STRUCTURES

(PART 2)

prepared by Senem Kumova Metin
modified by İlker Korkmaz

Review on "struct" concept

- A **structure** has components, called ***members***, which can be of various types.
- The ***declaration of a structure*** captures the information needed to represent the related structure.
- The keyword is ***struct***

Structures (III)

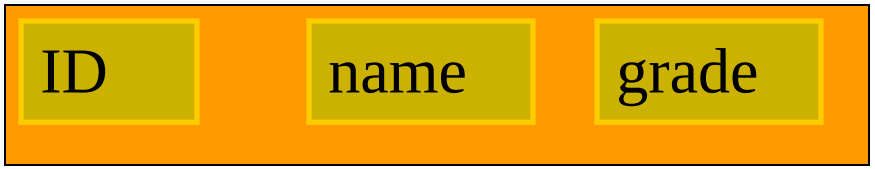
```
struct record { int ID; char * name; char grade; };
//alternative: struct record { int ID; char name[20]; char grade; };
```

s1

```
struct record s1;
```

s2

```
struct record s2;
```



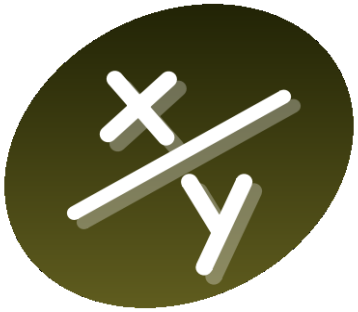
Structures : DECLARATION ALTERNATIVES (I)

Declaration 1 :

```
struct record {
    int ID;
    char name[20]; // or: char * name;
    char grade;
};
struct record s1; // sample definitions
struct record s2;
struct record s3;
```

Exercise 1 on Structures (I)

- Declare a structure to represent fractions
- Create 2 variables of fractions
- Ask user to fill the variables
- Calculate and print out the multiplication result



- → Fraction → x / y (such as 3/7, 8/5...)
- → Need 2 numbers as numerator and denominator
- → User has to give 2 member values for each variable
- → Multiplication rule → a/b * c/d = (a*c) / (b*d)

Exercise 1 on Structures (II)

```
#include <stdio.h>

struct fraction {
        int n;  // an integer to represent numerator
        int d;  // an integer to represent denominator
};

int main() {
    struct fraction obj1, obj2;   //  Create input variables
    struct fraction result;       //  Create a variable to store the result

    printf("please enter the values for 2 members of each fraction structures:\n");
    scanf("%d%d", &obj1.n , &obj1.d);
    scanf("%d%d", &obj2.n , &obj2.d);

    result.n = obj1.n * obj2.n;
    result.d = obj1.d * obj2.d;

    printf("result is %d/%d", result.n, result.d);
     return 0;
 }
```

Exercise 2 on Structures (I)

- Declare a structure to represent a "customer" (name, surname, phone number)
- Create 5 customer variables
- Ask user to fill the variables
- Print out the information for all customers, such as the following:

```
customer 1 : Gabriel Gray          1222222
customer 2 : Claire Bennet         1234567
customer 3 : Hiro Nakamura         2222222
customer 4 : Nathan Petrelli       1234569
customer 5 : Niki Sanders          2333333
```

Exercise 2 on Structures (II)

```
#include <stdio.h>
struct customer
{
    char name[100];
    char surname[100];
    int phone;
};
int main() {
    struct customer heroes[5];
    int i;
    printf("please fill the customer records:\n");
    for(i=0;i<5;i++)
        scanf("%s%s%d", heroes[i].name, heroes[i].surname, &heroes[i].phone);
    for(i=0;i<5;i++)
        printf("customer %d :  %s %s\t%d\n", i+1, heroes[i].name, heroes[i].surname, heroes[i].phone);
    return 0;
}
```

Exercise 3 on Structures (I)

- Declare a structure to represent the complex numbers (real and imaginary part)
- Create a dynamic variable to point a complex number at run time (use calloc/malloc to allocate the memory dynamically)
- Ask user to fill the variable
- Sample output:

```
Please give the real and the imaginary member values of the complex number : 3  5
Info of your complex number :  3 + 5i
```

Exercise 3 on Structures (II)

```
#include <stdio.h>
#include <stdlib.h>   // to declare “malloc”

struct complex_num
{           int real;
            int imaginary;          };

int main()
{
   struct complex_num * p;

   /* dynamic memory allocation for the complex number*/
   p=(struct complex_num *) malloc(sizeof(struct complex_num));

   printf(  "Please give the values for complex number : " );
   scanf(  "%d%d", &(p->real), & (p->imaginary)  );

   printf(  "Complex number :   %d + %d i", p->real, p->imaginary );

   return 0;
}
```

A general example for structures: STACK (a LIFO or FILO data structure)

```
/*1st file of the project: stack.h*/
#define STACK_SIZE 4
typedef struct st { int data [STACK_SIZE]; int top;} stack;

/*2nd file of the project: stack.c*/
#include<stdio.h>
#include "stack.h"
void reset(stack * stk) {
        stk->top=-1;
}

void push( int c, stack * stk){
        if( stk->top!= STACK_SIZE-1)
        {
                stk->top++;
                stk->data[stk->top]=c;
        }
        else
                printf("Stack is full!!\n");
}

int pop (stack * stk){
        if(stk->top!=-1)
                return (stk->data[stk->top--]);
        else
        {
                printf("stack is empty");
                return -1;
        }
}
```

```
/*3rd file of the project: test.c*/
#include<stdio.h>
#include "stack.h"
int main(){
        stack n;
        int x;
        reset(&n);
        push(4,&n);
        push(5,&n);
        x= pop(&n);
        printf("%d\n",x);
        push(3,&n);
        push(2,&n);
        x= pop(&n);
        printf("%d\n",x);
    return 0;
}
```