CS1100 – Introduction to Programming

Instructor:

Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Lecture 29

How to store multiple related items?

Suppose you want to store information for 10 students. For each student you need to store, Roll Number, Name, Age, Program (BTech / DD / MTech)

How to store multiple related items?

Suppose you want to store information for 10 students. For each student you need to store, Roll Number, Name, Age, Program (BTech / DD / MTech)

- A possible way is to define 4 arrays – each of the appropriate type.

How to store multiple related items?

Suppose you want to store information for 10 students. For each student you need to store, Roll Number, Name, Age, Program (BTech / DD / MTech)

- A possible way is to define 4 arrays – each of the appropriate type.
- Arrays allow us to store multiple items but all of them need to be of the same type.

How to store multiple related items?

Suppose you want to store information for 10 students. For each student you need to store, Roll Number, Name, Age, Program (BTech / DD / MTech)

- A possible way is to define 4 arrays – each of the appropriate type.
- Arrays allow us to store multiple items but all of them need to be of the same type.
- Instead it would be good to have a way to store a collection of different types of data – related to one particular object (in this case student).

How to store multiple related items?

Suppose you want to store information for 10 students. For each student you need to store, Roll Number, Name, Age, Program (BTech / DD / MTech)

- A possible way is to define 4 arrays – each of the appropriate type.
- Arrays allow us to store multiple items but all of them need to be of the same type.
- Instead it would be good to have a way to store a collection of different types of data – related to one particular object (in this case student).
- Structures in C allow us to do the same.

What is a structure?

- Structures allow us to store variables of different data types together.
- Useful for logical organization even if all variables are of the same type.

What is a structure?

- Structures allow us to store variables of different data types together.
- Useful for logical organization even if all variables are of the same type.
  - Consider storing integer co-ordinates of n points in 2D.
  - Can be stored using an array of size $2n$.
  - But more logical to have x-coordinate in a separated from y-coordinate.

Defining a structure : Syntax

Defining a structure : Syntax

```
struct [structure tag]
   {
       member definition;
       member definition;
       ...
       member definition;
   };
```

Defining a structure : Syntax

```
struct [structure tag]
   {
       member definition;
       member definition;
       ...
       member definition;
   };
```

```
struct student {
       char rollNumber[6];
       char name[20];
       int age;
       int program;
};
```

Defining a structure : Syntax

```
struct [structure tag]
   {
       member definition;
       member definition;
       ...
       member definition;
   };
```

```
struct student {
       char rollNumber[6];
       char name[20];
       int age;
       int program;
};
```

- `struct student` is a new data-type.
- We can use struct student in the program just like a basic data type like int.
- `struct student s;` - defines a new variable s which is "type" `struct student`.
- Note the semicolon after the definition of the structure.

Using structures

Using structures

```
#include<stdio.h>
#include<string.h>
struct student {
     char rollNumber[6];
     char name[20];
     int age;
     int program;
};

struct student s;
```

Accessing values in a structure :
`name.member` gives you the value
stored in the member.
Eg : `s.name`

Using structures

```
#include<stdio.h>
#include<string.h>
struct student {
     char rollNumber[6];
     char name[20];
     int age;
     int program;
};
```

```
struct student s;
```

Accessing values in a structure :
`name.member` gives you the value stored in the member.
Eg : `s.name`

```
int main() {
    struct student S1;
    strcpy(S1.rollNumber, "CH17B005");
    strcpy(S1.name, "Mahendar");
    S1.age = 18;
    S1.program = 1;
    printf("Name: %s\n", S1.name);
    printf("Program: %d\n", S1.program)
}
```

Using structures

```
#include<stdio.h>
#include<string.h>
struct student {
     char rollNumber[6];
     char name[20];
     int age;
     int program;
};
```

```
struct student s;
```

Accessing values in a structure :
`name.member` gives you the value stored in the member.
Eg : `s.name`

```
int main() {
    struct student S1;
    strcpy(S1.rollNumber, "CH17B005");
    strcpy(S1.name, "Mahendar");
    S1.age = 18;
    S1.program = 1;
    printf("Name: %s\n", S1.name);
    printf("Program: %d\n", S1.program)
}
```

We can also initialize a structure by :

```
struct student S1 =
{"AE18B002","BAKUL",18,1};
```

Assigning a structure to another

```
#include<stdio.h>
#include<string.h>
struct student {
    char rollNumber[6];
    char name[20];
    int age;
    int program;
};
```

```
int main()
{
    struct student S1,S2;
    strcpy(S1.rollNumber, "CS15B1");
    strcpy(S1.name, "Ameet Deshpande");
    S1.age = 18;
    S1.program = 1;
    S2 = S1;
}
```

Assigning a structure to another

```
#include<stdio.h>
#include<string.h>
struct student {
    char rollNumber[6];
    char name[20];
    int age;
    int program;
};
```

```
int main()
{
    struct student S1,S2;
    strcpy(S1.rollNumber, "CS15B1");
    strcpy(S1.name, "Ameet Deshpande");
    S1.age = 18;
    S1.program = 1;
    S2 = S1;
}
```

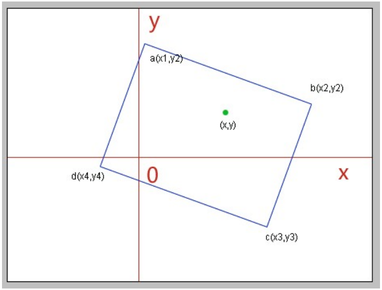
- Assigning one structure to another is supported.
- However checking for equality or not equal of two structures is not supported by the language. S1 == S2 is syntax error.

Using structures again

Given a rectangle and a point in 2D, determine if the point is inside the rectangle.

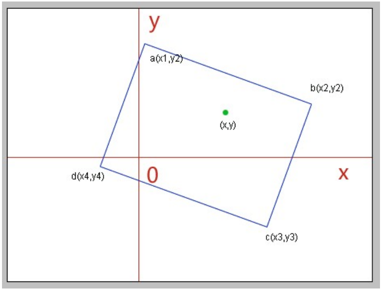
Using structures again

Given a rectangle and a point in 2D, determine if the point is inside the rectangle.



Using structures again

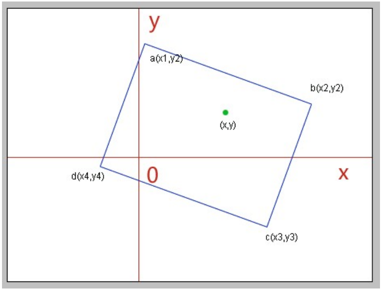
Given a rectangle and a point in 2D, determine if the point is inside the rectangle.



- Simplifying assumption : Assume rectangle is axis-parallel.

Using structures again

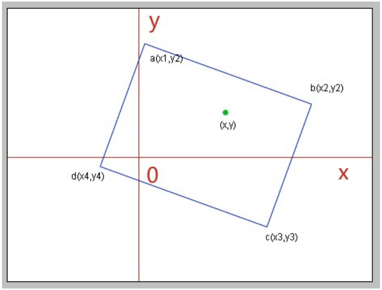
Given a rectangle and a point in 2D, determine if the point is inside the rectangle.



- Simplifying assumption : Assume rectangle is axis-parallel.
- How do we represent a point?

Using structures again

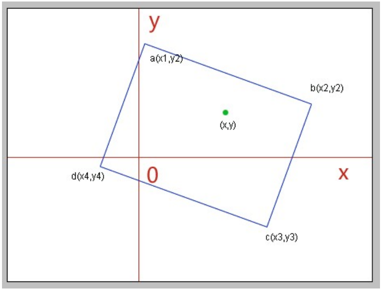
Given a rectangle and a point in 2D, determine if the point is inside the rectangle.



- Simplifying assumption : Assume rectangle is axis-parallel.
- How do we represent a point?
- How do we represent a rectangle?

Using structures again

Given a rectangle and a point in 2D, determine if the point is inside the rectangle.



- Simplifying assumption : Assume rectangle is axis-parallel.
- How do we represent a point?
- How do we represent a rectangle?
- Given a rectangle specified by the endpoints of a diagonal, how do we determine if a point lies inside the rectangle?

Define Appropriate Structures

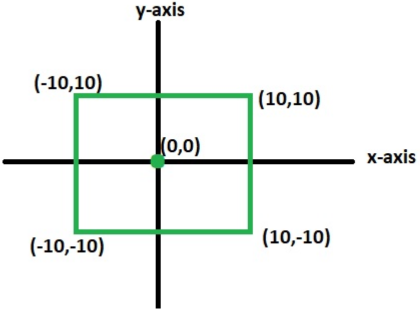
```
#include<stdio.h>
struct point {
     int xCoord;
     int yCoord;
};

struct rectangle {
     struct point lowerLeft;
     struct point upperRight;
};

int IsInside(struct rectangle, struct point);
```

Check whether point is inside

```
int IsInside(struct rectangle R, struct point P)
{
    // to be filled.
}
```

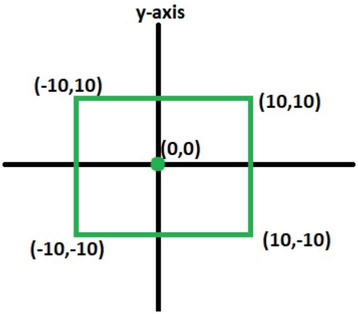
Check whether point is inside

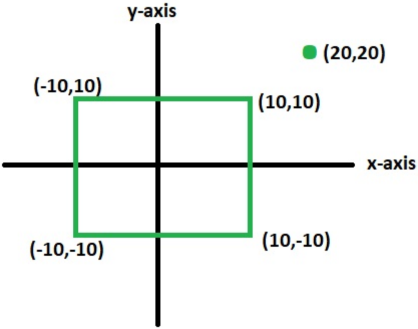
```
int IsInside(struct rectangle R, struct point P)
{
    // to be filled.
}
```



Check whether point is inside

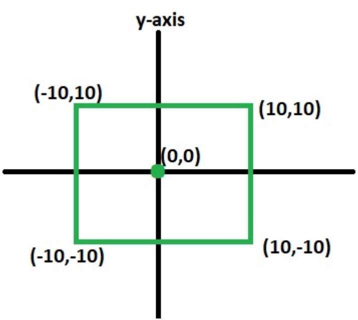
```
int IsInside(struct rectangle R, struct point P)
{
    // to be filled.
}
```

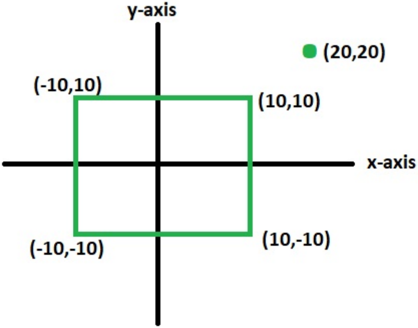




Check whether point is inside

```
int IsInside(struct rectangle R, struct point P)
{
    // to be filled.
}
```





Exercise : Complete the function `is_inside`.

Main Program : Scan the inputs & Invoke fn.

```
main() {
    struct rectangle R;
    struct point P;

    scanf("%d", &R.lowerLeft.xCoord);
    scanf("%d", &R.lowerLeft.yCoord);
    scanf("%d", &R.upperRight.xCoord);
    scanf("%d", &R.upperRight.yCoord);
    scanf("%d", &P.xCoord);
    scanf("%d", &P.yCoord);
    printf("%d\n", IsInside(R, P));
}
```

modularize the code further

- Write a function to get a point.
- Write a function to print a point.

modularize the code further

- Write a function to get a point.
- Write a function to print a point.

```
void get_point (struct point pt) {
     scanf("%d", &pt.xCoord);
     scanf("%d", &pt.yCoord);
}

void print_point (struct point pt) {
     printf("%d\t", pt.xCoord);
     printf("%d\n", pt.yCoord);
}
```

Corresponding main file

```
int main() {
    struct rectangle R;
    struct point P;

    GetPoint(R.lowerLeft);
    GetPoint(R.upperRight);
    GetPoint(P);

    printf("%d\n", IsInside(R, P));
    return 0;
}
```

Corresponding main file

```
int main() {
    struct rectangle R;
    struct point P;

    GetPoint(R.lowerLeft);
    GetPoint(R.upperRight);
    GetPoint(P);

    printf("%d\n", IsInside(R, P));
    return 0;
}
```

- Structures are passed by value.

Corresponding main file

```
int main() {
    struct rectangle R;
    struct point P;

    GetPoint(R.lowerLeft);
    GetPoint(R.upperRight);
    GetPoint(P);

    printf("%d\n", IsInside(R, P));
    return 0;
}
```

- Structures are passed by value. When the function is invoked - the structure `R.lowerLeft` is copied to the structure `pt`.

Corresponding main file

```
int main() {
    struct rectangle R;
    struct point P;

    GetPoint(R.lowerLeft);
    GetPoint(R.upperRight);
    GetPoint(P);

    printf("%d\n", IsInside(R, P));
    return 0;
}
```

- Structures are passed by value. When the function is invoked - the structure `R.lowerLeft` is copied to the structure `pt`.
- Changes made to contents of the structure are not visible outside the function.

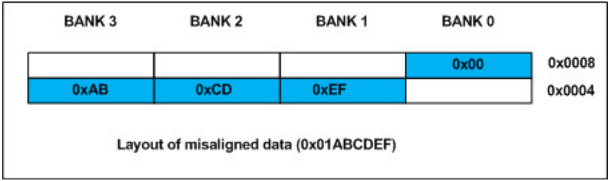
Corresponding main file

```
int main() {
    struct rectangle R;
    struct point P;

    GetPoint(R.lowerLeft);
    GetPoint(R.upperRight);
    GetPoint(P);

    printf("%d\n", IsInside(R, P));
    return 0;
}
```

- Structures are passed by value. When the function is invoked - the structure `R.lowerLeft` is copied to the structure `pt`.
- Changes made to contents of the structure are not visible outside the function. For that we need to pass by reference.

How are structures stored?

- When the structure is defined - no memory is allocated.
- Only when it is used to declare a structure variable - memory is allocated.
- Contiguous memory allocations are assigned but with some gap filler bytes to fix the memory alignment.

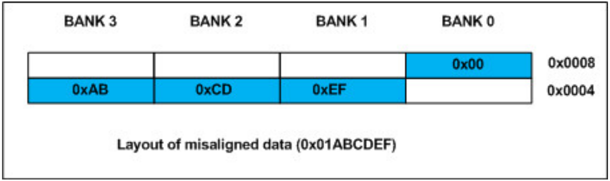


BANK 3	BANK 2	BANK 1	BANK 0	
			0x00	0x0008
0xAB	0xCD	0xEF		0x0004

Layout of misaligned data (0x01ABCDEF)

How are structures stored?

- When the structure is defined - no memory is allocated.
- Only when it is used to declare a structure variable - memory is allocated.
- Contiguous memory allocations are assigned but with some gap filler bytes to fix the memory alignment.



BANK 3	BANK 2	BANK 1	BANK 0	
			0x00	0x0008
0xAB	0xCD	0xEF		0x0004

Layout of misaligned data (0x01ABCDEF)

- The total size required to store a structure will depend on these alignments.

size of a structure

```
#include<stdio.h>
struct student {
   char rollNumber[6];
   char name[20];
   int age;
   int program;
};
int main() {
   printf("size of integer = %ld \n size = %ld\n",
  sizeof(int),sizeof(struct student));
}
```

size of a structure

```
#include<stdio.h>
struct student {
   char rollNumber[6];
   char name[20];
   int age;
   int program;
};
int main() {
   printf("size of integer = %ld \n size = %ld\n",
  sizeof(int),sizeof(struct student));
}
```

- What is the output of the program?
- Assume size of int is 4 bytes.
- Why does it print 36 instead of 34?