Unit 5

- Structure
- Union
- Dynamic memory allocation
- File handling

Mohd Adil

Assistant Professor

Dayananda Sagar College of Engineering

Structure in C

- A Structure is a user defined data type that can be used to group elements of different types into a single type!

For example:

```
Struct student {
char name[10];
char course[5];
Int year;
} student 1, student 2;
```

Structure in C

```
#include <stdio.h>
struct student {
   char name[10];
} student1 = {"xyz"}, student2 = {"abc"};

int main() {

   printf("%s\n", student1.name);
   printf("%s", student2.name);

   return 0;
}
```

Initializing structure variables and accessing members of structure

```
struct book {
   char title[50];
   char author[30];
   int pages;
   float price;
};
```

```
int main() {
   struct book b1 = {"Introduction to C
Programming", "Dennis Ritchie", 250,
399.50};
   struct book b2 = {"Data Structures in C",
"Reema Thareja", 480, 550.00};
   return 0;
}
```

Two books with same properties but different values.

Accessing Members of structure

- We can access members of the structure using dot (.) operator.

```
struct car {
int fuel_tank_cap;
} c1, c2;
int main() {
c1.fuel_tank_cap = 45;
c2.fuel_tank_cap = 30;
printf("%d %d", c1.fuel_tank_cap, c2.fuel_tank_cap);
return 0;
}
```

Declaring an array of structure

- Instead of declaring multiple variables, we can also declare an array of structure in which each element of the array will represent a structure variable.

Example

```
#include <stdio.h>
#include <string.h>
struct book {
   char title[50];
   char author[30];
   int pages;
   float price;
};
```

```
struct book library[3];
   for(int i = 0; i < 3; i++) {
      printf("Enter title of book %d: ", i+1);
      scanf("%s", library[i].title);
      printf("Enter author of book %d: ", i+1);
      scanf("%s", library[i].author);
      printf("Enter pages of book %d: ", i+1);
      scanf("%d", &library[i].pages);
      printf("Enter price of book %d: ", i+1);
      scanf("%f", &library[i].price);
      printf("\n");
   }
```

Conti……

```
printf("\n----- Book Details -----\n");
  for(int i = 0; i < 3; i++) {
     printf("Book %d:\n", i+1);
     printf("Title: %s\n", library[i].title);
     printf("Author: %s\n", library[i].author);
     printf("Pages: %d\n", library[i].pages);
     printf("Price: %.2f\n\n", library[i].price);
  }

  return 0;
}
```

Accessing members of structure using structure pointer

```
struct abc {
int x;
int y;
};
int main() {
struct abc a = {0, 1};
struct abc *ptr = &a;
printf("%d %d", ptr->x, ptr->y);
return 0;
}
```

Introduction to unions

Union is a user defined data type but unlike structures, union members share same memory location.

Example:

```
struct abc {
int a;
char b;
};
```

a's address = 6295624

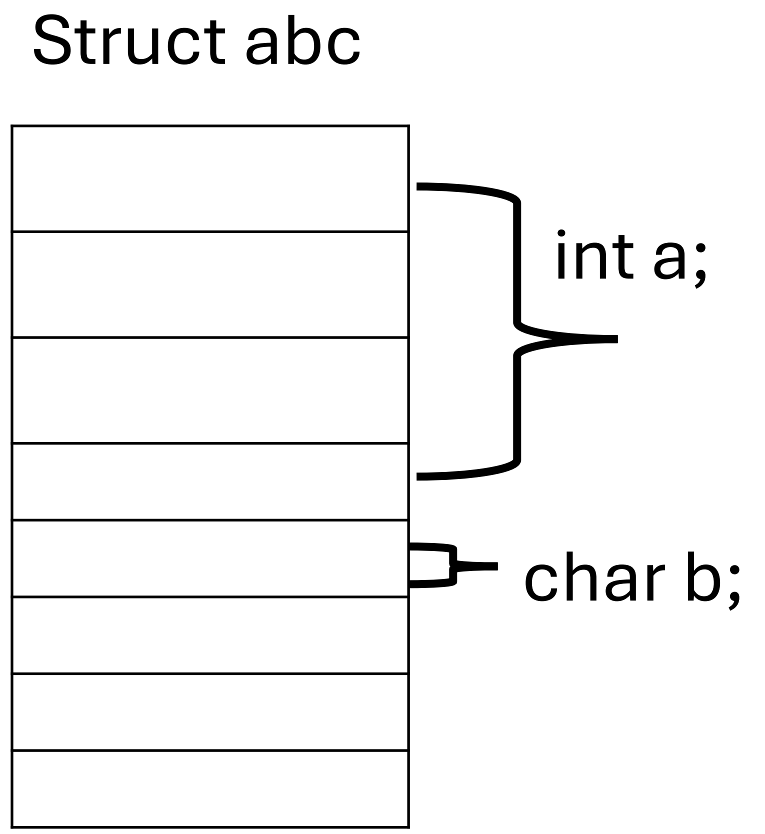
b's address = 6295628

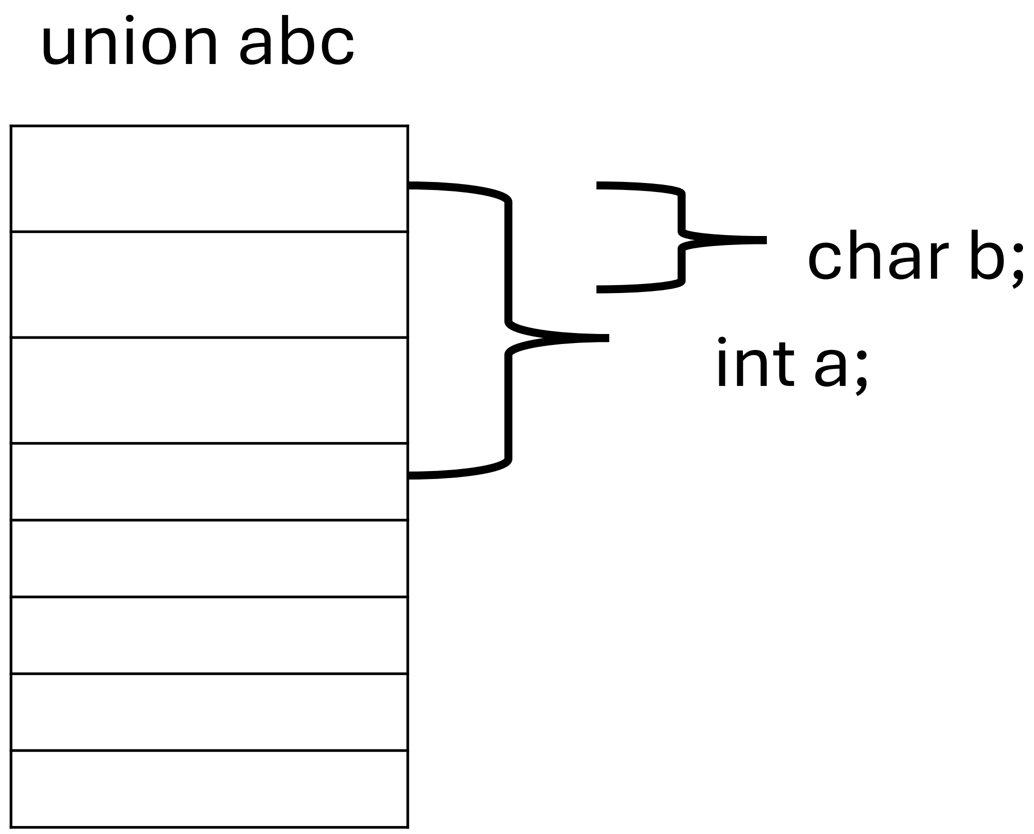
```
union abc{
int a;
char b;
};
```

a's address = 6295616

b's address = 6295616

Memory Allocation





- In union, members will share same memory location. If we make changes in one member then it will be reflected to another member as well.

Example:

```
union abc {
int a;
char b;
}var;
```

```
int main()
 {
var.a = 65;
printf("a =%d\n", var.a);
printf("b=%c", var.b);
return 0;
}
```

Accessing members using pointers

- We can access members of union through pointers by using the arrow (->)operator.

```
union abc {
int a;
char b;
};
```

```
int main()  {
union abc var;
var.a = 90;
union abc *p = &var;
printf("%d %c", p->a, p->b);
return 0;
  }
```

Introduction Memory Allocation

Static Memory Allocation

Memory allocated during compile time is called static memory.

The memory allocated is fixed and cannot be increased or decreased during run time.

EXAMPLE:

```
int main(){
int arr[5] = {1, 2, 3, 4, 5};
}
```

Memory is allocated at compile time and is fixed

Introduction Memory Allocation

If you are allocating memory for an array during compile time, then you must fix the size at the time of declaration.

Size is fixed and user cannot increase or decrease the size of the array at run time.

If the values stored by the user in the array at run time is less than the size specified, then there will be wastage of memory.

If the values stored by the user in the array at run time is more than the size specified, then the program may crash or misbehave.

Dynamic Memory Allocation

The process of allocating memory at the time of execution is called dynamic memory allocation.

POINTERS PLAY AN IMPORTANT ROLE IN DYNAMIC MEMORY ALLOCATION.

ALLOCATED MEMORY CAN ONLY BE ACCESSED THROUGH POINTERS.

Dynamic Memory Allocation

BUILT IN FUNCTIONS:

1. malloc()
2. calloc()
3. realloc()
4. free()

What is malloc()

malloc is a built-in function declared in the header file <stdlib.h>

malloc is the short name for "memory allocation" and is used to dynamically allocate a single large block of contiguous memory according to the size specified.

Syntax (void*) malloc(size_t size)

malloc function simply allocates a memory block according to the size specified in the heap and on success it returns a pointer pointing to the first byte of the allocated memory else returns NULL.

Why void pointer

malloc doesn't have an idea of what it is pointing to.

It merely allocates memory requested by the user without knowing the type of data to be stored inside the memory.

The void pointer can be typecasted to an appropriate type.

```
int *ptr=(int*)malloc(4)
```

malloc allocates 4 bytes of memory in the heap and the address of the first byte is stored in the pointer ptr

Example

```
#include <stdio.h>
#include <stdlib.h>
int main() {
int*ptr=(int*)malloc(5*sizeof(int));
if(ptr == NULL) {
printf("Memory not available.");
exit(1);
}
for(i=0; i<5; i++) {
ptr[i]=i;
}
for(i=0; i<5; i++)
printf("%d ",p[i]);
return 0;
}
```

What is calloc() -> clear Allocation

calloc() function is used to dynamically allocate multiple blocks of memory.

It is different from malloc in two ways:

calloc() needs two arguments instead of just one

Syntax: (void*) calloc(size_t n , size_t size);

number of blocks

size of each block

What is Realloc()

realloc() function is used to change the size of the memory block without losing the old data.

Syntax: void *realloc(void *ptr , size_t newsize);

Pointer to the previously allocated memory.

New size

on failure realloc return Null

What is Realloc()

Example:

```
int *ptr = (int *)malloc(sizeof(int));
ptr = (int *)realloc(ptr, 2*sizeof(int));
```

- This will allocate memory space of 2*sizeof(int).
- Also, this function moves the contents of the old block to a new block, and the data of the old block is not lost.
- We may lose the data when the new size is smaller than the old size.
- Newly allocated bytes are uninitialized.

What is free()

free() function is used to release the dynamically allocated memory in heap.

SYNTAX: void free(ptr)

The memory allocated in heap will not be released automatically after using the memory. The space remains there and can't be used.

It is the programmer's responsibility to release the memory after use.

Introduction to file handling

What is a Stream?

Before understanding files, we must understand *streams*.

Stream = Flow of data

A **stream** is simply a **flow of data** between your C program and an input/output device.

Types of Streams in C

1. Input Stream

- Data flows *from* keyboard *to* program
- **scanf()**

2. Output Stream

- Data flows *from* program *to* screen
- **printf()**

Introduction to file handling

What is a File?

A file is a collection of data or information that has a name called as filename

- File = Data stored on disk (permanent storage)
- A **file** is a place where we can store data **permanently** on secondary storage like HDD or SSD.
- program uses **streams** to send/receive data from files.

Accessing the file in c

Accessing a stream in c program is done through file pointers.

- File pointers can be declared using the file type **FILE ***
- **Note:** file type is declared in <stdio.h> header file

If a program needs a stream in addition to the standard ones than

We can declare a FILE pointer to access the stream like following.

```
FILE  *fp1  ,  fp2;
```

File operations (opening & closing a file)

Opening a file

Syntax:

FILE * fopen(const char* filename , const char * mode);

fopen returns a pointer to a file (the file whose filename is specified which we can store in a pointer for later use

Example

FILE* fp;

fp=open(“filename”, “mode);

Mode string specifies what operations we want to perform on the file.
Not only that , it also defines whether a file contain text data or binary data.

File operations (opening & closing a file)

File opening modes

Modes	**Meaning of mode**	**If file doesn’t exist then**
r	**Open for reading**	**fopen() return NULL**
w	**Open for writing**	**If the file exists then its contents are overwritten else file will be created.**
a	**Open for append**	**File will be created**

File operations (opening & closing a file)

File opening modes

Modes	Meaning of mode	If file doesn’t exist then
r+	Open for both reading and writing	fopen() return NULL
w+	Open for both reading and writing	If the file exists then its contents are overwritten else file will be created.
a+	Open for both reading and appending	File will be created

File operations (opening & closing a file)

File opening modes

Modes	**Meaning of mode**	**If file doesn’t exist then**
rb	**Open for reading in binary mode**	**fopen() return NULL**
wb	**Open for writing in binary mode**	**If the file exists then its contents are overwritten else file will be created.**
ab	**Open for appending In binary mode**	**File will be created**

File operations (opening & closing a file)

Closing file

A file needs to be closed after performing operations on it.

fclose() is used to close a particular file

Syntax:

Int fclose (FILE * pointer)

Returns zero if the file is successfully closed else return EOF on failure

File operations (writing to text file using fprintf)

Steps involved:

1. Create a pointer to file (FILE pointer)
2. Open the text file in write mode and store the return address in file pointer.
3. Test the file whether or not the file pointer is null.
4. Write something on the file using fprintf() function
5. Close the file.

fprintf()

Fprintf() function is used to print content in the file just like printf()

Function which prints the content on the console window.

Syntax:

fprintf(FILE *fptr, const char * str)

Example:

Using fprintf() function

```
#include<stdio.h>
Int main(){
FILE *fptr;
char str[100];
fptr=fopen(“file1.txt”, “w”);
if(fptr==NULL)
                printf(“Error!”);
printf(“Enter a string”);
scanf(“%s”,str);
fprintf(fptr , “%s”,str);
fclose(fptr);
}
```

Fputc-> function is used to write single character at a time to a file

```
Int main(){
FILE *fptr=fopen(“output.txt”, “w”);
int i;
char str[]=“C programming language”;
if(fptr==NULL)
                printf(“Error!”);
for(i=0;str[i]!=‘\0’;i++)
                fputc(str[i], fptr);
fclose(fptr);
return 0;
}
```

File operations (Reading from text file)

Steps involved:

1. Create a pointer to file(FILE pointer)
2. Open the text file in **read mode** and store the return address in file pointer.
3. Test whether or not the file pointer is Null.
4. Read the file using **fscanf** function.
5. Close the file

```
int main(){
FILE *fptr=fopen(“file2.txt”, “r”);
char str[100];
If(fptr==NULL)
        printf(“Error!”);
fscanf(fptr, %s”,str);
printf(“%s”,str);
fclose(fptr);
return 0;
}
```

File operations (Reading from text file)

fgetc()

fgetc is used to read single character at a time from a file

Syntax:

int fgetc(FILE* pointer);

FILE* pointer –specify the pointer to a file on which the operation is to performed

Returns a character

Example:

```
#include<stdio.h>
int main(){
FILE *fptr=fopen(“input.txt”, “r”);
char ch;
while(1)
{
ch=fgetc(fptr);
If(ch==EOF)
        break;
printf(“%c”,ch);
}
fclose(fptr);
return 0;
}
```

File operations (Reading from text file)

fgets()

fgets read a string from the specified stream. It stops when either the eof is reached (n-1) character are read or the newline character is read.

Syntax:

char* fgets(char *str,int n, FILE* pointer);

```
int main(){
FILE *fptr=fopen(“input.txt”, “r”);
char str[100];
fgets(str,10,fptr)//Reading only 9 ch
printf(“%s”,str);
fclose(fptr);
return 0;
}
```

File operations (r+ and w+ modes)

Both are used to open the file for reading and writing at the same time.

In the traditional way we first open the file for writing the data into it.

Then we have to close the file and re open it again and in order to read the contents of the file

```
int main(){
char *str=“c programming”;
char str1[100];
FILE *fptr=open(“file2.txt”, “r+);
if(fptr==NULL)
		printf(“Could not open file!”);
fputs(str,fptr);
rewind(fptr);
fgets(str1,100,fptr);
fclose(fptr);
printf(“%s”,str1);
return 0;
}
```

File operations (a+ modes)

a+ mode opens the file for both reading and appending (writing to the end of the file).

Also, if the file does not exist then it will be created else the new content will be appended at the end of the file

Noted that initially the pointer will be at the beginning of the file but the output will always be appended at the end of the file.

Example :

```
#include<stdio.h>
int main(){
FILE *fptr=fopen(“output.txt” ,
“a+”);
char str[100];
char *s= “is useful for beginners”;
fputs(s,fptr);
rewind(fptr);
fgets(str,100,fptr);
printf(“%s”,str);
fclose(fptr);
return 0;
}
```