = 6
Circle area = 2.01
```

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

struct Box { int width, height , length; };

int GetVolume(struct Box b)
{
    return b.width * b.height * b.length;
}

int main()
{
    struct Box b;

    printf("Enter the box dimensions (width length height): ");
    scanf("%d %d %d", &b.width, &b.length, &b.height);

    printf("Box volume = %d\n", GetVolume(b));
}
```

Note: == Comparison doesn't work

```
struct Box { int width, length, height; };
```

```
int main()
```

```
{
```

```
    struct Box b, c;
```

```
    b.width = 5; b.length=1; b.height = 2;
```

```
    c = b;
```

```
    if (c == b)      /* Error when you compile! */
```

```
        printf("c and b are identical\n");
```

```
    else
```

```
        printf("c and b are different\n");
```

```
} t
```

Error message: invalid operands to binary == (have 'Box' and 'Box')

Create your own equality test

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

```
struct Box { int width, height , length; };
```

```
int IsEqual(struct Box b, struct Box c)
{
    if (b.width==c.width &&
        b.length==c.length &&
        b.height==c.height)
        return 1;
    else
        return 0;
}
```

```
struct Box b, c;
b.width = 5; b.length=1; b.height = 2;
c = b;

if (IsEqual(b,c))
    printf("c and b are identical\n");
else
    printf("c and b are different\n");
```

typedef

typedef is a way in C to give a name to a custom type.

```
typedef type newname;
```

```
typedef int dollars;  
typedef unsigned char Byte;
```

I can declare variables like:

```
dollars d;  
Byte b, c;
```

It's as if the type already existed.

typedef for Arrays

There is a special syntax for arrays:

```
typedef char Names[40];  
typedef double Vector[4];  
typedef double Mat4x4[4][4];
```

Now, instead of:


```
double mat[4][4];
```

I can do:

```
Mat4x4 mat;
```


Using Structs with Typedef

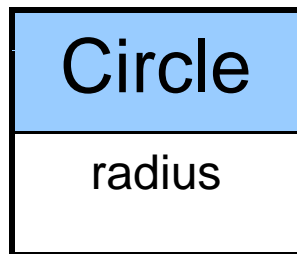
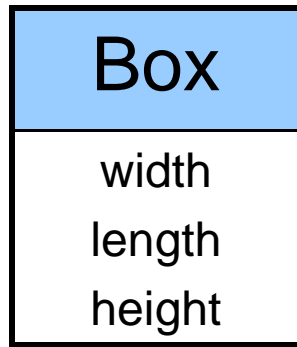
```
typedef struct [nameOfStruct]
{
    type member;
    type member;
    ...
} TypeName;
```



optional

To declare a variable: **TypeName** variable_name;

Example



```
#include <stdio.h>

typedef struct
{
    int width;
    int length;
    int height;
} Box;

typedef struct { double radius; } Circle;

int main()
{
    Box b; /* instead of struct Box */
    Circle c; /* instead of struct Circle */
    b.width = 10;
    b.length = 30;
    b.height = 10;
    c.radius = 10;
}
```

Arrays of structs

You can declare an array of a structure and manipulate each one

```
typedef struct
{
    double radius;
    int x;
    int y;
    char name[10];
} Circle;
```

```
Circle circles[5];
```

Size of a Struct: sizeof

```
typedef struct
```

```
{
```

```
    double radius;        /* 8 bytes */
```

```
    int x;                /* 4 bytes */
```

```
    int y;                /* 4 bytes */
```

```
    char name[10];        /* 10 bytes */
```

```
} Circle;
```

```
printf("Size of Circle struct is %d\n",  
      sizeof(Circle));
```

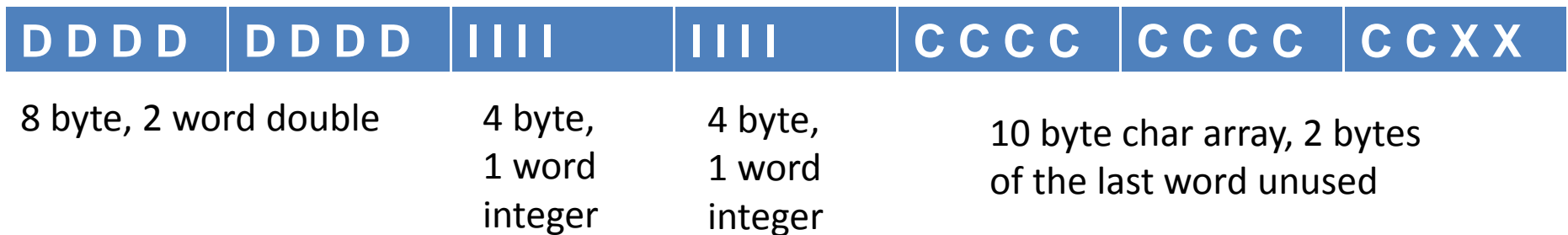
Size of a Struct

$$8 + 4 + 4 + 10 = 26$$

- But sizeof() reports 28 bytes!!!

Most machines require alignment on 4-byte boundary (a word)

- last word is not filled by the char (2 bytes used, 2 left over)



Pointers to structs

```
typedef struct  
{  
    int width;  
    int length;  
    int height;  
} Box;
```

```
Box b; /* A variable of type Box */  
Box *c; /* A pointer to a Box */  
double w;
```

```
b.width = 5; b.height = 7; b.length = 3;
```

```
c = &b; /* Same as before */
```

```
w = c->width;
```

To access the members of a struct, we use:

. for a variable of the struct's type
-> for a pointer to a struct

struct Concepts

```
struct Box
{
    double wid, hit;
};
```

```
typedef struct
{
    double radius;
    int x;
    int y;
    char name[10];
} Circle;
```

```
struct Box b;      /* No typedef */
Circle c;          /* typedef */

struct Box *pBox; /* Pointer to Box */
Circle *pCirc;    /* Pointer to Circle */

pBox = &b;         /* Get pointer to a Box */
b.wid = 3;
pBox->wid = 7;

pCirc = &c;
(*pCirc).radius = 9;
```

