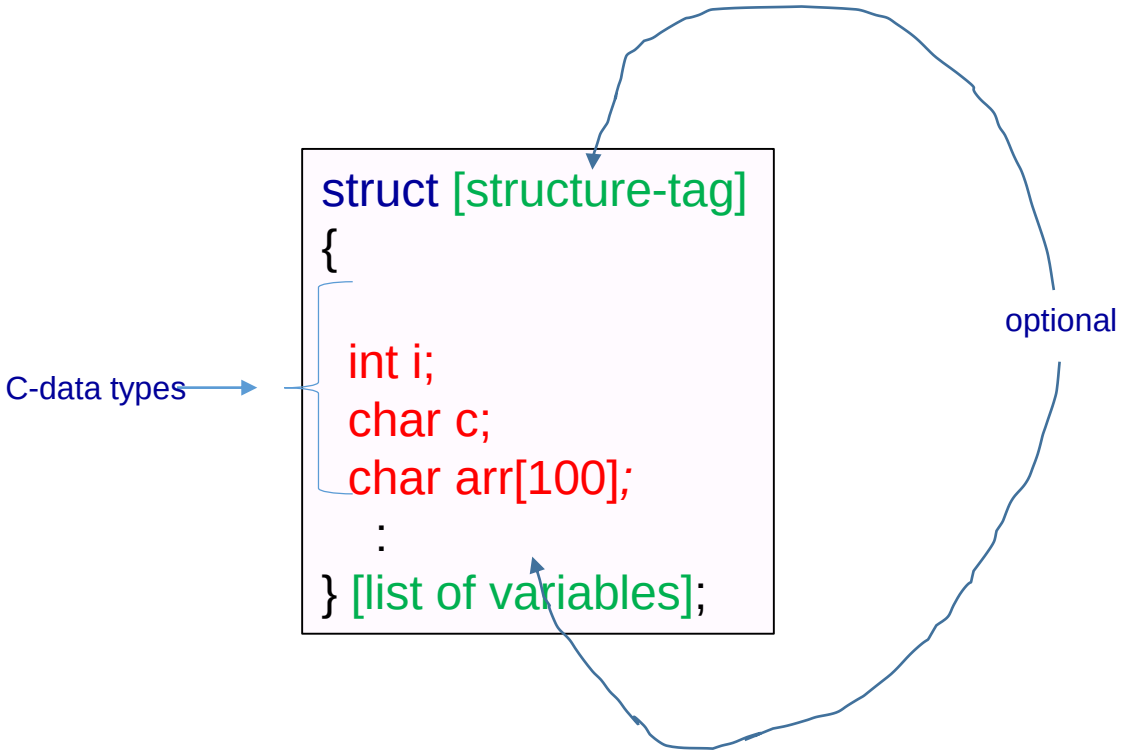
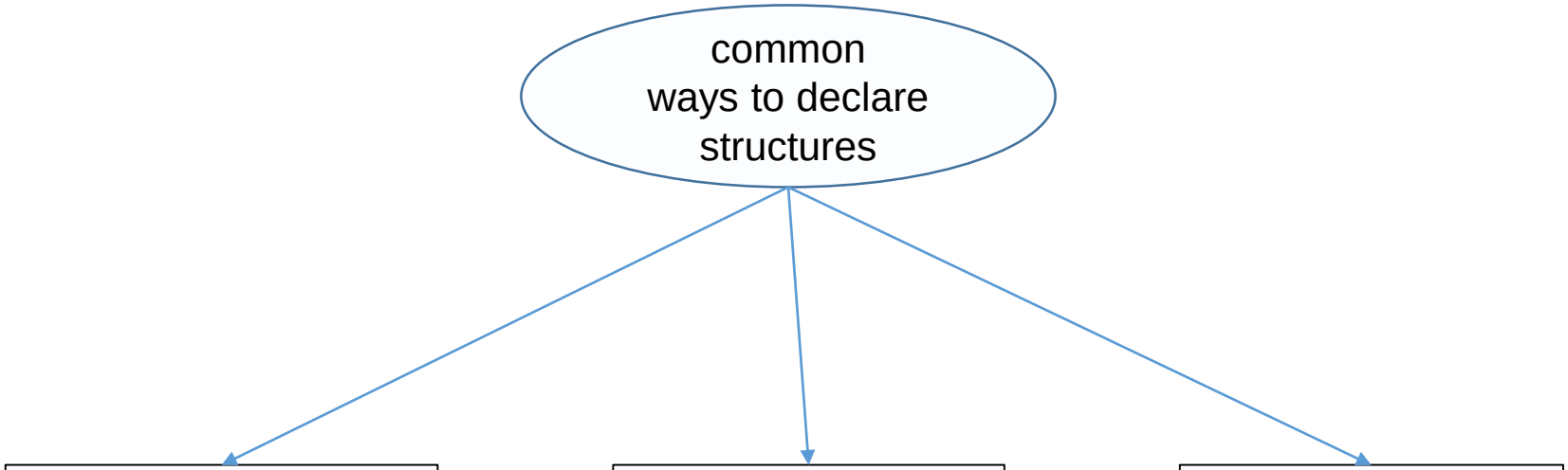
C - Structures

- A structure is a collection of members that can be of different data types.
- A structure is a user defined data type.
- The **struct** keyword is used to define a structure.
- While an array holds several items of the same type, a structure allows to define data type that combine data items of different types.



Structure Declaration




```
struct Student {
 int id;
 char name[10];
};

struct Student s1, s2;
```

- Declaring a tag name and then using it to declare the actual variables.
- s1 and s2 … are variables of Student type.

```
struct  {
    int id;
    char name[10];
} s1, s2;
```

- Declaring a structure without a tag name.
- useful if the structure is used only in one place.
- s1 and s2 … are variables with no type name.

```
struct Student
{
    int id;
    char name[10];
} s1, s2;
```

- Declaring a structure with a tag name and variables
- s1 and s2 … are variables of Student type.

Structure Initialization and Assignment

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

struct Student
{
    int id;
    char name[10];
};

int main()  {

 struct Student s;

 s.id = 123;
 strcpy(s.name,"Avi");

 printf("%d %s\n",s.id ,
s.name);

 return 0;
}
```

assignment

Output:
123 Avi

```
#include <stdio.h>

struct Student
{
    int id;
    char name[10];
};

int main()  {

 struct Student s = {123,"Avi"};

 printf("%d %s\n", s.id ,
s.name);

 return 0;
}
```

Output:
123 Avi

initialization

```
#include <stdio.h>

struct Student
{
    int id;
    char name[10];
} s = {123,"Avi"};

int main()  {

 printf("%d %s\n", s.id ,
s.name);

 return 0;
}
```

Output:
123 Avi

```
#include <stdio.h>

struct Student
{
    int id;
    char name[10];
}
s1 = {1,"Avi"},
s2 = {2,"Dado"};

int main()  {

 printf("%d %s\n",s1.id , s1.name);
 printf("%d %s\n",s2.id , s2.name);

 return 0;
}
```

Output:
1 Avi
2 Dado

un-initialized local structure variables are not initialized with default values.

```
#include <stdio.h>

struct test
{
    char c;
    int i;
    double d;
    char *p;
    char name[10];
};

int main()  {
 struct test s;
 printf("%d %d %f %p %d\n", s.c, s.i, s.d, s.p,s.name[0]);

 return 0;
}
```

Output:
-47 -1 -0.000000 0xb7629235 112

← garbage values

un-initialized global/static structure variables are automatically initialized with default values.

```
#include <stdio.h>

struct test
{
    char c;
    int i;
    double d;
    char *p;
    char name[10];
};

struct test s;

int main()  {

 printf("%d %d %f %p %d\n", s.c, s.i, s.d, s.p,s.name[0]);

 return 0;
}
```

Output:
0 0 0.000000 (nil) 0

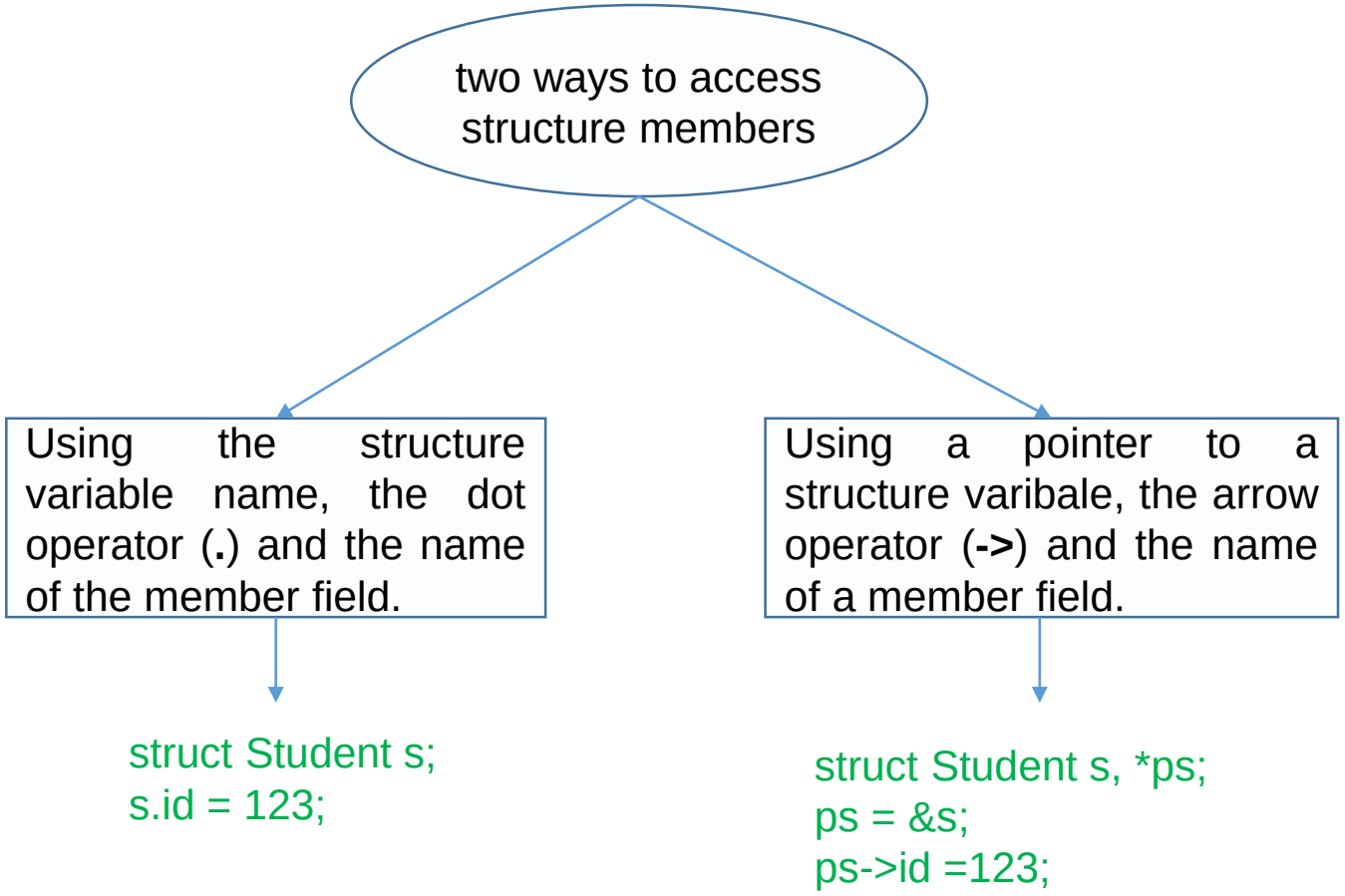
Default values:
- ✓ 0 for integers and floating point
- ✓ '\0' for char (of course this is just the same as 0, and char is an integer type)
- ✓ NULL for pointers.

```c
#include <stdio.h>

struct test
{
    char c;
    int i;
    double d;
    char *p;
    char name[10];
};

int main()  {
 struct test s = {12};
 printf("%d %d %f %p %d\n", s.c, s.i, s.d, s.p, s.name[0]);

 return 0;
}
```

Output:
12 0 0.000000 (nil) 0

Once one member is initialized, then all remaining members are automatically initialized with the default values.

Accessing Structure Members



Structure Definition
And Members Accessing

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

struct Person
{
    char name[10];
    char fname[10];
    int id;
};

int main()  {
    struct Person p1;

    strcpy(p1.name, "Alex");
    strcpy(p1.fname, "Dado");
    p1.id = 123;

    printf("%s %s %d\n", p1.name , p1.fname , p1.id);

    return 0;
}
```

struct keyword **is used** to define variables of structure type.

Output:
Alex Dado 123

Pointers to Structures

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

struct Person
{
    char name[10];
    char fname[10];
    int id;
};

int main() {
    struct Person p1;
    struct Person *ptr;

    ptr = &p1;

    strcpy(ptr->name , "Alex");
    strcpy(ptr->fname , "Dado");
    ptr->id = 123;

    printf("%s %s %d\n",ptr->name,ptr->fname,ptr->id);

    return 0;
}
```

Output:
Alex Dado 123

Size of Structures

the size (**without compiler's data structure padding**) of a structure is the aggregated sum of its members..

10 bytes

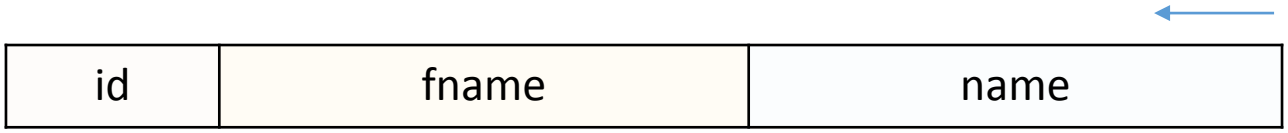
10 bytes

4 bytes

```
#include <stdio.h>
struct Person
{
    char name[10];
    char fname[10];
    int id;
};

int main()  {
    struct Person p1;
    printf("%d\n",sizeof(struct Person));
    printf("%d\n",sizeof(p1));
    return 0;
}
```

The size of a structure can be calculated using the sizeof() operator. The argument of operator sizeof() can be the struct type or the variable of struct type

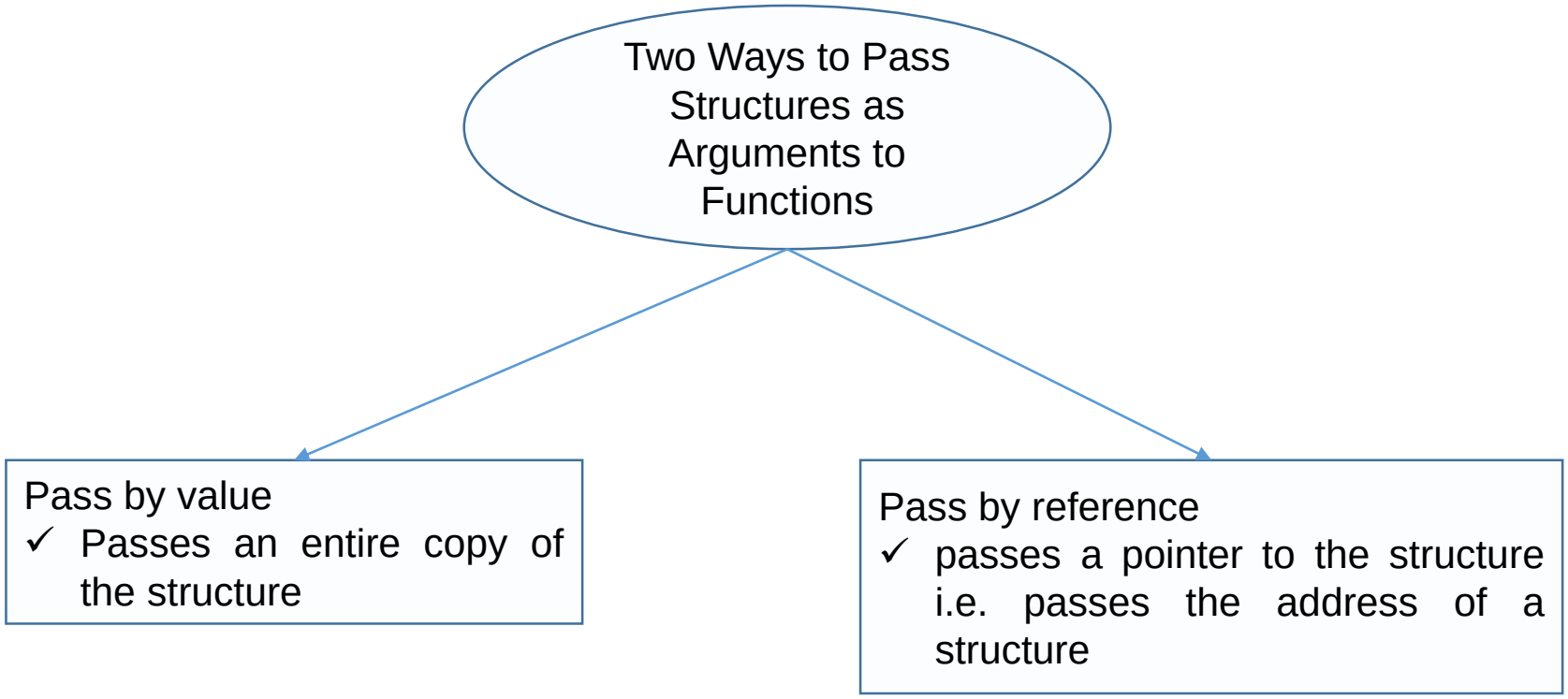
Output :
24
24



Bytes 4 10 10

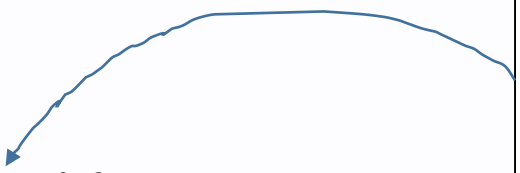
in C the address of a struct is the same as the address of its first member

Structures And Function Arguments



Pass by value

```c
#include <stdio.h>
#include <string.h>
struct Person
{
    char name[10];
    char fname[10];
    int id;
};

void whoAmI(struct Person tmp) {
  printf("%d\n", sizeof(tmp));
  printf("%s %s %d\n", tmp.name, tmp.fname , tmp.id);
}

int main() {
  struct Person p1;

  strcpy(p1.name , "Alex");
  strcpy(p1.fname, "Dado");
  p1.id = 123;

  whoAmI(p1);
  return 0;
}
```

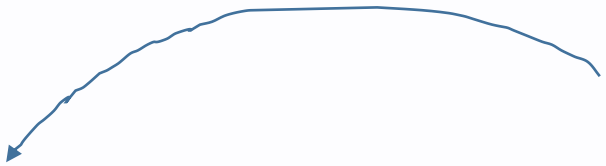


- ✓ a copy of p1.
- ✓ tmp is local variable in the function.
- ✓ any change on tmp will not impact the origin variable p1.
- ✓ it's size is 24.

Output:
24
Alex Dado
123

Pass by reference

```
#include <stdio.h>
#include <string.h>
struct Person
{
    char name[10];
    char fname[10];
    int id;
};

void whoAmI(struct Person* ptr)
{
    printf("%d\n", sizeof(ptr));
    printf("%s %s %d\n", ptr->name, ptr->fname, ptr->id);
}

int main() {
    struct Person p1;

    strcpy(p1.name, "Alex");
    strcpy(p1.fname, "Dado");
    p1.id = 123;

    whoAmI(&p1);
    return 0;
}
```



- ✓ a pointer to p1.
- ✓ it's size is the size of a pointer → 4.
- ✓ any change through ptr will impact p1.

Output:
4
Alex Dado
123

Array of Structures

```c
#include <stdio.h>
struct Student
{
    int id;
    char name[10];
};
int main()
{
    struct Student arr[] =
    {
        {1,"Alex"},   {2,"Avi"},  {3,"David"}
    };
    int i,n;

    n=sizeof(arr)/sizeof(struct Student);

    for(i=0; i<n; i++)
        printf("%d %s\n", arr[i].id, arr[i].name);

    return 0;
}
```

Output:
1 Alex
2 Avi
3 David

```c
#include <stdio.h>
#define EOD -1
struct Student
{
    int id;
    char name[10];
};
int main()
{
    struct Student arr[] =
    {
        {1,"Alex"}, {2,"Avi"}, {3,"David"},{EOD,""}
    };

    struct Student *pArr = arr;

    while(pArr->id != EOD) {
        printf("%d %s\n",pArr->id, pArr->name);
        pArr++;
    }
    return 0;
}
```

Output:
1 Alex
2 Avi
3 David

Array of structure as argument to a function

```c
#include <stdio.h>
#define SIZE 3
struct Course
{
    char name[12];
    float score;
};
void showAll(struct Course a[ ]) {
    int i;
    for(i=0; i<SIZE; i++)
        printf("%s %f\n",a[i].name, a[i].score);
}
int main() {
    struct Course arr[SIZE] =
    {
        {"Java",85},   {"C",100},  {"History",90}
    };
    showAll(arr);
    return 0;
}
```

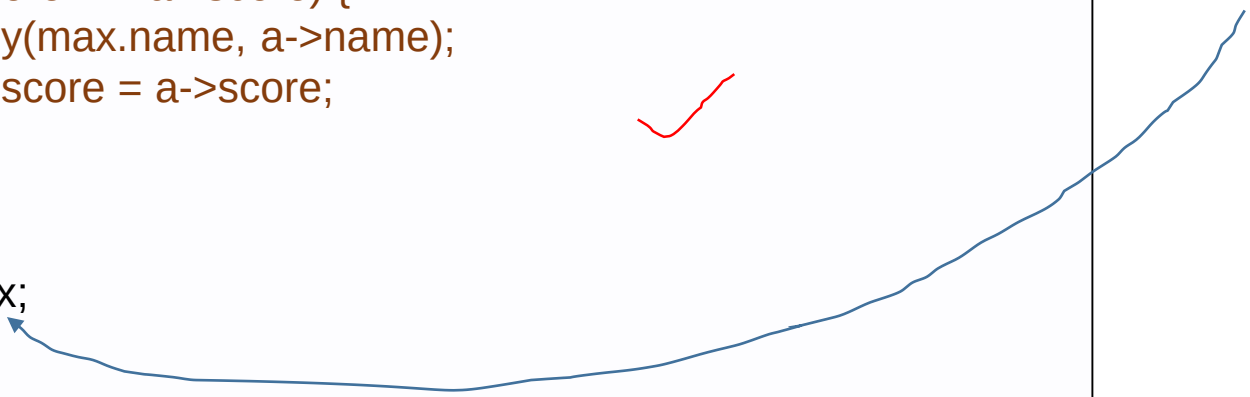
Output:
Java 85.000000
C 100.000000
History 90.000000

```c
#include <stdio.h>
#define EOD -1
struct Course
{
    char name[12];
    float score;
};
void showAll(struct Course *pa) {
    while(pa->score != EOD) {
        printf("%s %f\n",pa->name,pa->score);
        pa++;
    }
}
int main() {
 struct Course arr[] =
 {
    {"Java",85},  {"C",100}, {"History",90}, {"",EOD}
 };
 showAll(arr);
 return 0;
}
```

Output:
Java 85.000000
C 100.000000
History 90.000000

Function returning a Structure

```
#include <stdio.h>
#include <string.h>
#define EOD -1
struct Course
{
    char name[12];
    float score;
};
struct Course getCourseOfMaxScore(struct Course a[]) {
    struct Course max;
    strcpy(max.name, a[0].name);   max.score = a[0].score;
    while(a->score != EOD) {
        if(a->score > max.score) {
            strcpy(max.name, a->name);
            max.score = a->score;
        }
        a++;
    }
    return max;
}
int main() {
 struct Course arr[] = { {"Java",85},   {"C",100},  {"History",90},{"",-1} };
 struct Course result = getCourseOfMaxScore(arr);
 printf("%s %f\n", result.name,result.score);
 return 0;
}
```



- ✓ It is possible to return a structure from a function.
- ✓ It is similar as returning int,char...
- ✓ A copy of a local variable will be returned and then the local variable will be destroyed. here 'max' will disappear after leaving the function.

Output:
C 100.000000

Function Returning a Pointer to a Structure

```
#include <stdio.h>
#include <string.h>
struct Course
{
    char name[12];
    float score;
};

struct Course* createCourse(char*s, float f)
{
    struct Course tmp;
    strcpy(tmp.name,s);
    tmp.score=f;
    return &tmp;
}

int main()
{
    struct Course* p = createCourse("C",100);
    printf("%s %f\n", p->name,p->score);
    return 0;
}
```

Function cannot return a pointer to a local variable

It is legal if a function returns a point to a global or static variable, or dynamically allocated memory…

```c
#include <stdio.h>
#include <string.h>
struct Course
{
    char name[12];
    float score;
};

struct Course* createCourse(char*s, float f)
{
    static struct Course tmp;
    strcpy(tmp.name,s);
    tmp.score=f;
    return &tmp;
}

int main()
{
    struct Course* p = createCourse("C",100);
    printf("%s %f\n", p->name,p->score);
    return 0;
}
```

Even we can return a pointer to a static variable, is it really what we want? In below example tmp is created once and each time we enter the function we overwrite it!

```c
#include <stdio.h>
#include <string.h>
struct Course
{
    char name[12];
    float score;
} glob;

struct Course* createCourse(char*s, float f)
{
  strcpy(glob.name,s);
  glob.score=f;
  return &glob;
}

int main() {
    struct Course* p = createCourse("C",100);
    printf("%s %f\n", p->name,p->score);
    return 0;
}
```

Note: better to avoid using global variables if possible!

Dynamic Allocation of Memory for Structures

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct Course
{
    char name[12];
    float score;
};

struct Course* createCourse(char*s, float f)
{
    struct Course *p;
    p = (struct Course*)malloc( sizeof(struct Course) );
    strcpy(p->name,s);
    p->score=f;
    return p;
}

int main()
{
    struct Course* p = createCourse("C",100);
    printf("%s %f\n", p->name,p->score);
    free(p);
    return 1;
}
```

p is a pointer to object of type Course.

Never forget to free the allocated memory!

Nested Structures

version1

```
#include <stdio.h>
struct Birthday {
   short day, month, year;
};

struct Person {
 int id;
 char name[12];
 char *fn;
 struct Birthday birthday;
};

int main()
{
   struct Person p = {1,"David","Cohen",{22,5,1975}};
   printf("%d %s %s\n",p.id, p.name, p.fn);
   printf("%d %d %d\n",p.birthday.day, p.birthday.month, p.birthday.year);
   return 0;
}
```

Output:
1 David Cohen
22 5 1975

version2

```
#include <stdio.h>
#include <stdlib.h>
struct Birthday {
   short day, month, year;
};

struct Person {
 int id;
 char name[12];
 char *fn;
 struct Birthday* pb;
};

int main()
{
   struct Birthday b = {22,5,1975};
   struct Person p = {1,"David","Cohen"};
   p.pb = &b;
   printf("%d %s %s\n",p.id, p.name, p.fn);
   printf("%d %d %d\n",p.pb->day, p.pb->month, p.pb->year);
   return 0;
}
```

Output:
1 David Cohen
22 5 1975

version3

```c
#include <stdio.h>
#include <stdlib.h>
struct Birthday {
   short day, month, year;
};

struct Person {
 int id;
 char name[12];
 char *fn;
 struct Birthday* pb;
};

int main()
{
   struct Person p = {1,"David","Cohen"};
   p.pb = (struct Birthday*)malloc(sizeof(struct Birthday));
   p.pb->day = 22;
   p.pb->month = 5;
   p.pb->year = 1975;
   printf("%d %s %s\n",p.id,p.name,p.fn);
   printf("%d %d %d\n",p.pb->day,p.pb->month,p.pb->year);
   free(p.pb);
   return 0;
}
```

Output:
1 David Cohen
22 5 1975

Nested Structure Array

```
#include <stdio.h>
struct Course {
    char name[12];
    float score;
};

struct Student {
    int id;
    char name[12];
    struct Course a[3];
};

int main() {
    struct Student s =
    {
        1,
        "David",
        {{"java",85},{"C",100},{"History",90}}
    };
    printf("%d %s %s %f\n",s.id, s.name, s.a[0].name, s.a[0].score);
    printf("%d %s %s %f\n",s.id, s.name, s.a[1].name, s.a[1].score);
    printf("%d %s %s %f\n",s.id, s.name, s.a[2].name, s.a[2].score);
    return 0;
}
```

Array of Courses

sizeof(struct Student) → 64

Output:
1 David java 85.000000
1 David C 100.000000
1 David History 90.000000

version2

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Course {
    char name[12];
    float score;
};
struct Student {
    int id;
    char name[12];
    struct Course* a[3];
};
int main() {
    struct Student s =  {1,"David",{NULL,NULL,NULL}};

    struct Course c1 = {"Java",90};
    s.a[0] = &c1;

    s.a[1] = (struct Course*)malloc(sizeof(struct Course));
    strcpy(s.a[1]->name,"C");
    s.a[1]->score=100;

    printf("%d %s %s %f\n",s.id,s.name,s.a[0]->name,s.a[0]->score);
    printf("%d %s %s %f\n",s.id,s.name,s.a[1]->name,s.a[1]->score);
    free(s.a[1]);
    return 0;
}
```

Array of pointers to Courses

sizeof(struct Student) → 28

Output:
1 David java 90.000000
1 David C 100.000000

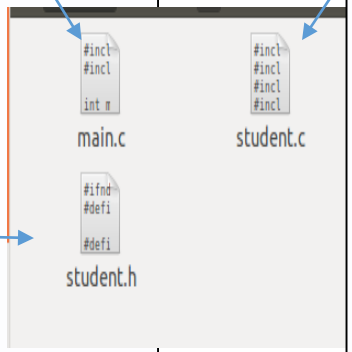
Exercise1: write the student.c according to the giving main.c and student.h files

main.c

```
#include <stdio.h>
#include "student.h"
int main()  {
   struct Student* students[MAX_STUDENTS];
    :
}
```

student.h

```
#define MAX_STUDENTS 5
#define MAX_COURSES 3
struct Course
{
    char name[12];
    float score;
};
struct Student {
 int id;
 char name[12];
 int numOfCourses;
 struct Course* courses[MAX_COURSES];
};
struct Student* createStudent(int id,char* name);
void removeStudent(struct Student* students[],int id);
void addCourse(struct Student *s, char* n,float f);
void removeCourse(struct Student *s, char* n);
void showStudents(struct Student* students[]);
void init(struct Student* students[]);
:
```



student.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "student.h"
struct Student* createStudent(int id,char* name)
{
   :
}

void removeStudent(struct Student* students[],int id)
{
   :
}
void addCourse(struct Student *s, char* n,float f)
{
   :
}
void removeCourse(struct Student *s, char* n)
{
   :
}
void showStudents(struct Student* students[])
{
   :
}
void init(struct Student* students[])
{
   :
}
```

student.h

```
#ifndef _STUDENT_H
#define _STUDENT_H
#define MAX_STUDENTS 5
#define MAX_COURSES 3

struct Course
{
    char name[12];
    float score;
};

struct Student
{
    int id;
    char name[12];
    int numOfCourses;
    struct Course* courses[MAX_COURSES];
};
struct Student* createStudent(int id,char* name);
void removeStudent(struct Student* students[],int id);
void addCourse(struct Student *s, char* n,float f);
void removeCourse(struct Student *s, char* n);
void showStudents(struct Student* students[]);
void init(struct Student* students[]);

#endif
```

Create a student

Delete a student from array

add Course to a student

Delete Course which belongs to a student

Show all students on array

Remove all students from the array and free all places from memory

main.c

```c
#include <stdio.h>
#include "student.h"
int main()  {
   struct Student* students[MAX_STUDENTS];
   int i;
   for(i=0;i<MAX_STUDENTS;i++)
     students[i]=NULL;

   students[0] = createStudent(1,"Alex");
   addCourse(students[0],"Java",80);
   addCourse(students[0],"C",90);
   addCourse(students[0],"History",100);

   students[1] = createStudent(2,"Avi");
   addCourse(students[1],"Java",60);
   addCourse(students[1],"C",70);

   showStudents(students);
   removeCourse(students[0],"History");
   showStudents(students);
   removeStudent(students,1);
   showStudents(students);
   init(students);
   showStudents(students);
   return 0;
}
```

array of pointers to students

Initialize array with NULL

Create a students and add courses

Output:

```
Student:1 Alex:
Java 80.000000
C 90.000000
History 100.000000

Student:2 Avi:
Java 60.000000
C 70.000000

Student:1 Alex:
Java 80.000000
C 90.000000

Student:2 Avi:
Java 60.000000
C 70.000000

Student:2 Avi:
Java 60.000000
C 70.000000
```

student.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "student.h"

struct Student* createStudent(int id, char* name)
{
    int i;
    struct Student* s = (struct Student*)malloc(sizeof(struct Student));
    if(s != NULL)
    {
        s->id = id;
        strcpy(s->name, name);
        s->numOfCourses = 0;
        for(i=0; i<MAX_COURSES; i++)
            s->courses[i]=NULL;
        return s;
    }
    return NULL;
}
:
```

student.c cont.

```
void removeStudent(struct Student* students[], int id)
{
    int i;
    if(students == NULL)
          return;
    for(i=0; i<MAX_STUDENTS; i++)
    {
        if(students[i] != NULL && students[i]->id == id)
        {
            int j;
            for(j=0; j<students[i]->numOfCourses; j++)
            {
                if(students[i]->courses[j] != NULL)
                {
                    free(students[i]->courses[j]);
                    students[i]->courses[j] = NULL;
                }
            }
            free(students[i]);
            students[i] = NULL;
            return;
        }
    }
}
:
```

student.c cont.

```
void addCourse(struct Student *s, char* n, float f)
{
    if(s == NULL)
        return;
    if(s->numOfCourses < MAX_COURSES)
    {
        s->courses[s->numOfCourses] = (struct Course*)malloc(sizeof(struct Course));
        if(s->courses[s->numOfCourses] != NULL)
        {
            strcpy(s->courses[s->numOfCourses]->name,n);
            s->courses[s->numOfCourses]->score = f;
            s->numOfCourses++;
        }
    }
}
:
```

student.c cont.

```
void removeCourse(struct Student *s, char* n)
{
    int i;

    if(s == NULL)
        return;

    for(i=0; i<s->numOfCourses; i++)
    {
        if(strcmp(s->courses[i]->name,n) == 0)
        {
            free(s->courses[i]);
            s->courses[i] = s->courses[s->numOfCourses];
            s->courses[s->numOfCourses] = NULL;
            s->numOfCourses--;
            return;
        }
    }
}
:
```

student.c cont.

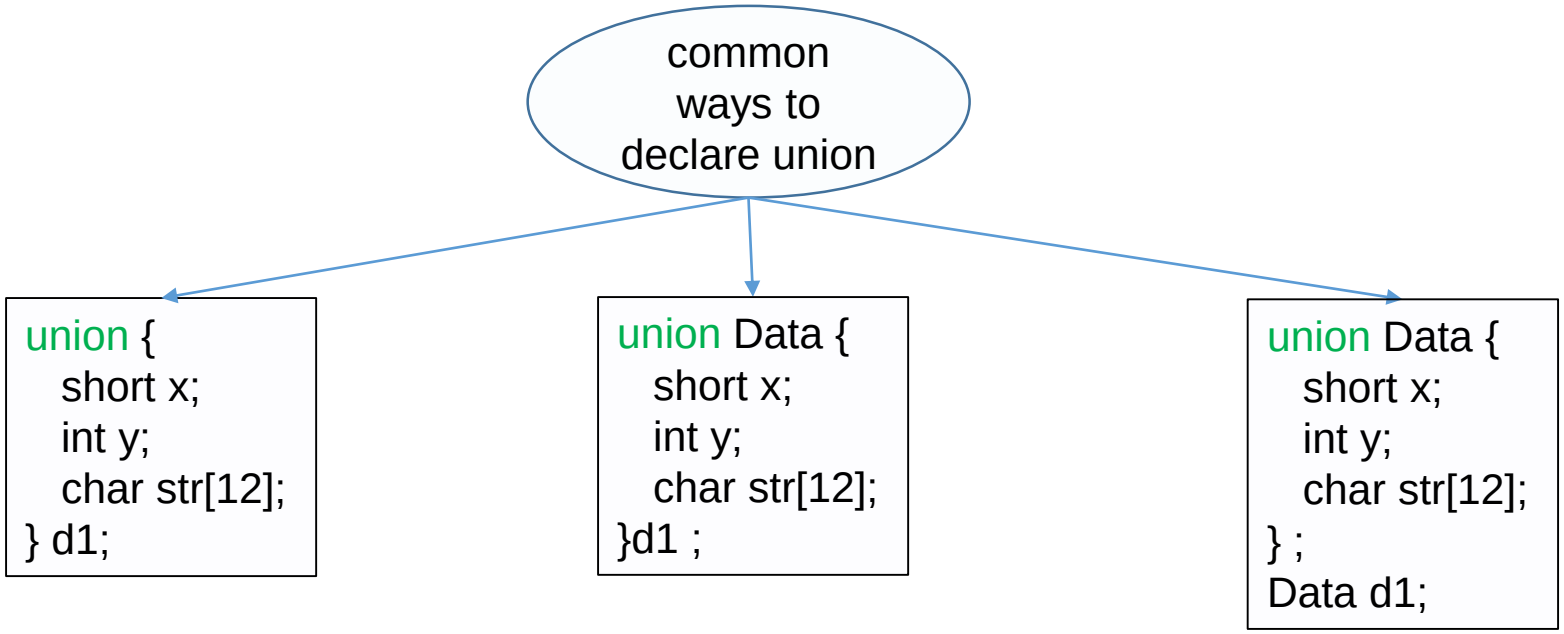
```
void showStudents(struct Student* students[])
{
    int i;
    if(students == NULL)
          return;
    for(i=0; i<MAX_STUDENTS; i++)
    {
        if(students[i] != NULL)
        {
            int j;
            printf("\nStudent:%d %s:\n",students[i]->id,students[i]->name);
            for(j=0; j<MAX_COURSES; j++)
            {
                if(students[i]->courses[j] != NULL)
                {
                    printf("%s %f\n",students[i]->courses[j]->name,students[i]->courses[j]->score);
                }
            }
        }
    }
}
:
```

student.c cont.

```
void init(struct Student* students[])
{
    int i;
    if(students == NULL)
        return;
    for(i=0; i<MAX_STUDENTS; i++)
    {
        if(students[i] != NULL)
        {
            int j;
            for(j=0; j<students[i]->numOfCourses; j++)
            {
                if(students[i]->courses[j] != NULL)
                {
                    free(students[i]->courses[j]);
                    students[i]->courses[j] = NULL;
                }
            }
            free(students[i]);
            students[i] = NULL;
        }
    }
}
```

Unions

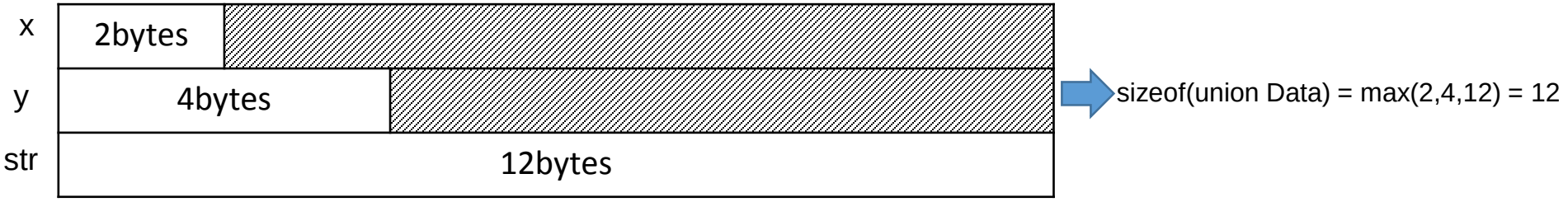
- ✓ **union** allows to store different data types in a shared memory location.
- ✓ at any given time only one member can contain a value.
- ✓ similar to structures, unions my have member fields. But a union can only hold one member at a time.
- ✓ Since a union only holds one member at a time, if two or more members are used without casting, the result could be strange.
- ✓ a Union provides an efficient way of using a shared memory location.



Size of Unions

The size of an instance of union is the amount of memory necessary to represent the largest member plus the padding that raises the length up to an appropriate alignment boundary.

```
#include <stdio.h>
union Data
{
    short x;
    int y;
    char str[12];
} d1;

int main() {
    union Data d2;
    printf("%d %d %d\n",sizeof(union Data), sizeof(d1), sizeof(d2));
    return 0;
}
```



The last initialized member is the one to be defined.

```
#include <stdio.h>
#include <string.h>
union Data{
   int x;
   double y;
   char str[12];
};

int main() {
    union Data d1;
    d1.x=2;
    printf("%d %f %s\n",d1.x,d1.y,d1.str);
    d1.y=2.5;
    printf("%d %f %s\n",d1.x,d1.y,d1.str);
    strcpy(d1.str,"hello!");
    printf("%d %f %s\n",d1.x,d1.y,d1.str);
    return 0;
}
```

```
printf("%d %f %s\n",(int)d1.x,(double)d1.y,(char*)d1.str);
```

use casting to the correct type (to avoid strange results) before use:

- ✓int xx = (int)d1.x;
- ✓double yy = (double)d1.y;
- ✓char* pstr = (char*)d1.str;

undefined

Output:

```
2 0.000000
0 2.500000
1819043176 2.016326 hello!
```

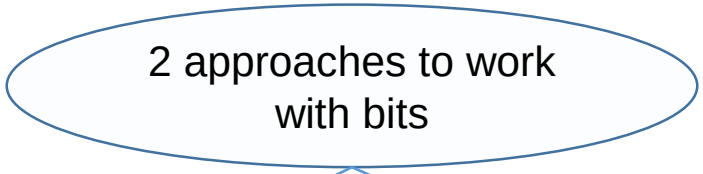
Offsets of Structure Members via Union Members

```
#include <stdio.h>
#include <string.h>
#include <stddef.h>
union Data1 {
    char x;
    char y;
    char str[12];
};

struct Data2 {
    char x;
    char y;
    char str[12];
};

int main() {
    printf("%d %d %d\n",offsetof(union Data1,x), offsetof(union Data1,y), offsetof(union Data1,str));
    printf("%d %d %d\n",offsetof(struct Data2,x), offsetof(struct Data2,y), offsetof(struct Data2,str));
    return 0;
}
```

Output:
0 0 0
0 1 2

Bit-fields

Bit Fields allow the packing of data in a structure



Traditional
- ✓ Using the bits operations (&,|,<<,>>,!)
- ✓ Complex to implement
- ✓ Machine dependent(have to know the order of bits in byte, byte order in words)
- ✓ Widely used in *C*

Modern
- ✓ Using Bit-Fields
- ✓ Access a bit **by name** and not by using bits operations.
- ✓ Machine dependent(have to know the order of bits in words and word size)
- ✓ Easy to implement
- ✓ Uses structures
- ✓ Fields maybe named or not.

Bit-fields Declaration

An integer type that determines how a bit-field's value is interpreted. The type should be 'unsigned int' (may be char, ..)

The number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type.

```
struct {
  type [member_name] : width ;
};
```

The name of the bit-field (optional)

```
#include <stdio.h>
struct {
   unsigned int f1 : 1;
} status;
```

max 4 byte
1 bit is needed
→ sizeof(status) = 4

```
struct {
   unsigned int f1 : 1;
   unsigned int f2 : 1;
} status;
```

max 8 bytes
2 bits are needed
→ sizeof(status) = 4

```
struct {
   unsigned int f1 : 1;
   unsigned int f2 : 5;
   unsigned int f3 : 20;
} status;
```

max 12 bytes
26 bits are needed
→ sizeof(status) = 4

```
struct {
   unsigned int f1 : 10;
   unsigned int f2 : 20;
   unsigned int f3 : 32;
   unsigned int f4 : 32;
} status;
```

max 16 bytes
94 bits are needed
→ sizeof(status) = 12

```
#include <stdio.h>
struct{
   unsigned int is_read: 1;
   unsigned int is_write:1;
}FILE_STATUS;
int main() {
    FILE_STATUS.is_read = 1;
    FILE_STATUS.is_write = 0;
    printf("%d\n",sizeof(FILE_STATUS));
    printf("%d %d\n",FILE_STATUS.is_read,FILE_STATUS.is_write);
    return 1;
}
```

Only 1 bit can be used. If trying to assigned a number more than 1 bit e.g. 2 (binary=11) then system will not allow.

Output:
4
1 0

Typedef

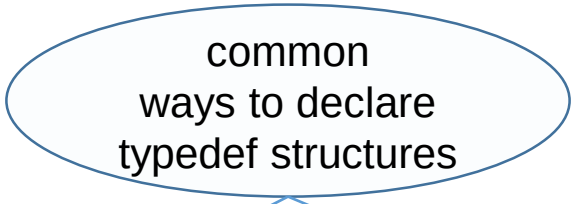
- ✓ C programming language allows giving a new name to an existing data type.
- ✓ **typedef** keyword is used to rename the existing data type.
- ✓ Similar to typedef, define is also used to give aliases for various data types.
  - **typedef** is performed by the compiler.
  - **define** statement is processed by the pre-processor.
  - **typedef** is limited to giving symbolic names to types only whereas define can be used to define alias for values as well

byte is a new name to 'char'

```
#include <stdio.h>
int main() {
   typedef char byte;

   byte x = 255;
   printf("%d", x);

   return 1;
}
```

```
#include <stdio.h>
#define byte char
int main() {

   byte x = 'a';
   printf("%c", x);

   return 1;
}
```

Typedef with Structures

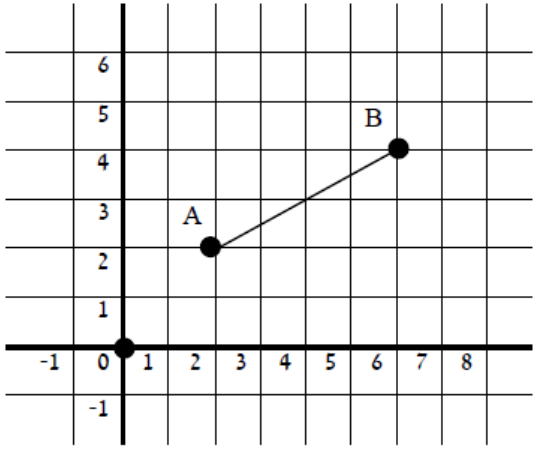



```c
#include <stdio.h>

typedef struct {
    int id;
    char name[12];
} Student;

int main() {
    Student s1 = {1,"Alex"};
    Student s2 = {2,"Avi"};
    printf("%d %s\n",s1.id,s1.name);
    printf("%d %s\n",s2.id,s2.name);
    return 1;
}
```

```c
#include <stdio.h>

struct Stud {
    int id;
    char name[12];
};

int main() {
    typedef struct Stud Student;
    Student s1 = {1,"Alex"};
    Student s2 = {2,"Avi"};
    printf("%d %s\n",s1.id,s1.name);
    printf("%d %s\n",s2.id,s2.name);
    return 1;
}
```

main.c

```
#include <stdio.h>
#include "line.h"
int main()  {
    Point a = {2,2};
    Point b = {6,4};
    Line line = {{2,2},{6,4}};

    printf("%f\n",distance(&a,&b));
    printf("%f\n",lineLength(&line));

    return 1;
}
```



Exercise2: write line.c according to the given line.h and main.c

A = (2, 2), B = (6, 4)

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

4.472136

line.h

```
typedef struct {
    short x;
    short y;
} Point;

typedef struct {
    Point a;
    Point b;
} Line;

float distance(Point* pa, Point* pb);
float lineLength(Line* pLine);
```

line.c

```
#include <math.h>
#define NULL 0
#include "line.h“
float distance(Point* pa, Point* pb) {
    if(pa == NULL || pb == NULL)
        return -1;
    return sqrt(pow(pa->x - pb->x,2)+pow(pa->y - pb->y,2));
}
float lineLength(Line* pLine) {
    if(pLine == NULL)
        return -1;
    return distance(&pLine->a,&pLine->b);
}
```

Enums

- ✓ Enumerated Types are used to create a new Type in C (list of key words).
- ✓ enumeration is used for defining named constant values.
- ✓ **enum** keyword is used to declare enumerated types.
- ✓ Enumerated types make a program looks clearer.

```c
#include <stdio.h>

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

int main()
{
    enum Days d1 = sun;
    enum Days d2 = mon;
    enum Days d3 = sat;
    printf("%d %d %d", d1,d2,d3);
    return 1;
}
```

Output
:
0 1 6

```c
#include <stdio.h>

typedef enum Days {
  sun, mon, tue, wed, thu, fri, sat
}Days;

int main()
{
    Days d1 = sun;
    Days d2 = mon;
    Days d3 = sat;
    printf("%d %d %d", d1,d2,d3);
    return 1;
}
```

declare enum using typedef

Output
:
0 1 6

sun = 0
mon = 1
tue = 2
wed = 3
thu = 4
fri = 5
sat = 6

Enum as Function Argument

```c
#include <stdio.h>

enum Say
{
  Maa, Moo, Haw
};

void what(enum Say x)
{
  switch(x)
  {
   case 0: printf("Maa!\n"); break;
   case 1: printf("Moo!\n"); break;
   case 2: printf("Haw!\n"); break;
  }
}

int main()
{
    what(Maa);
    return 1;
}
```

```c
#include <stdio.h>

typedef enum Say
{
  Maa, Moo, Haw
}Say;

void what(Say x)
{
  switch(x)
  {
   case 0: printf("Maa!\n"); break;
   case 1: printf("Moo!\n"); break;
   case 2: printf("Haw!\n"); break;
  }
}

int main()
{
    what(Maa);
    return 1;
}
```

declare enum using typedef

Functions returning Enums

```c
#include <stdio.h>
enum Boolean
{
    false,
    true
};

enum Boolean isPositive(int x)
{
    return x>0?true:false;
}
int main() {
    enum Boolean b1 = isPositive(1);
    enum Boolean b2 = isPositive(-1);

    if(b1 == true && b2 == false)
        printf("Bingo!\n");
    return 1;
}
```

declare enum using typedef

```c
#include <stdio.h>
typedef enum Boolean
{
    false,
    true
} Boolean;

Boolean isPositive(int x)
{
    return x>0?true:false;
}
int main(){
    Boolean b1 = isPositive(1);
    Boolean b2 = isPositive(-1);

    if(b1 == true && b2 == false)
        printf("Bingo!\n");
    return 1;
}
```

Exercise3: re-write the student.c according to the giving main.c and student.h files **using typedef**

main.c

```c
#include <stdio.h>
#include "student.h"
int main() {
    Student* students[MAX_STUDENTS];
    int i;
    for(i=0; i<MAX_STUDENTS; i++)
        students[i]=NULL;

    students[0] = createStudent(1,"Alex");
    addCourse(students[0],"Java",80);
    addCourse(students[0],"C",90);
    addCourse(students[0],"History",100);
    students[1] = createStudent(2,"Avi");
    addCourse(students[1],"Java",60);
    addCourse(students[1],"C",70);

    showStudents(students);
    removeCourse(students[0],"History");
    showStudents(students);
    removeStudent(students,1);
    showStudents(students);
    init(students);
    showStudents(students);
    return 1;
}
```

student.h

```c
#ifndef _STUDENT_H
#define _STUDENT_H
#define MAX_STUDENTS 5
#define MAX_COURSES 3
typedef struct {
    char name[12];
    float score;
} Course;

typedef struct {
    int id;
    char name[12];
    int numOfCourses;
    Course* courses[MAX_COURSES];
} Student;

Student* createStudent(int id,char* name);
void removeStudent(Student* students[],int id);
void addCourse(Student *s, char* n,float f);
void removeCourse(Student *s, char* n);
void showStudents(Student* students[]);
void init(Student* students[]);
#endif
```

student.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "student.h"

Student* createStudent(int id,char* name)
{
   int i;
   Student* s = (Student*)malloc(sizeof(Student));
   if(s != NULL)
   {
      s->id = id;
      strcpy(s->name, name);
      s->numOfCourses = 0;
      for(i=0; i<MAX_COURSES; i++)
         s->courses[i]=NULL;
      return s;
   }
   return NULL;
}
:
```

student.c cont.

```
void removeStudent(Student* students[],int id)
{
   int i;
   if(students == NULL)
        return;
   for(i=0; i<MAX_STUDENTS; i++)
   {
      if(students[i] != NULL && students[i]->id == id)
      {
         int j;
         for(j=0; j<students[i]->numOfCourses; j++)
         {
            if(students[i]->courses[j] != NULL)
            {
               free(students[i]->courses[j]);
               students[i]->courses[j] = NULL;
            }
         }
         free(students[i]);
         students[i] = NULL;
         return;
      }
   }
}
:
```

student.c cont.

```
void addCourse(Student *s, char* n,float f)
{
  if(s == NULL)
      return;
   if(s->numOfCourses < MAX_COURSES)
   {
      s->courses[s->numOfCourses] = (Course*)malloc(sizeof(Course));
      if(s->courses[s->numOfCourses] != NULL)
      {
         strcpy(s->courses[s->numOfCourses]->name,n);
         s->courses[s->numOfCourses]->score = f;
         s->numOfCourses++;
      }
   }
}
:
```

student.c cont.

```
void removeCourse(Student *s, char* n)
{
   int i;

   if(s == NULL)
      return;

   for(i=0; i<s->numOfCourses; i++)
   {
      if(strcmp(s->courses[i]->name,n) == 0)
      {
         free(s->courses[i]);
         s->courses[i] = s->courses[s->numOfCourses];
         s->courses[s->numOfCourses] = NULL;
         s->numOfCourses--;
         return;
      }
   }
}
:
```

student.c cont.

```
void showStudents(Student* students[])
{
   int i;
   if(students == NULL)
        return;
   for(i=0; i<MAX_STUDENTS; i++)
   {
      if(students[i] != NULL)
      {
         int j;
         printf("\nStudent:%d %s:\n",students[i]->id,students[i]->name);
         for(j=0; j<MAX_COURSES; j++)
         {
            if(students[i]->courses[j] != NULL)
            {
               printf("%s %f\n",students[i]->courses[j]->name,students[i]->courses[j]->score);
            }
         }
      }
   }
}
:
```

student.c cont.

```
void init(Student* students[])
{
   int i;
   if(students == NULL)
        return;
   for(i=0; i<MAX_STUDENTS; i++)
   {
      if(students[i] != NULL)
      {
         int j;
         for(j=0; j<students[i]->numOfCourses; j++)
         {
            if(students[i]->courses[j] != NULL)
            {
               free(students[i]->courses[j]);
               students[i]->courses[j] = NULL;
            }
         }
         free(students[i]);
         students[i] = NULL;
      }
   }
}
```


END