JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8251 – C PROGRAMMING

(Regulation 2017)

Year/Semester: I/II CSE

2020 – 2021

Prepared by

Mr.H.Shine

Assistant Professor/CSE

UNIT -V

Union is derived data type contains collection of different data type or dissimilar elements. All definition declaration of union variable and accessing member is similar to structure, but instead of keyword struct the keyword union is used, the main difference between union and structure .

Each member of structure occupy the memory location, but in the unions members share memory. Union is used for saving memory and concept is useful when it is not necessary to use all members of union at a time.

Where union offers a memory treated as variable of one type on one occasion where (struct), it read number of different variables stored at different place of memory.

Syntax of union:

union student

{

datatype member1;

datatype member2;

};

Like structure variable, union variable can be declared with definition or separately such as

union union name

{

Datatype member1;

}var1;

Example:- union student s;

Union members can also be accessed by the dot operator with union variable and if we have pointer to union then member can be accessed by using (arrow) operator as with structure.

Example:- struct student

```
struct student
{
int i;
char  ch[10];
};struct student s;
```

Here datatype/member structure occupy 12 byte of location is memory, where as in the union side it occupy only 10 byte.

Nested of Union

When one union is inside the another union it is called nested of union.

Example:-

```
union a
{
int i;
int age;
};
union b
{
char name[10];
union a aa;
```

}; union b bb;

There can also be union inside structure or structure in union.

Example:-

```
 void  main()
   {
  struct a
 {
int i;
char ch[20];
};
struct  b
{
int i;
char d[10];
};
union z
{
struct a a1;
struct b b1;
};
 union z z1;
z1.b1.j=20;
```

z1.a1.i=10;

z1.a1.ch[10]= “ i“;

z1.b1.d[0]=”j “;

printf(“ “);

Dynamic memory Allocation

The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location.

Two types of problem may occur in static memory allocation.

If number of values to be stored is less than the size of memory, there would be wastage of memory.

If we would want to store more values by increase in size during the execution on assigned size then it fails.

Allocation and release of memory space can be done with the help of some library function called dynamic memory allocation function. These library function are called as **dynamic memory allocation function.** These library function prototype are found in the header file, “alloc.h” where it has defined.

Function take memory from memory area is called heap and release when not required.

Pointer has important role in the dynamic memory allocation to allocate memory.

malloc():
This function use to allocate memory during run time, its declaration is void*malloc(size);

malloc ()

returns the pointer to the 1st byte and allocate memory, and its return type is void,

which can be type cast such as:

int *p=(datatype*)malloc(size)

If memory location is successful, it returns the address of the memory chunk that was allocated and it returns null on unsuccessful and from the above declaration a pointer of type**(datatype)** and size in byte.

And **datatype** pointer used to typecast the pointer returned by malloc and this typecasting is necessary since, malloc() by default returns a pointer to void.

Example int*p=(int*)malloc(10);

So, from the above pointer p, allocated IO contigious memory space address of 1st byte and is stored in the variable.

We can also use, the size of operator to specify the the size, such as *p=(int*)malloc(5*size of int) Here, 5 is the no. of data.

Moreover , it returns null, if no sufficient memory available , we should always check the malloc return such as, **if(p==null)**

printf(“not sufficient memory”);

Example:

/*calculate the average of mark*/

void main()

{

int n , avg,i,*p,sum=0;
printf("enter the no. of marks ”);

scanf(“%d”,&n);

p=(int *)malloc(n*size(int));

if(p==null)

```
printf(“not sufficient”);
exit();
}
for(i=0;i<n;i++)
scanf(“%d”,(p+i));
for(i=0;i<n;i++)
Printf(“%d”,*(p+i));
sum=sum+*p;
avg=sum/n;
printf(“avg=%d”,avg);
```

2nd argument specify

size of each block.

Example:-

```
int *p= (int*) calloc(5, 2);
int*p=(int *)calloc(5, size of (int));
```

Another difference between malloc and calloc is by default memory allocated by malloc contains garbage value, where as memory allocated by calloc is initialised by zero(but this initialisation) is not reliable.

realloc()

The function realloc use to change the size of the memory block and it alter the size of the memory block without loosing the old data, it is called reallocation of memory.

It takes two argument such as;

int *ptr=(int *)malloc(size);

int*p=(int *)realloc(ptr, new size);

The new size allocated may be larger or smaller.

If new size is larger than the old size, then old data is not lost and newly allocated bytes are uninitialized. If old address is not sufficient then starting address contained in pointer may be changed and this reallocation function moves content of old block into the new block and data on the old block is not lost.

Example:

```
#include<stdio.h>

#include<alloc.h>

void main()

int i,*p;
p=(int*)malloc(5*size of (int));

if(p==null)

{

printf(“space not available”);

exit();

printf(“enter 5 integer”);

for(i=0;i<5;i++)

{

scanf(“%d”,(p+i));

int*ptr=(int*)realloc(9*size of (int) );

if(ptr==null)

{
```

```
printf(“not available”);
exit();
}
printf(“enter 4 more integer”);
for(i=5;i<9;i++)
scanf(“%d”,(p+i));
for(i=0;i<9;i++)
                printf(“%d”,*(p+i));
}
```

free()

Function free() is used to release space allocated dynamically, the memory released by free() is made available to heap again. It can be used for further purpose.

Syntax for free declaration .

void(*ptr)

Or

free(p)

When program is terminated, memory released automatically by the operating system. Even we don’t free the memory, it doesn’t give error, thus lead to memory leak.

We can’t free the memory, those didn’t allocated.

Dynamic array

Array is the example where memory is organized in contiguous way, in the

dynamic memory allocation function used such as malloc(), calloc(), realloc() always made up of contiguous way and as usual we can access the element in two ways as:

Subscript notation

Pointer notation

Example:

```
#include<stdio.h>
#include<alloc.h>
void main()
{
printf("enter the no.of values");
scanf("%d",&n);
p=(int*)malloc(n*size of int);
If(p==null)
printf("not available memory");
exit();
}
for(i=0;i<n;i++)
{
printf("enter an integer");
scanf("%d",&p[i]);
for(i=0;i<n;i++)
{
```

```
printf("%d",p[i]);
}
}
```

File handling

File: the file is a permanent storage medium in which we can store the data permanently.

Types of file can be handled

we can handle three type of file as

(1) sequential file

(2) random access file

(3) binary file

File Operation

opening a file:

Before performing any type of operation, a file must be opened and for this fopen() function is used.

syntax:

file-pointer=fopen("FILE NAME ","Mode of open");

example:

```
FILE *fp=fopen("ar.c","r");
```

If fopen() unable to open a file than it will return NULL to the file pointer.

File-pointer: The file pointer is a pointer variable which can be store the address of a special file that means it is based upon the file pointer a file gets opened.

Declaration of a file pointer:-

FILE* var;

Modes of open

The file can be open in three different ways as

Read mode ' r'/rt

Write mode 'w'/wt

Appened Mode 'a'/at

Reading a character from a file

getc() is used to read a character into a file

Syntax:

character_variable=getc(file_ptr);

Writing acharacter into a file

putc() is used to write a character into a file

puts(character-var,file-ptr);

ClOSING A FILE

fclose() function close a file.

fclose(file-ptr);

fcloseall () is used to close all the opened file at a time

File Operation

The following file operation carried out the file

(1)creation of a new file

(3)writing a file

(4)closing a file

Before performing any type of operation we must have to open the file.c, language communicate with file using A new type called **file pointer**.

Operation with fopen()

File pointer=fopen("FILE NAME","mode of open");

If **fopen()** unable to open a file then it will return **NULL** to the file-pointer.

Reading and writing a characters from/to a file

fgetc() is used for reading a character from the file

Syntax:

character variable= fgetc(file pointer);

fputc() is used to writing a character to a file

Syntax:

fputc(character,file_pointer);

/*Program to copy a file to another*/

#include<stdio.h>

void main()

```
{
FILE *fs,*fd;
char ch;
If(fs=fopen(“scr.txt”,”r”)==0)
{
printf(“sorry….The source file cannot be opened”);
return;
}
If(fd=fopen(“dest.txt”,”w”)==0)
{
printf(“Sorry…..The destination file cannot be opened”);
fclose(fs);
return;
}
while(ch=fgets(fs)!=EOF)
fputc(ch,fd);
fcloseall();
}
```

Reading and writing a string from/to a file

getw() is used for reading a string from the file

Syntax:

gets(file pointer);

putw() is used to writing a character to a file

Syntax:

fputs(integer,file_pointer);

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
FILE *fp;
int word;
/*place the word in a file*/
fp=fopen(“dgt.txt”,”wb”);
If(fp==NULL)
{
printf(“Error opening file”);
exit(1);
}
word=94;
putw(word,fp);
If(ferror(fp))
printf(“Error writing to file\n”);
else
printf(“Successful write\n”);
fclose(fp);
/*reopen the file*/
```

```
fp=fopen(“dgt.txt”,”rb”);
If(fp==NULL)
{
printf(“Error opening file”);
exit(1);
}

/*extract the word*/
word=getw(fp);
If(ferror(fp))
printf(“Error reading file\n”);
else
printf(“Successful read:word=%d\n”,word);
/*clean up*/
fclose(fp);
```

Reading and writing a string from/to a file

fgets() is used for reading a string from the file

Syntax:

fgets(string, length, file pointer);

fputs() is used to writing a character to a file

Syntax:

fputs(string,file_pointer);

```
#include<string.h>
```

```
#include<stdio.h>
void main(void)
{
FILE*stream;
char string[]=”This is a test”;
char msg[20];
/*open a file for update*/
stream=fopen(“DUMMY.FIL”,”w+”);

/*write a string into the file*/
fwrite(string,strlen(string),1,stream);
/*seek to the start of the file*/
fseek(stream,0,SEEK_SET);
```