ACHARYA

Dept. of ISE

ACHARYA

BENGALURU, INDIA

WHERE THE WORLD COMES TO LEARN

ACHARYA

Dept. of ISE

Module-5
STRUCTURE, UNION, EUMERATED DATA TYPES AND FILES

ACHARYA
Dept. of ISE

What is a Structure?

•A **structure** is a user-defined data type in C.
•It allows grouping of variables of **different data types** under a single name.

SYNTAX :-

```
struct Student {
    int id;
    char name[50];
    float marks;
};
```

ACHARYA
Dept. of ISE

What is a Structure?

Declaring Structure variables separately

```
struct Student {
char name[25];
 int age;
char branch[10];
 char gender;
};

struct Student S1, S2; //declaring variables of struct Student
```

Declaring Structure variables with structure definition

struct Student {
char name[25];
int age;
char branch[10];
char gender; } S1, S2; **//declaring variables of struct Student**

Here S1 and S2 are variables of structure Student.
However this approach is not much recommended.

Dept. of ISE

ACHARYA
Dept. of ISE

Declaring and Accessing Structure Variables

Declare structure variables with initialization

```
struct Student s1 = {"Geek", 20, “ISE”, ‘M’};
```

Access members

Use of dot (.) operator.

```
strcpy(s1.name, “Geek");
s1.age = 20;
strcpy(s1.branch, “ISE");
strcpy(s1.gender, ‘M’);
```

ACHARYA

Dept. of ISE

```
#include <stdio.h>
struct Student {
    char name[50];
    int age;
    float grade;
};
int main()
{
struct Student s1 = {"Geek", 20, 85.5};
printf("%s\n", s1.name);
printf("%d\n", s1.age);
printf("%.2f\n", s1.grade);
printf("Size: %d bytes", sizeof(s1));
return 0;
}
```

Output

```
Geek
20
85.50
Size: 60 bytes
```

Explanation: In this example, we create a structure Student to store a student's name, age, and grade. Each of the members (name, age, grade) is stored in its own separate memory location, and we access them individually. The size is also the size of all members combined plus structure padding.

Dept. of ISE

Passing Structure to Functions

By Value

```
void display(struct Student s);
```

By Reference

```
void update(struct Student *s);
```

ACHARYA
Dept. of ISE

```
#include <stdio.h>
struct Book {
    char title[30];
    int pages;
};
void printBook(struct Book b) {
    printf("Title: %s\nPages: %d", b.title, b.pages);
}
int main()
{
    struct Book b1 = {"C Programming", 450};
    printBook(b1);
    return 0;
}
```

ACHARYA
Dept. of ISE

Copying Structures

Two variables of the same structure type can be copied the same way as ordinary variables.
If persona1 and person2 belong to the same structure, then the following statements are valid.

```
person1 = person2;
person2 = person1;
```

C does not permit any logical operators on structure variables. In case, we need to compare them, we may do so by comparing members individually.

```
person1 = =person2
person1 ! = person2
```

The above statements are not permitted.

Example

```
struct class
{
int number; char
name[20]; float marks;
};
main()
{
  int x;
  struct class student1 =
{111,"Rao",72.50};
struct class student2 =
{222,"Reddy",67.00};

struct class student3;

student3 = student2;// copying structures
```

ACHARYA
Dept. of ISE

ACHARYA
Dept. of ISE

```
x = ((student3.number ==
student2.number) &&
(student3.marks ==
student2.marks)) ? 1  : 0;
 if(x == 1)
{
printf("\nstudent2 and
student3 are same\n\n");
printf("%d %s %f\n",
student3.number,
student3.name,student3.mar
ks);
}
else
printf("\nstudent2 and student3 are different\n\n");
}
```

ACHARYA
Dept. of ISE

Nested Structures

- Nested structure in C is nothing but structure within structure.
- One structure can be declared inside other structure as we declare structure members inside a structure.
- The structure variables can be a normal structure variable or a pointer variable to access the data.

ACHARYA
Dept. of ISE

STRUCTURE WITHIN STRUCTURE USING NORMAL VARIABLE:

This program explains how to use structure within structure in C using normal variable. “student_college_detail’ structure is declared inside “student_detail” structure in this program.

Both structure variables are normal structure variables. Please note that members of “student_college_detail” structure are accessed by 2 dot(.) operator and members of “student_detail” structure are accessed by single dot(.) operator.

ACHARYA

Dept. of ISE

```
#include <stdio.h>
#include <string.h>
struct student_college_detail
{
int college_id;
char college_name[50];
};
struct student_detail
{
int id;
char name[20]; float percentage;
// structure within structure
struct student_college_detail clg_data;
}stu_data;
```

ACHARYA

Dept. of ISE

```
int main()
{
 struct student_detail stu_data = {1, "Raju", 90.5, 71145, “AIT"};
 printf(" Id is: %d \n", stu_data.id);
 printf(" Name is: %s \n",
 stu_data.name);
 printf(" Percentage is: %f \n\n", stu_data.percentage);
  printf(" College Id is: %d \n", stu_data.clg_data.college_id);
 printf(" College Name is: %s \n", stu_data.clg_data.college_name);
 return 0;
}
```

OUTPUT:

```
Id is: 1
 Name is : Raju
Percentage is : 90.500000
College Id is : 71145
College Name is : AIT
```

ACHARYA

Dept. of ISE

};

Union

- A **union in C** is similar to a structure, but with a key difference all members of a union share the same memory location.
- This means only one member of the union can store a value at any given time.
- The size of a union is determined by the size of its largest member.

Syntax:

```
union name {
member1 definition;
…………..
memberN definition;
};
```

ACHARYA

Dept. of ISE

};

ACHARYA

Dept. of ISE

```
#include <stdio.h>

union Data {
  int i;
  double d;
 char c;
};

int main() {
      // Create a union variable
union Data data;
// Store an integer in the union
  data.i = 100;
  printf("%d", data.i);
```

};

ACHARYA
Dept. of ISE

```
  // Store a double in the union (this will
 // overwrite the integer value)
  data.d = 99.99;
  printf("%.2f", data.d);

  // Store a character in the union (this will
      // overwrite the double value)
  data.c = 'A';
  printf("%c", data.c);
  printf("Size: %d", sizeof(data));

  return 0;
}
```

ACHARYA
Dept. of ISE

Union in Structure

```
struct Packet {
    int type;  // 0 for ID, 1 for message
    union {
        int id;
        char msg[50];
    } data;
};

int main() {
    struct Packet p1;
    p1.type = 1;
    strcpy(p1.data.msg, "Hello");
    printf("Message: %s", p1.data.msg);
    return 0;
}
```

Since msg overwrites the memory that id was using, accessing id afterwards (p1.data.id) causes **undefined behavior**. The value printed may be garbage.

};

Similarities Between Structure and Union

ACHARYA

Dept. of ISE

•Both are user-defined data types used to store data of different types as a single unit.

•Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.

•Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.

•A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.

•'.' operator or selection operator, which has one of the highest precedences, is used for accessing member variables inside both the user-defined datatypes.

};

ACHARYA

Dept. of ISE

Differences Between Structure and Union

Parameter	Structure	Union
Definition	A structure is a user-defined data type that groups different data types into a single entity.	A union is a user-defined data type that allows storing different data types at the same memory location.
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union
Size	The size is the sum of the sizes of all members, with padding if necessary.	The size is equal to the size of the largest member, with possible padding.
Memory Allocation	Each member within a structure is allocated unique storage area of location.	Memory allocated is shared by individual members of union.
Data Overlap	No data overlap as members are independent.	Full data overlap as members shares the same memory.
Accessing Members	Individual member can be accessed at a time.	Only one member can be accessed at a time.

};

ACHARYA
Dept. of ISE

C Enumeration (enum)

An **enum** is a special type that represents a group of constants (unchangeable values).
To create an enum, use the enum keyword, followed by the name of the enum, and separate the enum items with a comma:

Syntax

```
enum EnumName {CONST1, CONST2, ..., CONSTN};
```

Example:

```
enum Color { RED, GREEN, BLUE } c1;
```

};

C Enumeration (enum)

Default Values

By default, the first name is assigned the value 0, the second is 1, and so on.

Example:

```
enum Days { SUN, MON, TUE, WED, THU, FRI, SAT };
```

ACHARYA

Dept. of ISE

C Enumeration (enum)

Note that the last item does not need a comma.

- It is not required to use uppercase, but often considered as good practice.
- Enum is short for "enumerations", which means "specifically listed".
- To access the enum, you must create a variable of it.
- Inside the main() method, specify the enum keyword, followed by the name of the enum (Level) and then the name of the enum variable (myVar in this example):

```
enum Level myVar;
```

ACHARYA

Dept. of ISE

};

ACHARYA
Dept. of ISE

C Enumeration (enum)

- Now that you have created an enum variable (myVar), you can assign a value to it.
- The assigned value must be one of the items inside the enum (LOW, MEDIUM or HIGH):
- enum Level myVar = MEDIUM;
- By default, the first item (LOW) has the value 0, the second (MEDIUM) has the value 1, etc.
- If you now try to print myVar, it will output 1, which represents MEDIUM:

};

ACHARYA
Dept. of ISE

C Enumeration (enum)

Example 1: Basic Enum Usage

```
#include <stdio.h>
 enum Color { RED, GREEN, BLUE };
 int main() {
    enum Color c = GREEN;
    printf("Color: %d", c);
    return 0;
}
```

};

C Enumeration (enum)

Assigning Custom Values

You can specify your own values.

Example:

```
enum Level { LOW = 10, MEDIUM = 20, HIGH = 30 };
```

Note that if you assign a value to one specific item, the next items will update their numbers accordingly:

```
enum Level {
  LOW = 5,
  MEDIUM, // Now 6
  HIGH // Now 7
};
```

ACHARYA
Dept. of ISE

};

ACHARYA
Dept. of ISE

C Enumeration (enum)

Assigning Custom Values

You can specify your own values.

Example:

```
enum Level { LOW = 10, MEDIUM = 20, HIGH = 30 };
```

Note that if you assign a value to one specific item, the next items will update their numbers accordingly:

```
enum Level {
  LOW = 5,
  MEDIUM, // Now 6
  HIGH // Now 7
};
```

};

ACHARYA

Dept. of ISE

Enum in a Switch Statement

```
enum Level {
  LOW = 1,
  MEDIUM,
  HIGH
};

int main() {
  enum Level myVar = MEDIUM;
```

};

ACHARYA

Dept. of ISE

```
switch (myVar) {
 case 1:
  printf("Low Level");
  break;
 case 2:
  printf("Medium level");
  break;
 case 3:
  printf("High level");
  break;
}
return 0;
}
```

};

Why And When To Use Enums?

- Enums are used to give names to constants, which makes the code easier to read and maintain.

- Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

ACHARYA

Dept. of ISE

};

ACHARYA
Dept. of ISE

C FILES

File Handling

In C, you can create, open, read, and write to files by declaring a pointer of type FILE, and use the fopen() function:

```
FILE *fptr;
fptr = fopen(filename, mode);
```

FILE is basically a data type, and we need to create a pointer variable to work with it (fptr). For now, this line is not important. It's just something you need when working with files.

To actually open a file, use the fopen() function, which takes two parameters:

};

C Files

ACHARYA
Dept. of ISE

Parameter	**Description**
filename	The name of the actual file you want to open (or create), like filename.txt
mode	A single character, which represents what you want to do with the file (read, write or append): w - Writes to a file a - Appends new data to a file r - Reads from a file

};

C Files

ACHARYA

Dept. of ISE

Parameter	Description
filename	The name of the actual file you want to open (or create), like filename.txt
mode	A single character, which represents what you want to do with the file (read, write or append): w - Writes to a file a - Appends new data to a file r - Reads from a file

};

ACHARYA

Dept. of ISE

C Files

Create a File

To create a file, you can use the w mode inside the fopen() function.

The w mode is used to write to a file.

However, if the file does not exist, it will create one for you:

Example

```
FILE *fptr; // Create a file
fptr = fopen("filename.txt", "w");
fclose(fptr); // Close the file
```

Note: The file is created in the same directory as your other C files, if nothing else is specified.

};

C Files

Create a File

To create a file, you can use the w mode inside the fopen() function.

The w mode is used to write to a file.

However, if the file does not exist, it will create one for you:

Example

```
FILE *fptr; // Create a file
fptr = fopen("filename.txt", "w");
fclose(fptr); // Close the file
```

Note: The file is created in the same directory as your other C files, if nothing else is specified.

ACHARYA

Dept. of ISE

ACHARYA

Dept. of ISE

Tip: If you want to create the file in a specific folder, just provide an absolute path (remember to use double backslashes to create a single backslash

```
fptr = fopen("C:\\directoryname\\filename.txt", "w");
```

Closing the file

Did you notice the fclose() function in our example above? This will close the file when we are done with it

ACHARYA
Dept. of ISE

Write To a File

Let's use the w mode from the previous chapter again, and write something to the file we just created.
The w mode means that the file is opened for **writing**. To insert content to it, you can use the fprintf() function and add the pointer variable (fptr in our example) and some text:

Example

```
FILE *fptr;  // Open a file in writing mode
fptr = fopen("filename.txt", "w"); // Write some text to the file
fprintf(fptr, "Some text")
fclose(fptr); // Close the file.
```

ACHARYA
Dept. of ISE

Note: If you write to a file that already exists, the old content is deleted, and the new content is inserted. This is important to know, as you might accidentally erase existing content.
For example:

```
fprintf(fptr, "Hello World!");
```

As a result, when we open the file on our computer, it says "Hello World!" instead of "Some text":

ACHARYA
Dept. of ISE

Append Content To a File

If you want to add content to a file without deleting the old content, you can use the a mode.

The a mode appends content at the end of the file:

Example

```
FILE *fptr;
// Open a file in append mode
fptr = fopen("filename.txt", "a");
// Append some text to the file
fprintf(fptr, "\nHi everybody!");
// Close the file
fclose(fptr);
```

Note: Just like with the w mode; if the file does not exist, the a mode will create a new file with the "appended" content.

ACHARYA
Dept. of ISE

Read a File

In the previous chapter, we wrote to a file using w and a modes inside the fopen() function.

To **read** from a file, you can use the r mode:

Example

```
FILE *fptr;

// Open a file in read mode
fptr = fopen("filename.txt", "r");
```

This will make the filename.txt opened for reading.

ACHARYA
Dept. of ISE

Next, we need to create a string that should be big enough to store the content of the file.
For example, let's create a string that can store up to 100 characters:

Example

```
FILE *fptr;

// Open a file in read mode
fptr = fopen("filename.txt", "r");
// Store the content of the file
char myString[100];
```

ACHARYA
Dept. of ISE

the fgets() function.
The fgets() function takes three parameters:

Example
fgets(myString, 100, fptr);

1.The first parameter specifies where to store the file content, which will be in the myString array we just created.
2.The second parameter specifies the maximum size of data to read, which should match the size of myString (100).
3.The third parameter requires a file pointer that is used to read the file (fptr in our example).
Now, we can print the string, which will output the content of the file:

ACHARYA
Dept. of ISE

Example

```
FILE *fptr;
// Open a file in read mode
fptr = fopen("filename.txt", "r");
// Store the content of the file
char myString[100];
// Read the content and store it inside myString
fgets(myString, 100, fptr);
// Print the file content
printf("%s", myString);
// Close the file
fclose(fptr);
```

ACHARYA
Dept. of ISE

Example

```
FILE *fptr;
// Open a file in read mode
fptr = fopen("filename.txt", "r");
// Store the content of the file
char myString[100];
// Read the content and print it
while(fgets(myString, 100, fptr)) {
  printf("%s", myString);
}
// Close the file
fclose(fptr);
```

ACHARYA
Dept. of ISE

Example

```
FILE *fptr;

// Open a file in read mode
fptr = fopen("loremipsum.txt", "r");

// Print some text if the file does not exist
if(fptr == NULL) {
  printf("Not able to open the file.");
}

// Close the file
fclose(fptr);
```

ACHARYA
Dept. of ISE

Example

If the file exist, read the content and print it. If the file does not exist, print a message:

```
FILE *fptr;
fptr = fopen("filename.txt", "r"); // Open a file in read mode
// Store the content of the file
char myString[100];
// If the file exist
if(fptr != NULL) {
    while(fgets(myString, 100, fptr)) {
    printf("%s", myString);
  }
// If the file does not exist
} else {
  printf("Not able to open the file.");
}
fclose(fptr); // Close the file
```