**DAY-13**

**UNIONS**

* Unions are same as structures.
* But the difference is only one member can be accessed at a time in union.
* The size of the union is the highest datatype size but not the sum of all datatypes as only one member can be accessed at a time, the memory is sufficient to allocate other datatype members.

E.g.: In game at at time only one operation done: jump, run

* We will send Login details as a structure (Username and Password).
* In client server model when we send a connection handshake signals are there between. When we try to get transaction details. Only that part is sent but not all. SO here union is used to send only one part

YouTube load balancing

Typedef Struct EMP

{

Int id;

Float sal;

Char g;

Char Name[20];

}sEMP;

Typedef Union uEmployee

{

Int id;

int sal;

Char g;

Char Name[20];

}uEMP;

Int main()

{

sEMP es;

uEMP eu;

Printf(“\n size of union = %d”,sizeof(eu));

Printf(“\n size of structure = %d”,sizeof(es));

Printf(“\n\n”);

eu.id = 101;

eu.sal= 10000

printf(“\n%d”,eu.id); /// Here the sal is overridden and the output is 10000

eu.id = 101;

eu.sal= 10000;

strcpy(eu.Name,”Bhima”);

printf(“\n%d”,eu.id) /// Here it prints the ascii values

eu1= (uEMP \*)malloc(sizeof(uEMP));

eu1 ->id = 102;

printf(“%d”,eu1->id);

**🡪Enumerated**

Set of named integers. The values are given names

It is useful in a convention way i.e in a way where only names are used for numbers

Enum enum-type-name

{

Enumeration list

}variable\_list;

E.g.:

Enum daysOfWeek

{

Monday=1000, Tuesday,Wednesday,

Thursday=4005, Friday, Saturday,

Sunday

} Days;

Int main()

{

Days=Monday;

Printf(“\n Day= %d”,Days);

Return0;

}

Enum RegClrCodes

{

Black, Brown, Red, Orange,

Yellow, Green, Blue, Violet, Gray, White

} RCC;

int main()

{

Scanf(“%d”,&n);

Switch(n)

{

Case Black: printf(“Black);

Break;

Case Brown: printf(“Brown”);

Break;

}

}

In switch case instead of number we can use strings

**Files**

* The ram memory will get cleared when we switch off and on
* FILE is a data structure that has all contains like name of file, date and time modified

Two types of Files:

1. Sequential File : All records are arranged in order

2. Random Access: Files are at random

* In C, there is no concept of sequential or indexed file reading. So we can write/ read from anywhere in the file

FILE \*fp;

FILE \*fopen(char\*name, char \* mode);

fp= fopen(“file name”, “mode”);

* r Open text file for reading. The stream is positioned at the beginning of the file.
* r+ Open for reading and writing. The stream is positioned at the beginning of the file.
* w Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
* w+ Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
* a Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
* a+ Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file. Fopen will return the address of the file
* fseek (fp, long offset, int whence)èto move to a particular point of a file

if whence(start position) = SEEK\_SET, SEEK\_CUR, or SEEK\_END, offset is start of file

The working is if we give (fp,10,SEEK\_SET)🡪we move 10 char

(fp,0,SEEK\_END)🡪 start to end of file, moves to end of file

(fp,-10,SEEK\_CUR)->We can give negative values also for offset

ftell(fp) èThe position where the file pointer is

fclose(fp)

FILE 🡪 Data structure

* fd 🡪file pointer to file data structure
* fopen(“Name of file”, “mode”) 🡪 open file
* fclose(fd)🡪 close opened file
* fprintf, fscanf 🡪 formatted writing and reading resp.
* fputs,fgets 🡪unformatted i/o operation
* fwrite, fread 🡪 reading/writing binary objects
* size\_t fread (void \* ptr, size\_t size, size\_t nmemb, FILE \*stream);

The ptr stores the file read, size gives the number of bytes to be read and nmem tells number of times from file and returns the number of bytes read

size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream);

fread does not distinguish to EOF so callers must use feof()

int main()

{

FILE \*fd=NULL;

char c;

fd = fopen("Test1.txt","r");

if(fd==NULL)

{

perror("fopen: ");

printf("\n%p\n",fd);

exit(1);

}

printf("Opened the file successfully %p \n",fd);

while((c=fgetc(fd))!=EOF)

putchar(c);

printf("\n");

fclose(fd);

return 0;

}

**To copy contents from one file to another**

fd = fopen("Test2.txt","w");

if(fd==NULL)

{

perror("fopen: ");

printf("\n%p\n",fd);

exit(1);

}

fd2 = fopen("Test1.txt","r");

if(fd2==NULL) /// to check if file is created or not. Directory permissions

{

perror("fopen: ");

printf("\n%p\n",fd2);

exit(1);

}

printf("Opened the file successdully %p \n",fd);

while((c=fgetc(fd2))!=EOF)

{

putchar(c);

fputc(c,fd);

}

**To write a string into file**

FILE \*fd=NULL;

Char Name[]= “Anjalli”

for(i=0;i<strlen(Name);i++)

fputc(Name[i],fd);

**To read a string from file**

Char rName[20];

FILE \*fd=NULL;

i=0;

while((c=fgetc(fd2))!=EOF)

{

rName[i]=c;

i++;

}

rName[i]='\0';

To clear the screen : system(“clear”);

* Execl 🡪to execute the commands