First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #08

# Introduction to Programming

Structures in C

Course Instructor:

**Dr. Monidipa Das**

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

# What is a Structure?

- User-defined data type

- It is a convenient tool for handling a group of logically related data items.
  - Student name, roll number, and marks
  - Real part and complex part of a complex number

- Helps in organizing complex data in a more meaningful way.

# Defining a Structure

- The composition of a structure may be defined as:

```
struct tag {
          member 1;
          member 2;
          …
          …
          member m;
            };
```

  - **struct** is the required keyword.
  - *tag* is the name of the structure.
  - *member 1*, *member 2*, … are individual member declarations.

# Defining a Structure

- The individual members can be ordinary variables, pointers, arrays, or other structures.
  - The member names within a particular structure must be distinct from one another.
  - A member name can be the same as the name of a variable defined outside of the structure.

- Once a structure has been defined, individual structure-type variables can be declared as:

**struct tag** variable_1, variable_2, …, variable_n;

# Example Structure

- A structure definition:

```
struct student {
                char name[30];
                int roll_number;
                int total_marks;
                char dob[10];
               };
```

- Declaring structure variables:

```
struct student a1, a2, a3;
```

**A new data-type**

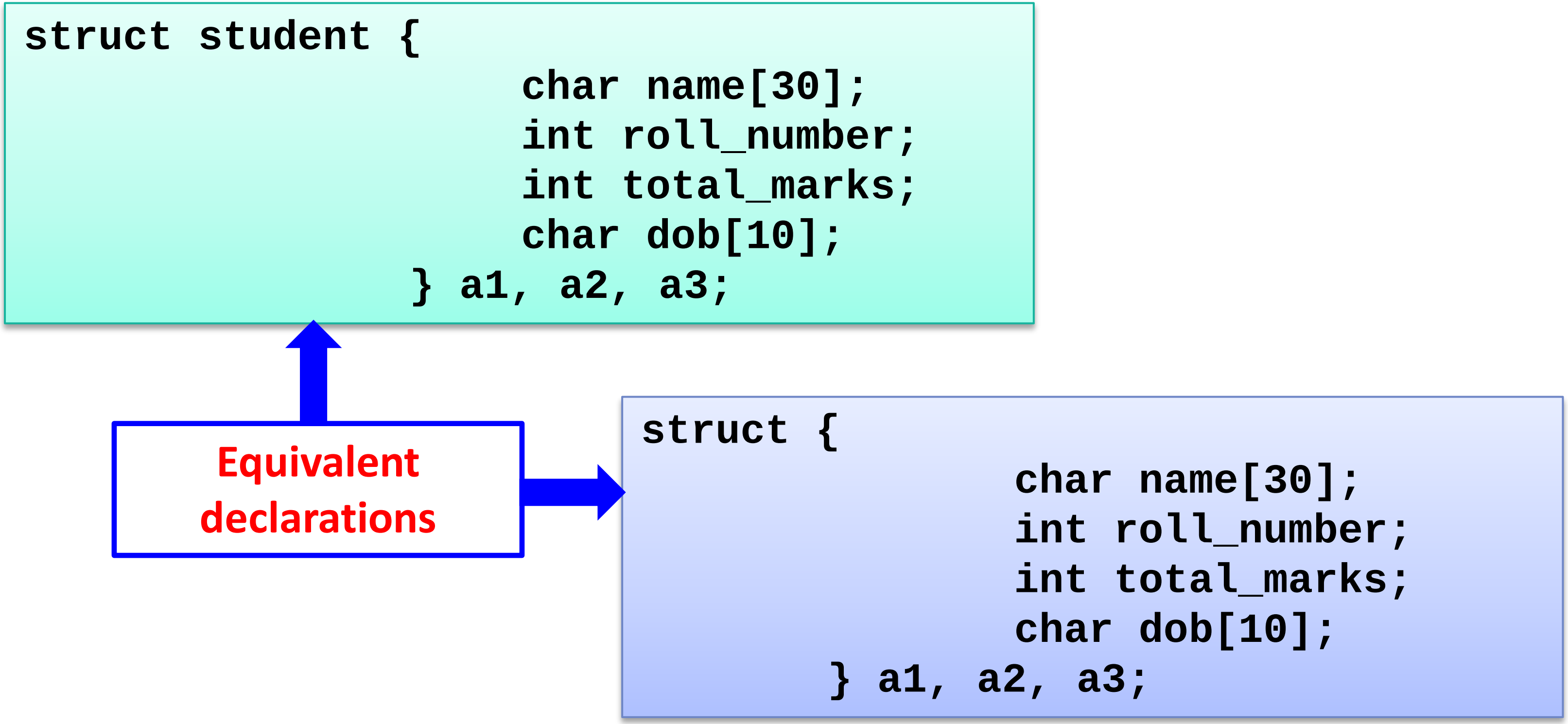
# A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
               member 1;
               member 2;
               :::::
               member m;
              } variable_1, …, variable_n;
```

- In this form, “tag” is optional.

# Example – Structure Declaration



```
struct student {
                    char name[30];
                    int roll_number;
                    int total_marks;
                    char dob[10];
                } a1, a2, a3;
```

Equivalent declarations

```
struct {
               char name[30];
               int roll_number;
               int total_marks;
               char dob[10];
        } a1, a2, a3;
```

# Processing a Structure

- The members of a structure are processed individually, as separate entities.

- A structure member can be accessed by writing ***variable.member*** where ***variable*** refers to the *name of a structure type variable*, and ***member*** refers to the *name of a member within the structure*.

- **Examples:**

```
struct student s1, s2;
s1.name, s1.roll_number, s2.dob
```

```
struct student {
                char name[30];
                int roll_number;
                int total_marks;
               char dob[10];
            };
```

# Example: Complex number addition

```
#include<stdio.h>
int main()
{
        struct complex
        {
                float real;
                float complex;
        } a, b, c;

        scanf ("%f%f", &a.real, &a.complex);
        scanf ("%f%f", &b.real, &b.complex);
        c.real = a.real + b.real;
        c.complex = a.complex + b.complex;
        printf ("\n %f + %f i", c.real, c.complex);
        return 0;}
```

**Scope restricted within main()**

**Structure definition and Variable Declaration**

**Reading a member variable**

**Accessing members**

# Comparison of Structure Variables

- Unlike arrays, group operations can be performed with structure variables.
  - A structure variable can be directly assigned to another structure variable of the same type.

    a1 = a2;
  - All the individual members get assigned.

- Two structure variables cannot be compared for equality or inequality.
  - if (a1 = = a2) …….
  - Compare all members and return 1 if they are equal; 0 otherwise.

# Arrays of Structures

- Once a structure has been defined, we can declare an array of structures.

  **struct student** bstat1[50];

- The individual members can be accessed as:

  bstat1[i].name

  bstat1[8].roll_number

# Arrays within Structures

- A structure member can be an array:

```
struct student {
                    char name[30];
                    int roll_number;
                    int marks[5];
                    char dob[10];
               } a1, a2, a3;
```

- The array element within the structure can be accessed as: a1.marks[2]

# Defining data type: using *typedef*

- One may define a structure data-type with a single name.

- General syntax:

```
typedef struct {
                member-variable1;
                member-variable2;
                ….
                member-variableN;
             } tag;
```

- **tag** is the name of the new data-type.

**Example:**

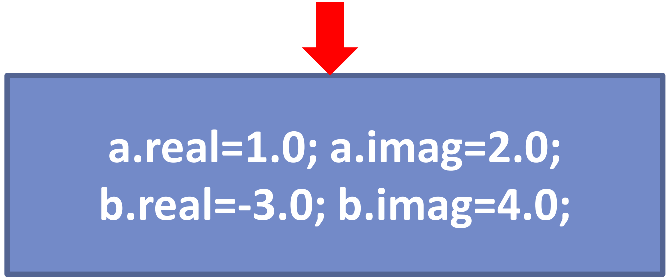
```
typedef struct{
        float real;
        float imag;
        } COMPLEX;

COMPLEX a,b,c;
```

# Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.

- **An example:**

```
COMPLEX a={1.0,2.0}, b={-3.0,4.0};
```



```
a.real=1.0; a.imag=2.0;
b.real=-3.0; b.imag=4.0;
```

# Parameter Passing in a Function

- Structure variables could be passed as parameters like any other variable. Only the values will be copied during function invocation.

```
void swap(COMPLEX a, COMPLEX b)
{
        COMPLEX tmp;
        tmp=a;
        a=b;
        b=tmp;
}
```

# An example program

```
#include <stdio.h>
typedef struct{
        float real;
        float imag;
} COMPLEX;

void swap(COMPLEX a, COMPLEX b)
{
        COMPLEX tmp;
        tmp=a;
        a=b;
        b=tmp;
}
```

```
void print(COMPLEX a)
{
        printf("(%f , %f) \n",a.real,a.imag);
}
main()
{
        COMPLEX x={4.0,5.0},y={10.0,15.0};
        print(x); print(y);
        swap(x,y);
        print(x); print(y);
}
```

# Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself.

```
COMPLEX add (COMPLEX a, COMPLEX b)
{
    COMPLEX tmp;
    tmp.real = a.real+b.real;
    tmp.imag = a.imag+b.imag;
    return(tmp);
}
```

**Direct arithmetic operations are not possible with Structure variables.**

# Pointers and Structures

- You may recall that the ***name of an array stands for the address of its zero-th element***.
  - Also true for the names of arrays of structure variables.

- **Consider the declaration:**

```
struct stud {
    int roll;
    char dept_code[25];
    float cgpa;
};
struct stud bstat1[100], *ptr;
```

# Pointers and Structures

- The name ***bstat1*** represents the address of the zero-th element of the structure array.
- ***ptr*** is a pointer to data objects of the type struct stud.

- The assignment

```
ptr = bstat1;
```

  will assign the address of ***bstat1[0]*** to ***ptr***.

- When the pointer ***ptr*** is incremented by one (***ptr++***)
  - The value of ***ptr*** is actually increased by ***sizeof(stud)***.
  - It is made to point to the next record.

# Pointers and Structures

- Once ***ptr*** points to a structure variable, the members can be accessed as:

```
ptr -> roll ;
ptr -> dept_code ;
ptr -> cgpa ;
```

- The symbol "–>" is called the arrow operator.

# Example

```
#include <stdio.h>
typedef struct {
        float real;
        float imag;
} COMPLEX;
```

```
void swap_ref(COMPLEX *a, COMPLEX *b)
{
        COMPLEX tmp;
        tmp=*a;
        *a=*b;
        *b=tmp;
}
```

```
void print(COMPLEX *a)
{
        printf("(%f,%f)\n",a->real,a->imag);
}
```

```
int main() {
        COMPLEX x={10.0,3.0};
        COMPLEX y={-20.0,4.0};
        print(&x);
        print(&y);
        swap_ref(&x,&y);
        print(&x);
        print(&y);
        return 0;
}
```

# Warning

- When using structure pointers, we should take care of operator precedence.
  - Member operator "." has higher precedence than "*".
    - ptr –> roll and (*ptr).roll mean the same thing.
    - *ptr.roll will lead to error.
  - The operator "–>" enjoys the highest priority among operators.
    - ++ptr –> roll will increment roll, not ptr.
    - (++ptr) –> roll will serve the purpose

# Example-1

- Define a structure type ***student*** to store the ***name, roll,*** and ***total-marks*** of any student. Write a program to read this information (from keyboard) for student and print the same on the screen.

```c
#include<stdio.h>
struct student{
        char name[30];
        int roll;
        float marks;
};
int main()
{
        struct student s1, s2, *sptr;
        sptr=&s2;
```

```c
        printf("Enter Name: ");
        gets(s1.name);
        printf("Enter Roll No.: ");
        scanf("%d",&s1.roll);
        printf("Enter Total Marks: ");
        scanf("%f",&s1.marks);
        fflush(stdin);
        printf("\nEnter Name: ");
        gets((*sptr).name);
        printf("Enter Roll No.: ");
        scanf("%d",&sptr->roll);
        printf("Enter Total Marks: ");
        scanf("%f",&sptr->marks);
```

# Example-1 [contd.]

```
        printf("\n\nName:%20s\n",s1.name);
        printf("Roll No.:%16d\n",s1.roll);
        printf("Total Marks:%13.2f\n",s1.marks);


        printf("\n\nName:%20s\n",sptr->name);
        printf("Roll No.:%16d\n",sptr->roll);
        printf("Total Marks:%13.2f\n",sptr->marks);


        printf("\n\nName:%20s\n",(*sptr).name);
        printf("Roll No.:%16d\n",(*sptr).roll);
        printf("Total Marks:%13.2f\n",(*sptr).marks);

        return 0;
}
```

```
Enter Name:  Nandini Das
Enter Roll No.: 42
Enter Total Marks: 9.24

Enter Name: Aruna Basak
Enter Roll No.: 55
Enter Total Marks: 9.13


Name:       Nandini Das
Roll No.:            42
Total Marks:       9.24


Name: Aruna Basak
Roll No.:            55
Total Marks:       9.13


Name: Aruna Basak
Roll No.:            55
Total Marks:       9.13
```

# Example-2

- Write a C program to perform *addition* and *multiplication* of any two complex numbers, taken as input from the terminal.

```
#include <stdio.h>

//Defining structure to represent complex numbers
struct complex
{
  float real;
  float imaginary;
};

void add_complex(struct complex, struct complex);
void multiply_complex(struct complex, struct complex);
```

# Example-2 [contd.]

```c
// Function to add two complex numbers
void add_complex(struct complex c1, struct complex c2)
{
     struct complex f;

      f.real = c1.real + c2.real;
      f.imaginary = c1.imaginary + c2.imaginary;

      if ( f.imaginary >= 0 )
        printf("\nSum of two complex numbers = %0.2f + %0.2fi",f.real,f.imaginary);
      else
        printf("\nSum of two complex numbers = %0.2f %0.2fi",f.real,f.imaginary);
}
```

# Example-2 [contd.]

```
//Function to multiply two complex numbers
void multiply_complex(struct complex c1, struct complex c2)
{
      struct complex f;

      f.real = c1.real*c2.real - c1.imaginary*c2.imaginary;
      f.imaginary = c1.imaginary*c2.real + c1.real*c2.imaginary;

      if (f.imaginary >= 0 )
        printf("\nMultiplication of two complex numbers = %0.2f + %0.2fi",f.real,f.imaginary);
      else
        printf("\nMultiplication of two complex numbers = %0.2f %0.2fi",f.real,f.imaginary);
}
```

# Example-2 [contd.]

```
int main()
{
        struct complex cn1,cn2;
        printf("Enter a and b where a + ib is the first complex number:");
        printf("\na = ");
        scanf("%f", &cn1.real);
        printf("b = ");
        scanf("%f", &cn1.imaginary);
        printf("\nEnter c and d where c + id is the second complex number:");
        printf("\nc = ");
        scanf("%f", &cn2.real);
        printf("d = ");
        scanf("%f", &cn2.imaginary);
        add_complex(cn1,cn2); // Calling function to add two complex numbers
        multiply_complex(cn1,cn2); // Calling function to multiply two complex numbers
        return 0;
}
```

# Questions?