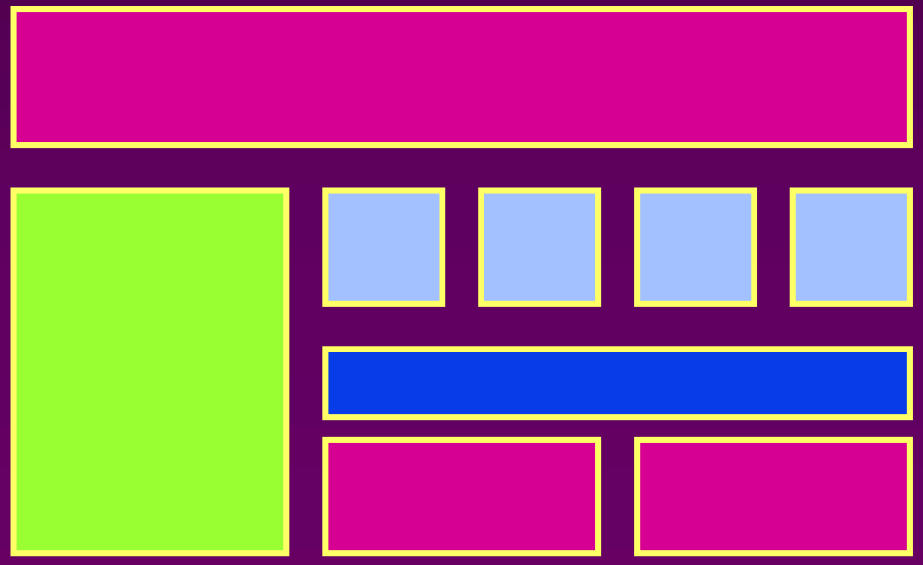
Programming



Structures

Structures

- A Structure is a collection of related data items, possibly of different types.
- A structure type in C is called struct.
- A struct is heterogeneous in that it can be composed of data of different types.
- In contrast, array is homogeneous since it can contain only data of the same type.

Structures

- Structures hold data that belong **together**.
- Examples:
  - Student record: student id, name, major, gender, start year, …
  - Bank account: account number, name, currency, balance, …
  - Address book: name, address, telephone number, …
- In database applications, structures are called records.

Structures

- Individual components of a struct type are called members (or fields).
- Members can be of different types (simple, array or struct).
- A struct is named as a whole while individual members are named using field identifiers.
- Complex data structures can be formed by defining arrays of structs.

struct basics

- Definition of a structure:

```
struct <struct-type>{
    <type> <identifier_list>;
    <type> <identifier_list>;
    ...
} ;
```

Each identifier defines a member of the structure.

- Example:

```
struct Date {
    int day;
    int month;
    int year;
} ;
```

The "Date" structure has 3 members, day, month & year.

struct examples

- Example:

```
struct StudentInfo{
    int Id;
    int age;
    char Gender;
    double CGA;
};
```

The “StudentInfo” structure has 4 members of different types.

- Example:

```
struct StudentGrade{
    char Name[15];
    char Course[9];
    int Lab[5];
    int Homework[3];
    int Exam[2];
};
```

The “StudentGrade” structure has 5 members of different array types.

struct examples

- Example:

```
struct BankAccount{
    char Name[15];
    int AcountNo[10];
    double balance;
    Date Birthday;
};
```

The “BankAcount” structure has simple, array and structure types as members.

- Example:

```
struct StudentRecord{
    char Name[15];
    int Id;
    char Dept[5];
    char Gender;
};
```

The “StudentRecord” structure has 4 members.

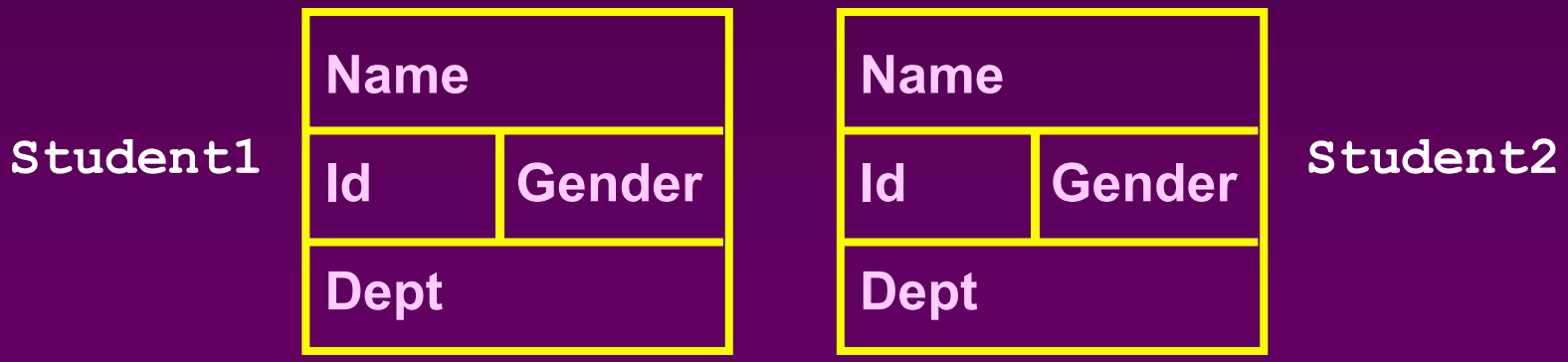
struct basics

- Declaration of a variable of struct type:

  `<struct-type> <identifier_list>;`

- Example:

```
StudentRecord Student1, Student2;
```



`Student1` and **`Student2`** are variables of **`StudentRecord`** type.

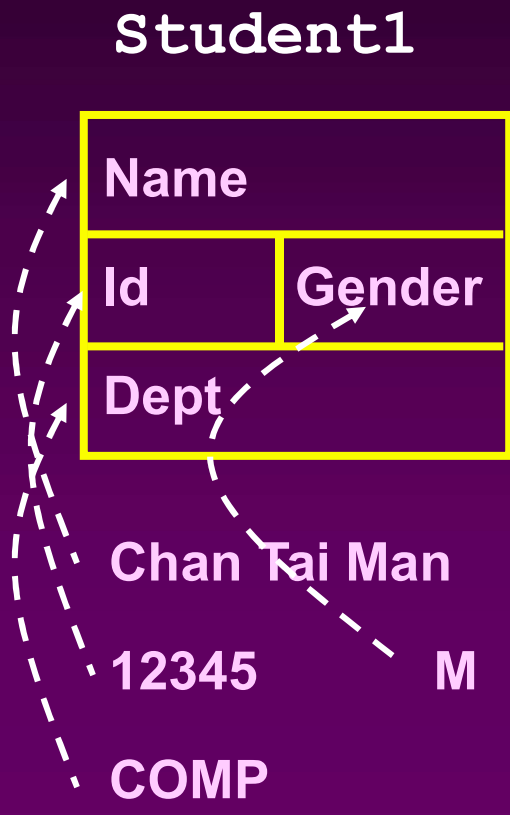
Ex. 1: struct basics

- The members of a struct type variable are accessed with the dot (.) operator:

```
<struct-variable>.<member_name>;
```

- Example:

```
strcpy(Student1.Name, "Chan Tai Man");
Student1.Id = 12345;
strcpy(Student1.Dept, "COMP");
Student1.gender = 'M';
printf("The student is ");
switch (Student1.gender){
    case 'F': printf("Ms."); break;
    case 'M': printf("Mr."); break;
}
printf("%s",Student1.Name);
```



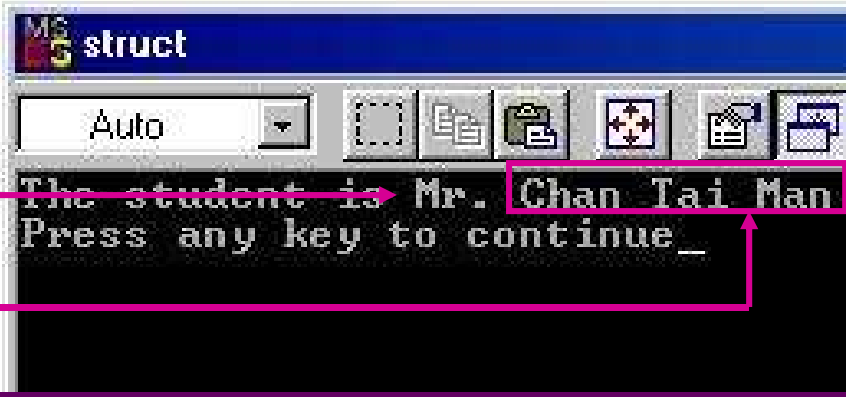
```
#include <string.h>

struct StudentRecord {
        char Name[22];
        int Id;
        char Dept[22];
        char gender;
};

int main()  {
    StudentRecord Student1;

    strcpy(Student1.Name, "Chan Tai Man");
    Student1.Id = 12345;
    strcpy(Student1.Dept, "COMP");
    Student1.gender = 'M';

    cout << "The student is ";
    switch (Student1.gender){
        case 'F': cout << "Ms. "; break;
        case 'M': cout << "Mr. "; break;
    }
    cout << Student1.Name << endl;
    return 0;
}
```

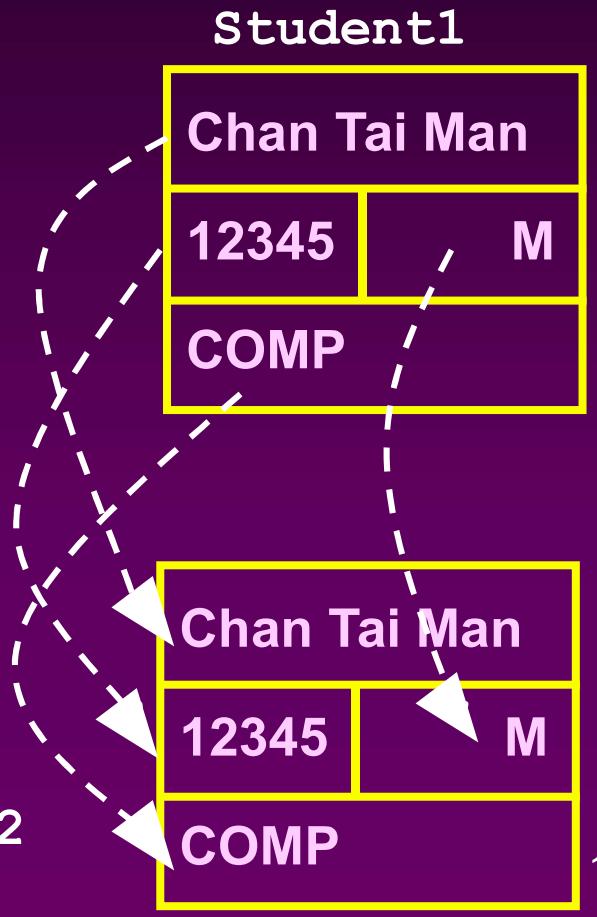


Ex. 2: struct-to-struct assignment

- The values contained in one struct type variable can be assigned to another variable of the same struct type.
- Example:

```
strcpy(Student1.Name,
       "Chan Tai Man");
Student1.Id = 12345;
strcpy(Student1.Dept, "COMP");
Student1.gender = 'M';

Student2 = Student1;
```



```
struct StudentRecord {
        char Name[22];
        int Id;
        char Dept[22];
        char gender;
};

int main()  {
    StudentRecord Student1, Student2;

    strcpy(Student1.Name, "Chan Tai Man");
    Student1.Id = 12345;
    strcpy(Student1.Dept, "COMP");
    Student1.gender = 'M';

    Student2 = Student1;
    Student2.gender = 'F';

    cout << "The student is ";
    switch (Student2.gender){
        case 'F': cout << "Ms. "; break;
        case 'M': cout << "Mr. "; break;
    }
    cout << Student2.Name << endl;
    return 0;
}
```

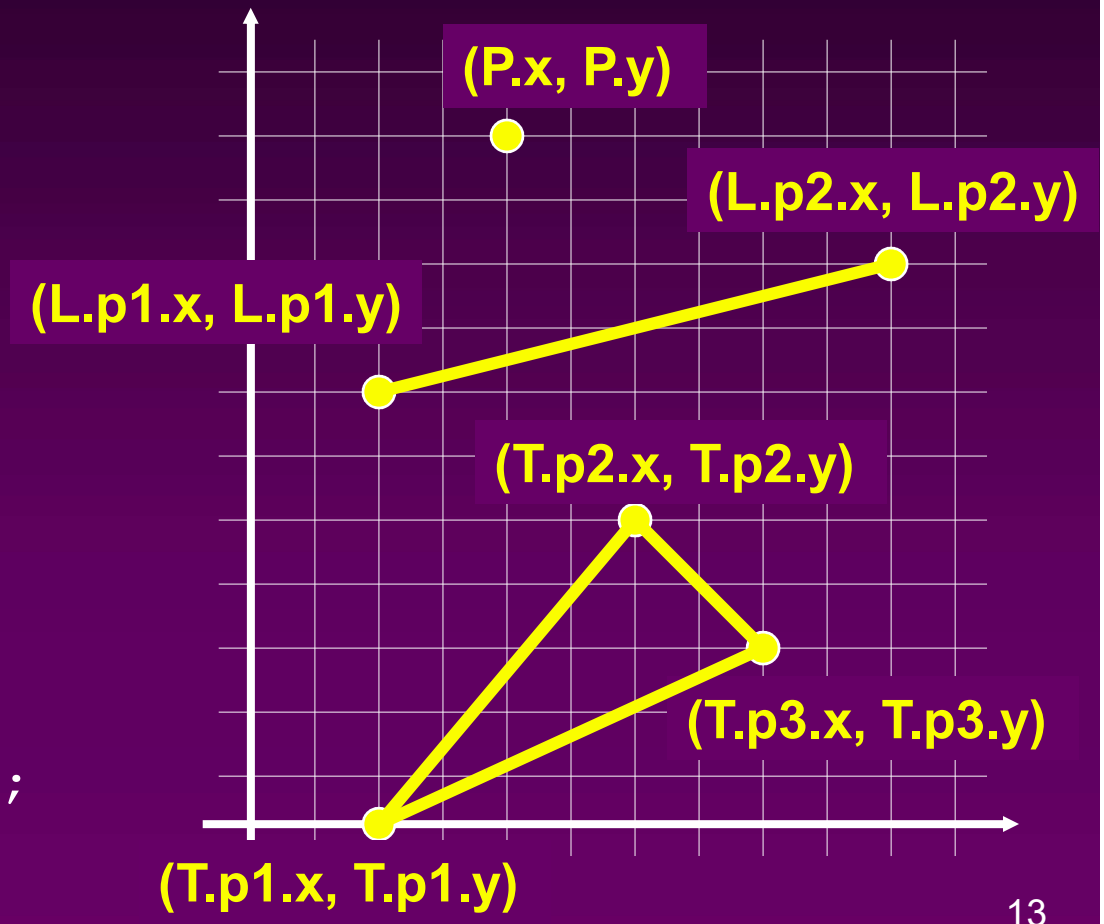
Ex. 3-5: Nested structures

- We can nest structures inside structures.
- Examples:

```
struct point{
    double x, y;
};
struct point P;

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

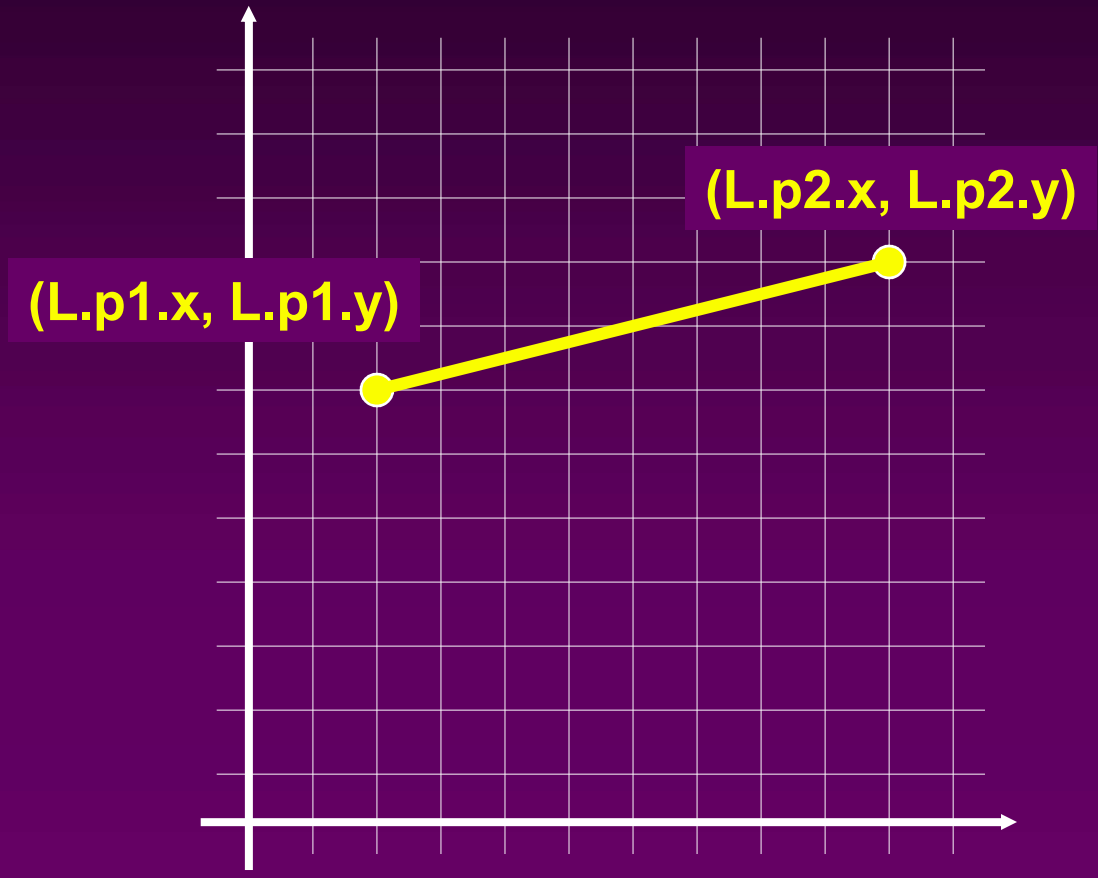
struct triangle{
    point p1, p2, p3;
};
struct triangle T;
```



Ex. 3-5: Nested structures

- We can nest structures inside structures.
- ```
  struct line{
      point p1, p2;
  };
  struct line L;
  ```

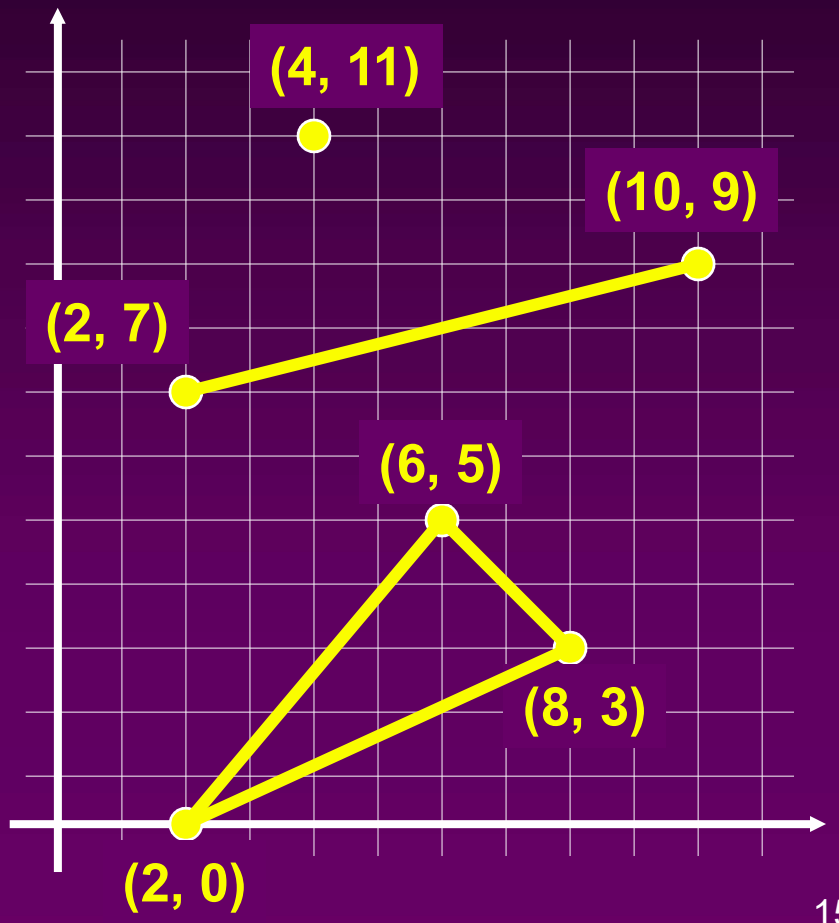
line	
p1	p2
x | y	x | y


  ```

Ex. 3-5: Nested structures

- Assign values to the variables `P`, `L`, and `T` using the picture:

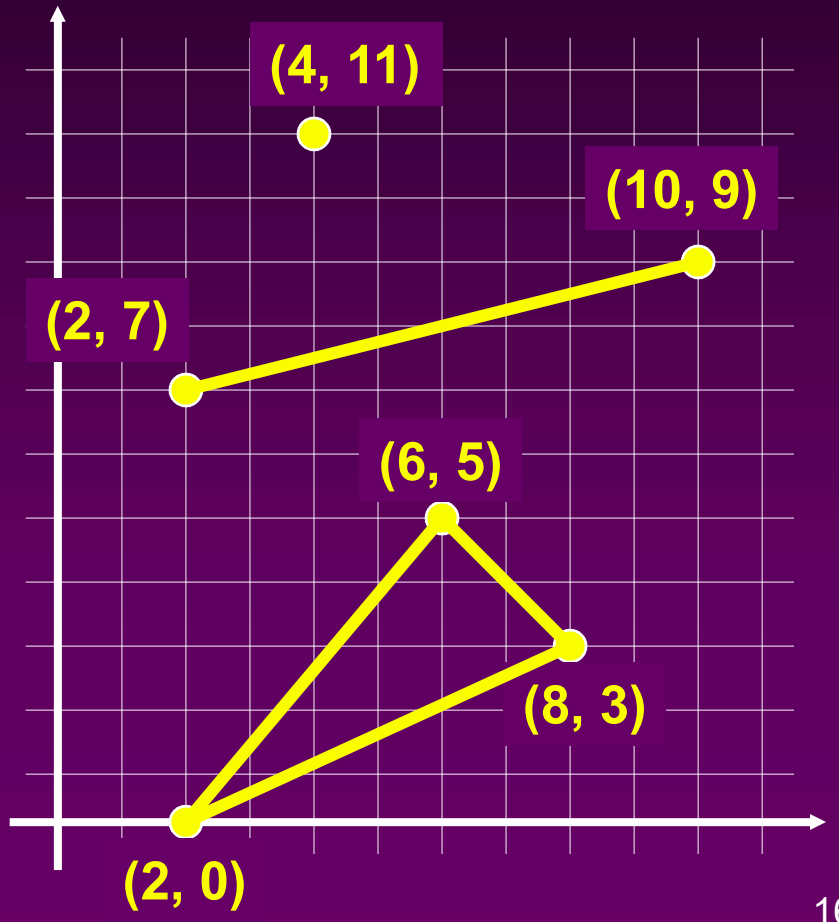
```
struct point P;
struct line L;
struct triangle T;
```



- Ex. 3: Graph a point
- Ex. 4: Graph a line
- Ex. 5: Graph a triangle

Ex. 3-5: Nested structures

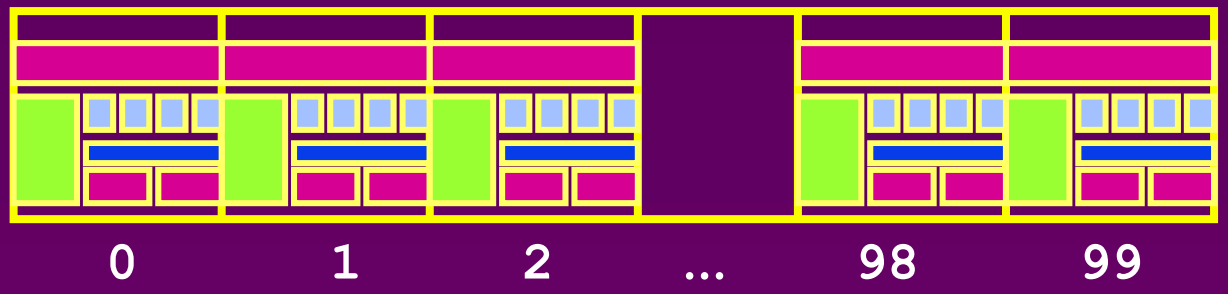
```
struct point P;
struct line L;
struct triangle T;
P.x = 4;
P.y = 11;
L.p1.x = 2;
L.p1.y = 7;
L.p2.x = 10;
L.p2.y = 9;
T.p1.x = 2;
T.p1.y = 0;
T.p2.x = 6;
T.p2.y = 5;
T.p3.x = 8;
T.p3.y = 3;
```



Arrays of structures

- An ordinary array: One type of data

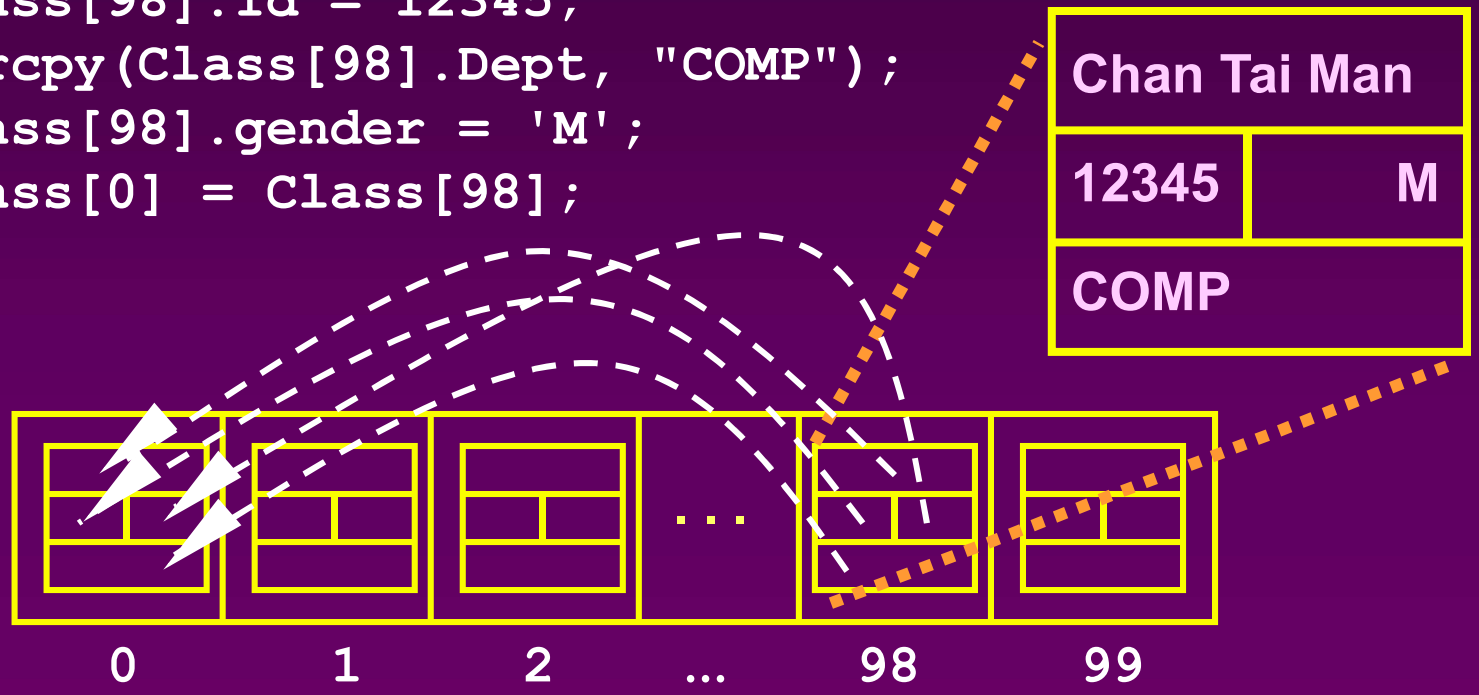
- An array of structs: Multiple types of data in each array element.



Arrays of structures

- We often use arrays of structures.
- Example:

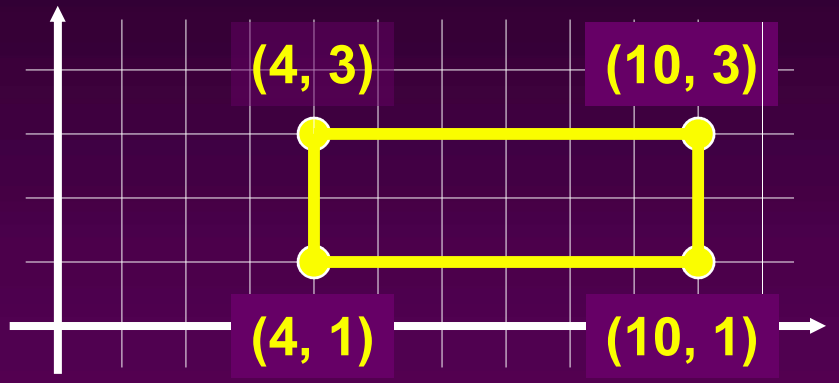
```
StudentRecord Class[100];
strcpy(Class[98].Name, "Chan Tai Man");
Class[98].Id = 12345;
strcpy(Class[98].Dept, "COMP");
Class[98].gender = 'M';
Class[0] = Class[98];
```



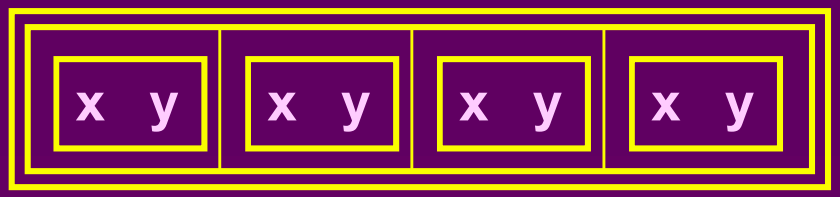
Arrays inside structures

- We can use arrays inside structures.
- Example:

```
struct square{
    point vertex[4];
};
square Sq;
```



- Assign values to **Sq** using the given square



Unions

- Like structures, but every member occupies the same region of memory!

```
union VALUE {
  float f;
  int i;
  char *s;
};
/* either a float xor an int xor a string */
```

Unions

- Up to programmer to determine how to interpret a union (i.e. which member to access)
- Often used in conjunction with a “type” variable that indicates how to interpret the union value

```
enum TYPE { INT, FLOAT, STRING };
struct VARIABLE {
  enum TYPE type;
  union VALUE value;
};
```

Unions

- Storage
  - size of union is the size of its largest member
  - avoid unions with widely varying member sizes;

    for the larger data types, consider using pointers instead
- Initialization
  - Union may only be initialized to a value appropriate for the type of its first member

Structs with Union (cont.)

```
#include <stdio.h>

union status
{
  int rank ;
  char deg[4] ;
} ;
```

```
struct personal
{
  long id ; float gpa ;
  union status level ;
} ;

struct identity
{
  char name[30] ;
  struct personal student ;
} ;
```

Structs with Union (cont.)

```
int main( )
{ struct identity jb = {"Joe Brown"}, *ptr = &jb;
   char u_g;
   jb.student.id = 123456789 ;
   jb.student.gpa = 3.4 ;
   printf ("Enter student status - u or g\n");
   scanf ("%c", &u_g);
   if (u_g == 'u')
     { printf ("Enter rank -- 1, 2, 3, 4 or 5\n");
       scanf ("%d", &jb.student.level.rank);
       printf ("%s is level %d\n" , jb.name ,
                  jb.student.level.rank);
     } /* End of if statement */
```

Structs with Union (cont.)

```
else
  {  printf ("Enter degree sought -- ms or phd\n");
     scanf ("%s", &jb.student.level.deg);
     printf ("%s is a %s candidate\n",
                  jb.name , jb.student.level.deg );
  } /* End of else statement */
  printf ("%s %ld %f\n" , jb.name , jb.student.id ,
              jb.student.gpa );
  printf ("%s%ld %f\n" , ptr->name , ptr->student.id ,
             ptr->student.gpa );
}  /*  End of program */
```

Enumeration

- Enumeration is a user-defined data type. It is defined using the keyword enum and the syntax is:

  enum tag_name {name_0, …, name_n};

- The tag_name is not used directly. The names in the braces are symbolic constants that take on integer values from zero through n. As an example, the statement:

  enum colors { red, yellow, green };

- creates three constants. red is assigned the value 0, yellow is assigned 1 and green is assigned 2.

Enumeration

```
/* This program uses enumerated data types to
   access the elements of an array */
#include <stdio.h>
int main( )
{
   int March[5][7]={{0,0,1,2,3,4,5},{6,7,8,9,10,11,12},
   {13,14,15,16,17,18,19},{20,21,22,23,24,25,26},
   {27,28,29,30,31,0,0}};
   enum days {Sunday, Monday, Tuesday,
        Wednesday, Thursday, Friday, Saturday};
```

Enumeration

```
    enum week {week_one, week_two, week_three,
                        week_four, week_five};

    printf ("Monday of the third week of March is %d\n",
    March [week_three] [Monday] );
}
```