Module 9 – Structures in C

BITS Pilani
Pilani Campus

Dr. Ashutosh Bhatia & Dr. Asish Bera
Department of Computer Science & Information Systems

Module Overview

- Tuple Data
- Structures
- Nested Structures and Array of Structures
- Passing Structure Variables to Functions
- Union
- Enumerator

innovate achieve lead

BITS Pilani

Pilani Campus

Tuple Data

Tuple Data

Consider student database. Every student has a few attributes:

- *Name*
- *ID*
- *Group*

This is known as tuple data where ***{Name, ID No, Group}*** is a tuple

Example: The below table contains 3 records of the above tuple type

ID	Name	Group
0910	Rama Sarma	B4
0313	Alex Mathew	A8
0542	Vijay Kumar	A7

Tuple Data is used in Database Systems

Representing Tuple Data

First attempt: 3 lists (i.e. arrays)

- **Name array**, **ID array**, and **Group array**.
- Problems: Consider add / delete operations.
  - Shifting to be done in 3 arrays separately.
  - 3 arrays are the 3 element values passed as parameters.
  - Data representation "does not say" the 3 things are related.

Better Solution: 1 list of triples

- Each triple is of the form **(ID, Name, Group)**
- add / delete operation: Shifting for one array only.
  - *Can use Structures in C to do this.*

BITS Pilani
Pilani Campus

Structures in C

Structures in C

- **Structure**
  - Group of logically related data items
  - Example: **ID, Name, Group**
- A single structure may contain data of one or more data types
  - ID: **int ID**
  - Name: **char Name[]**
  - Group: **char Group[]**
- The individual structure elements are called ***members***
- We are essentially defining a ***new data type***
- ***Then we can create variables of the new data type***
  - These variables can be ***global, static*** or ***auto***

Defining a Structure

```
struct struct_name {
    member 1;
    member 2;
    ...
};
```

- `struct` is the required keyword
- `struct_name` is the name of the structure
- `member 1, member 2, ...` are individual member declarations

Example: Student structure containing ID, Name, Group

```
struct student {
    int ID;
    char Name[20];
    char Group[3];
}
```

Members

Declaring Variables of the Structure type

innovate achieve lead

```
struct  struct_name var_1, var_2, …, var_n;
```

Example 1:

```
struct student {
   int ID;
   char Name[20];
   char Group[3];
};

// declaring variable of
// the type struct student
struct student s1, s2;
```

Example 2:

```
struct book {
   char Name[20];
   float price;
   char ISBN[30];
};

// declaring variable of
// the type struct book
struct book b1, b2;
```

Combining Structure Definition and Variable Definition


```
struct struct_name{
data_type member 1;
data_type member 2;
…
}var1, var2, … varn;
```

Example:

```
struct student {
   int ID;
   char Name[20];
   char Group[3];
} s1, s2;
```

```
struct {
data_type member 1;
data_type member 2;
…
}var1, var2, … varn;
```

Example:

```
struct {
   int ID;
   char Name[20];
   char Group[3];
} s1, s2;
```

Accessing members of a structure

innovate achieve lead

- A structure member can be accessed by writing

  structure_variable.MemberName

Example: Consider the following structure definition:

```
struct student {
    int ID;
    char Name[20];
    float Marks[5];
}s1, s2;
```

s1.Name → Value stored in the *Name* field of s1 variable

s2.Name → Value stored in the *Name* field of s2 variable

s2.Marks[i] → Value stored at the i^{th} position of the *Marks* array of s2

Example

```
#include main()
{
   struct complex {
       float real;
       float cmplex;
   } a, b, c;

   scanf ("%f %f", &a.real, &a.cmplex);
   scanf ("%f %f", &b.real, &b.cmplex);
   c.real = a.real + b.real;
   c.cmplex = a.cmplex + b.cmplex;
   printf ("\n %f + %f j", c.real, c.cmplex);
}
```

Where can we define structures?

innovate achieve lead

- **Structures can be defined in:**
  - **Global Space**
    - ***Outside all function definitions***
    - Available everywhere inside and outside all functions to declare variables of its type
  - **Local Space**
    - ***Inside a function***
    - Variables of this structure type can be declared and used only inside that function.

Examples

innovate achieve lead

Structure defined in global space:

```
#include <stdio.h>
struct student{
   int ID;
   char Name[20];
   float marks[5];
};
int f1() {
   struct student s1;
   … // operations on s1
}
int f2() {
   struct student s2;
   … // operations on s2
}
int main() {
   …
}
```

Structure defined in local space:

```
#include <stdio.h>
int f1() {
   struct student{
        int ID;
        char Name[20];
        float marks[5];
   };
   struct student s3;
   … // operations on s3
}
int f2() {
   // can't define a variable
   with struct student here
}
int main() {
   …
}
```

Structure Initialization

innovate achieve lead

```
struct book {
        char Name[20];
        float price;
        char ISBN[30];
};
struct book b1 = {"Book1", 550.00, "I1"},
            b2 = {"Book2", 650.00, "I2"};
```

```
b1.Name = Book1          b2.Name = Book2
b1.price = 550.00        b2. price = 650.00
b1.ISBN = I1             b2.ISBN = I2
```

typedef

Allows users to define new data-types

Syntax:

```
typedef type new-type
```

Example 1:

```
typedef int Integer;
Integer I1, I2;
```

Example 2:

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

COMPLEX c1, c2;
```

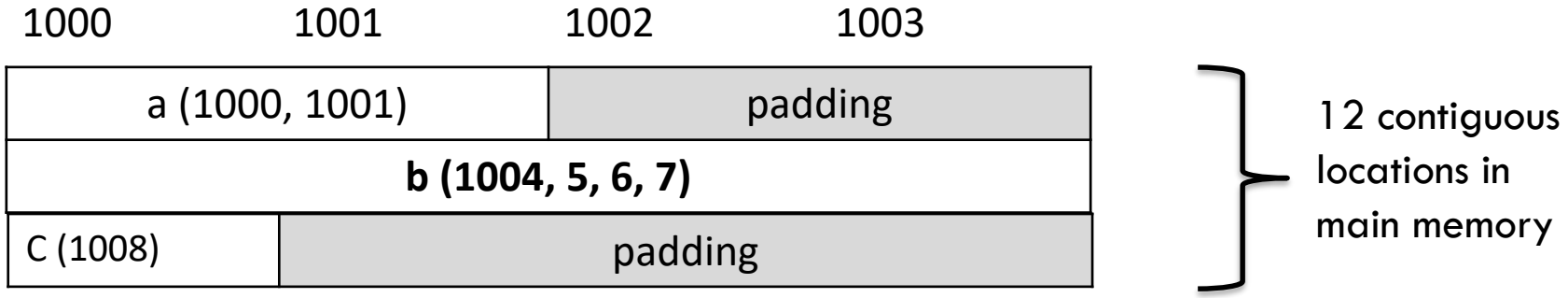
- It is a common practice to define structures using `typedef`
- It simplifies the syntax and increases readability of the program

sizeof() for struct variables

```
struct test_struct {
    short a; //2bytes
    int b; //4bytes
    char c; //1byte
} test;
```

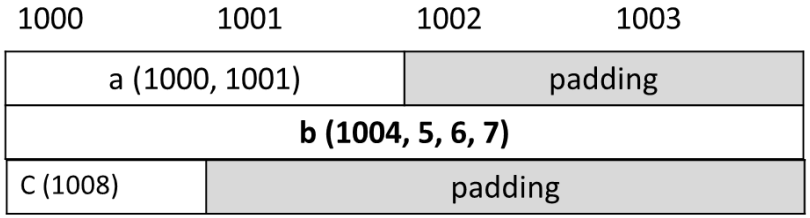
```
printf("a= %lu\n", sizeof(test.a)); = 2
printf("b= %lu\n", sizeof(test.b)); = 4
printf("c= %lu\n", sizeof(test.c)); = 1
printf("%lu\n", sizeof(test));      = 12
```

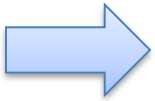
- We can notice that the `sizeof(test) > sizeof(test.a) + sizeof(test.b) + sizeof(test.c)`
- This is because the compiler adds (may add) padding for alignment requirements
- Padding means to append empty locations towards the end (or beginning)

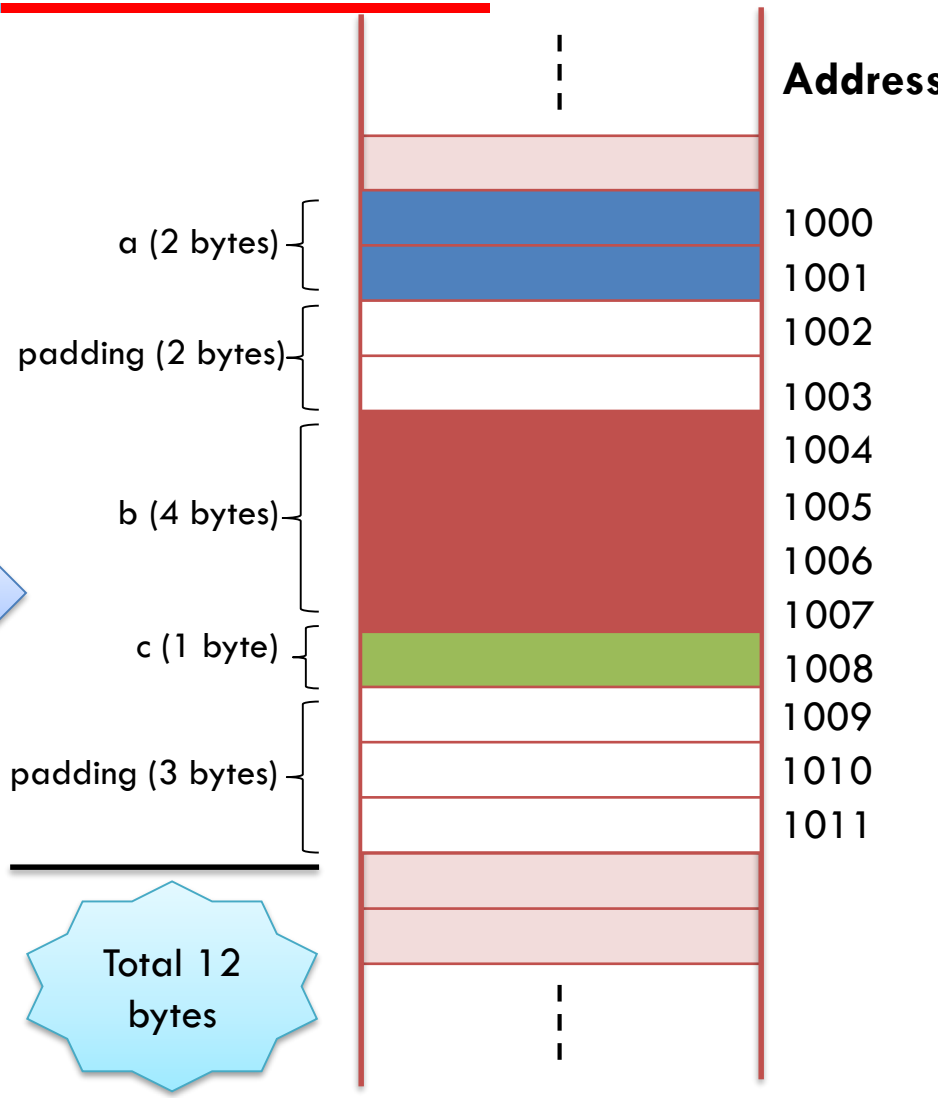


Memory view of storing a struct variable

```
struct test_struct {
    short a; //2bytes
    int b; //4bytes
    char c; //1byte
} test;
```







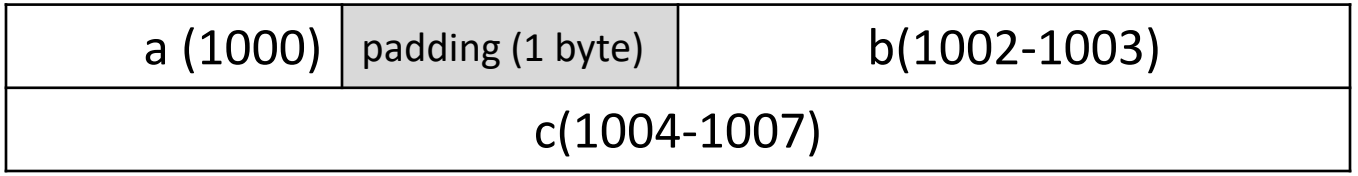
Note: This illustration considers that each address is represented by 16 bits.

Also note that in this example `int` is occupying 4 bytes. Some compilers use 2 bytes for `int`, and some use 4 bytes.

Another Example

```
struct test_struct {
    char a; //1byte
    short b; //2bytes
    int c; //4bytes
} test;
```

```
printf("a= %lu\n", sizeof(test.a)); = 1
printf("b= %lu\n", sizeof(test.b)); = 2
printf("c= %lu\n", sizeof(test.c)); = 4
printf("%lu\n", sizeof(test));      = 8
```



- We can notice that the padding added by compiler is less (**only 1 byte**), for the same set of variable types, *arranged with a different order in the structure*.
- The total number of bytes to be added for padding is decided by the compiler depending upon the arrangement of members inside the structure.

Operations on struct 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;
```

  The contents of members of a2 are copies to members of a1.
- You can't do this for two arrays, without using *"pointers"*
  - *We will study about pointers in the next module*
- You still can't do equality check for two structure variables, i.e. `if (a1 == a2)` is not valid in C
  - *You shall have to check equality for individual members*

Swap two structure variables

```
struct book {
        char Name[20];
        float price;
        char ISBN[30];
};
int main(){
    struct book b1 = {"Algorithms", 550.00, "I1"};
    struct book b2 = {"Programming", 650.00, "I2"};
    // swapping b1 and b2
    struct book b3;
    b3 = b1;
    b1 = b2;
    b2 = b3;
    printf("The name of Book in b1 is %s\n", b1.Name);
    printf("The name of Book in b2 is %s\n", b2.Name);
    return 0;
}
```

Output:

The name of the Book in b1 is Programming

The name of the Book in b2 is Algorithms

Incorrect way of using structure variables


```
struct book {
        char Name[20];
        float price;
        char ISBN[30];
};
int main(){
    struct book b1 = {"Algorithms", 550.00, "I1"};
    struct book b2 = {"Programming", 650.00, "I2"};
    // Incorrect way of using structure variables
    // if (b1 == b2)
    if(b1.price == b2.price) // Correct way
        printf("Both the books have the same price\n");
    else
        printf("Both the books DON'T have the same price");
    return 0;
}
```

innovate achieve lead

BITS Pilani

Pilani Campus

Nested Structures and Array of Structures

Nested Structures

```
struct student{
        float CGPA;
        char dept[10];
        struct address
        {
            int PINCODE;
            char city[10];
        } add;
} s1;
```

```
struct address
{
    int PINCODE;
    char city[10];
};
struct student
{
    float CGPA;
    char dept[10];
    struct address add;
}s1;
```

To access PINCODE → `s1.add.PINCODE`

Example of nested structures

innovate achieve lead

```
struct address
{
    int PINCODE;
    char city[10];
};
struct student
{
    float CGPA;
    char dept[10];
    struct address add;
};
```

```
int main() {
    struct student s1;
    s1.add.PINCODE = 333031;
    s1.add.city = "Vizag";
    s1.CGPA = 10.0;
    s1.dept = "CS";
    printf("s1's city is %s", s1.add.city);
    printf("s1's CGPA is %f", s1.CGPA);
}
```

Array of Structures

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

Example:

```
struct book {
        char Name[20];
        float price;
        char ISBN[30];
};
struct book bookList[50];
```

The individual members can be accessed as:

`bookList[i].price` → returns the price of the i^{th} book.

Example

innovate achieve lead

Program to store information of 5 students

```
#include <stdio.h>

struct student{
   char Name[10];
   float CGPA;
   float marks[5];
};

struct student s[5];
```

```
int main(){
printf("Enter the details:\n");
for(i = 0; i < 5; i++)
{
    printf("Enter name\n");
    scanf("%s", s[i].name);
    printf("Enter the marks\n");
    for(j = 0; j< 5; j++)
        scanf("%f",&s[i].marks[j]);
    printf("Enter the CGPA\n");
    scanf("%f",&s[i].CGPA);
}}
```

Storing database records in Array of Structures

- Remember our motivation for structures to store tuple data…

ID	Name	Group
0910	Rama Sarma	B4
0313	Alex Mathew	A8
0542	Vijay Kumar	A7

- The above table can be stored in a `struct student` array – `arr`
- Now its easy to add/delete records:
  - Simply create a new variable of the type `struct student` and append it to `arr`
  - If the array is sorted on ID numbers, it is sufficient to do necessary shiftings in `arr` itself and store the new record in its appropriate position
  - Deletion is also similar

Refer to slides of Module 8 to understand add/append/delete in an array.

Passing Structures to Functions

Passing struct variables to functions

- A structure can be passed as argument to a function
- Structures can be passed both by pass by value and pass by reference
  - *We will study about pass by reference later with "pointers"*
- A function can also return a structure
- Array of structures are by default passed by reference
  - Just like any other arrays.
  - *We will study later with pointers*

Example – Pass by value

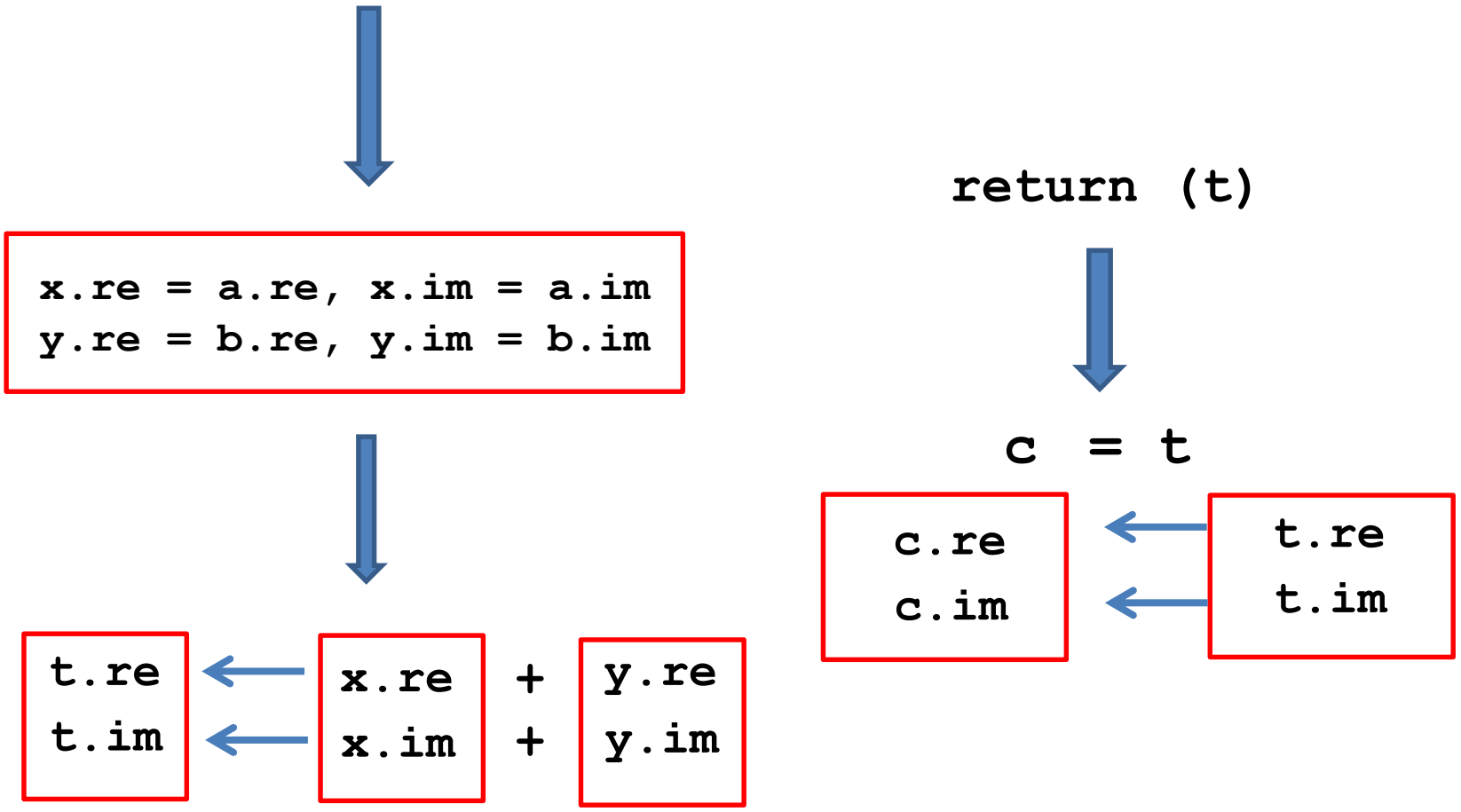
```
#include <stdio.h>
typedef struct {
    float re;
    float im;
} cmplx;

int main() {
    cmplx a, b, c;
    scanf ("%f %f", &a.re, &a.im);
    scanf ("%f %f", &b.re, &b.im);
    c = add (a, b) ;
    printf ("\n %f %f", c,re, c.im);
}
```

```
cmplx add(cmplx x,cmplx y){
    cmplx t;
    t.re = x.re + y.re ;
    t.im = x.im + y.im ;
    return (t) ;
}
```

- When the function **add() is called,** the **values of the members of variables a and b are copied into members of variables x and y.**
- When the function **add() returns, the values of the members of variable t are copied into the members of variable c.**

Example: Passing structure by value illustrated


```
cmplx add(cmplx x,cmplx y) and c = add(a, b);
```



Previous Example (modified)

Program to store information of 5 students, compute total marks for each student and print it.

```
#include <stdio.h>

struct student{
   char Name[10];
   float CGPA;
   float marks[5];
};

struct student s[5];
```

```
void computeSum(struct student[], int);
int main(){
printf("Enter the details:\n");
for(i = 0; i < 5; i++)
{
    printf("Enter name\n");
    scanf("%s", s[i].name);
    printf("Enter the marks\n");
    for(j = 0; j< 5; j++)
        scanf("%f",&s[i].marks[j]);
    printf("Enter the CGPA\n");
    scanf("%f",&s[i].CGPA);
}
computeSum(s,5);
}
```

Example (contd.)

```
void computeSum(struct student sArr[], int size){
    int i,j,sum;
    for(i=0;i<size;i++)
    {
        sum = 0;
        for(j=0;j<5;j++)
        {
            sum += sArr[i].marks[j];
        }
        printf("Total marks of %s are %d", sArr[i].Name, sum);
    }
    return;
}
```

Enumerator in C

Enumerator in C

- Consider this structure definition:

```
struct student {
    int ID;
    char Name[20];
    char Group[3];
};
```

- The member **Group** of the above structure has been declared as a character array of 3 characters, which can take any values.

- However, we know that there are only limited number of groups (or disciplines) offered in our Campus. They include:- {A1, A2, A3, A4, A5, A7, A8, B1, B2, B3, B4, B5, D2}

- Can we do something to restrict the values assigned to the member **Group** to always be from the above set?

- **enum** is the solution!

Enumerator in C

```
typedef enum {A1,A2,A3,A4,A5,A7,A8,B1,B2,B3,B4,B5,D2}
Group_lbl;
```

New definition to struct student:

```
struct student {
    int ID;
    char Name[20];
    Group_lbl Group;
};
```

Now the member variable `Group` can take a value only from the above list.

Each Value inside enum is actually an integer, i.e., `A1=0` , `A2=1` , … and so on. We can check by printing them using `%d`.

We can also assign custom numbers to each of the values of enum, like:-

```
typedef enum {A1=1,A2=2,A3=5,A4=10,…} Group_lbl;
```

Example

```
typedef enum {A1,A2,A3,A4,A5,A7,A8,B1,B2,B3,B4,B5,D2}
Group_lbl;

struct student {
    int ID;
    char Name[20];
    Group_lbl Group;
};
int main(){
    struct student s1 = {876, "Karthik", A1};
    struct student s2;
    s2.ID = 233;
    s2.Name = "Ganesh";
    s2.Group = B5;
    if(s2.Group == B5) printf("s2 is studying Physics");
}
```

innovate achieve lead

BITS Pilani

Pilani Campus

Thank you

Q & A