UNIT-V MEMORY MANAGEMENT

Contents

DMA Functions: mallaoc(), calloc(), sizeof(), free() and realloc(). Preprocessor Directives. File Management: File operations – Opening and Closing a File, Input Statements, Output Statements and Control Statements.

Static and Dynamic Memory Allocation in C

It is the procedure to allocate memory cells to the variables. There are two memory allocation methods. They are,

- Static memory allocation
- Dynamic memory allocation

Static Memory Allocation

The process of allocating memory space at compile time is known as static memory allocation. In static memory allocation size is fixed.

Example: Arrays

```
int a[5];
```

The above declaration of an array allocates spaces for storing 5 int**Example**ers in the memory. Static Memory allocation has the following Disadvantages:

- Size is not expandable
- Insertion is difficult(Time consuming process)
    - It requires a lot of data movement taking more time.
- Deletion is also difficult.

Dynamic Memory Allocation

The process of allocating memory space at run time known as dynamic memory allocation. In dynamic memory allocation size (no of cells is not allocated) is not fixed.

Example: Linked List

Linked List:

A linked list is collection of more than one nodes linked together. Each node has an element. A linked list has the following characteristics

- To store each element, we use a node.
- A node consists of data (elements) and link field.
- If there is only one link field, it is called as singly linked list.

- If it has 2 links, it is doubly linked list.
- The nodes need not be stored continuously in the memory.

The advantages of linked list are

- Size is not fixed.
- Insertion is easy which does not require any data movement.
- Deletion is also easy.

Dynamic Memory Allocation Functions: (malloc(), calloc(), sizeof(), free() and realloc())

DMA predefined functions are available in c library. These are used to allocating and reallocating the memory space in memory.DMA functions are available through the header file is stdlib.h and alloc.h.so u must include this library in order to use them.

(i) malloc( )

This function is used to allocate memory dynamically. Use the malloc function to allocate a single block of memory space of variable in specified size.

If there is not enough memory available it will return NULL.

Syntax:

```
pointer_variable =(type cast*)malloc(size in bytes);
```

typecast is a datatype. It will allocate the memory space with size of byte.

Example1: `a = (char*)malloc(10);`

It will occupies10 bytes memory and assign of first byte to a.

Example2: `a=(int*)malloc(40*sizeof(int));`

Size of pointer is how many bytes it takes to store the pointer, not the size of the memory block the pointer is pointing at.

(ii) realloc( )

It is necessary to alter the previously allocated memory. i.e.,to add additional memory or to reduce as and when required. Before using this statement,the user must allocate some memory previously by using the malloc( ) and calloc( )function.

Syntax:

```
pointer_variable =realloc(pointer variable , new size);
```

Example : Program to altering the allocated memory.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
main()
```

```
{
char  *p;
clrscr( );
p=(char*)malloc(6);
strcpy(p,”MADRAS”);
printf(“memory contains:%s\n”,p);
p=(char*)realloc(p,7);
strcpy(p,”CHENNAI”);
printf(“memory contains:%s\n”,p);
free(p);
 getch( );
}
```

Output:

```
memory contains : MADRAS
memory contains :CHENNAI
```

(iii) calloc( )

This function is used to allocates multiple block of memory space of specified size and each block contains same size and initializes them with zeros.

Syntax:

```
pointer_variable =(type cast*)calloc(n,elementsize);
```

Example

```
a = ( int *)calloc(10,sizeof(int)*2);
```

It occupies 10 blocks each of the 4 bytes memory and assign the address of first byte of fist block top.

Example Program:

```
 #include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int *p = (int*)calloc(2,sizeof(int)*2);
clrscr();
for(int i=0;i<4;i++)
  p[i]=i;
  printf("Memory Allocated Dynamically for 2 block each has 4 bytes");
  for(i=0;i<4;i++)
     printf("%d",p[i]);
   realloc(p,sizeof(int)*6);
  for( i=0;i<6;i++)
    p[i]=i;
  printf("Memory Allocated Dynamically for 2 block each has 6 bytes");
 for( i=0;i<6;i++)
    printf("%d",p[i]);
 getch();
 }
```

sizeof( )

It is an operator to find the size of the data-type or variable in terms of bytes.

Syntax:

int x=sizeof(data_type/variable);

Example : Program Depicting the Use of Function **sizeof( )**in C Programming

```
#include<stdio.h>
#include<conio.h>
void main( )
{
  char p;
  clrscr( );
  printf("%d", sizeof(p));
  getch();
}
```

Output:

```
1
```

Explanation: We know that a character variable takes 1 byte memory. Here, *p* is a character variable and **sizeof(p)** is 1(byte).

Example : Program Depicting the Use of Function **sizeof()**in C Programming

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int  p[5];
  clrscr();
  printf("%d", sizeof(p));
  getch();
}
```

Output:

```
10
```

Explanation: We know that a int**Example**er occupies 2 bytes in the memory. Here, *p* is a array of 5 elements and **sizeof(p)** is 10 bytes (5 elements X 2 byes each = 10 bytes).

(iv) free( )

It is used to delete an existing memory space

Syntax: free(p);

Example: Sum of array elements using pointer

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void main()
{
   int *a,i,n,result=0;
  clrscr();
   printf("Enter the no. of elements to be stored in an
array :");
   scanf("%d",&n);
   a=(int *)malloc(n*sizeof(int));
   printf("\nEnter the elements :");
   for(i=0;i<n;i++)
   scanf("%d",a+i);
   for(i=0;i<n;i++)
   result+=*(a+i);
printf("\nThe sum of elements in an array is :%d",result);
   free(a);
  getch();
}
```

Output:

```
Enter the no. of elements to be stored in an array: 5
Enter the elements:  2         20         25        35
18
The sum of elements in an array is :  100
```

PREPROCESSOR DIRECTIVES

Preprocessor directives are lines included in the code of programs preceded by a hash sign (#). These lines are not program statements but directives for the preprocessor. The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements.

These preprocessor directives extend only across a single line of code. As soon as a newline character is found, the preprocessor directive ends. No semicolon is expected at the end of a preprocessor directive. The only way a preprocessor directive can extend through more than one line is by preceding the newline character at the end of the line by a backslash (\).

Simply a C Preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation.

There are different types of directives. The important directives are

1. Macro Substitution Directives
2. File inclusion directives
3. Compiler Control directives

Macro Substitution Directives

#define Preprocessor defines a constant/identifier and a value that is substituted for identifier/constant each time it is encountered in the source file. Generally macro-identifiers/constant defined using #define directive are written in the capital case to distinguish it from other variables. constants defined using #define directive are like a name-value pair.

These directives are used for

- assigning a constant to an identifier
- Assigning a symbol to an identifier
- assigning an expression to a identifier
- assigning a name to a function declaring a function

The general format of the simple macro directive is

```
#define identifier constant / symbol / function.
```

Examples:

```
#define ALPHA 1000
#define LT <=
#define INPUT scanf
#define PI 3.14
```

Example 1:

```
#include<stdio.h>
#define PI 3.14

Void main()
```

```
                {
                float radius,area;
                printf(“Enter the radius of the circle:”);
                scanf(“%f”, &radius);
                area = PI * radius * radius;
                printf(“\nArea of the Circle is %f”,area);
                }
```

Output:

```
          Enter the radius of the circle:4
          Area of the Circle is 50.24000
```

Example 2:

```
#include<stdio.h>
#include<conio.h>
#define SQUARE(x) x*x
void main()
{
int x;
clrscr();
printf("\nEnter the side of the square:");
scanf("%d",&x);
printf("\nArea of the square is %d", SQUARE(x));
getch();
}
```

Output:

```
Enter the side of the square: 4
Area of the square is 16
```

File Inclusion Directives

It is used for linking header file or another C file to the current C File to call the library and user defined functions. It has two formats. They are

Format (i): `# include <filename with extension >`
Format (ii):`# include “filename with extension ”`

The format (i) is used for linking the Header file to the current C File. A Header file is a file which has the collection of library functions.

The format (ii) is used for linking another C file to the current C File where the C file has one or more library functions.

Examples:

```
#include <string.h>
```
// to call the string functions

```
#include <math.h>
```
// To call the mathematical functions

```
#include “p2.c”
```
// To call one or more user defined functions in p2.c in the current C file.

Compiler Control Directives

These directives are the special directives used for controlling the flow of execution. These directives are used for many situations.

Example 1:

```
#include <stdio.h>
#define RAJU 100
int main()
{
   #ifndef PINKY
   {
      printf("PINKY is not defined. So, now we are going to " \
             "define here\n");
      #define PINKY 300
   }
   #else
   printf("PINKY is already defined in the program”);

   #endif
   return 0;

}
```

Output:

```
PINKY is not defined. So, now we are going to define here
```

FILE

A file is a place on the disk where a group of related data is stored. C supports a number of functions to perform basic file operations, which include

- Naming a file
- Opening a file
- Reading data from a file
- Writing data to a file and
- Closing a file

Function Name	Operation
fopen()	Creates a new file for use,Opens a new existing file for use
fclose()	Closes a file which has been opened for use
getc()	Reads a character from a file
putc()	Writes a character to a file
fprintf()	Writes a set of data values to a file
fscanf()	Reads a set of data values from a file
getw()	Reads a integer from a file
putw()	Writes an integer to the file
fseek()	Sets the position to a desired point in the file
ftell()	Gives the current position in the file
rewind()	Sets the position to the begining of the file

Defining a file

If we want to store data in a file into the secondary memory, we must specify certain things about the file to the operating system. They include

- Fielname
- data structure
- purpose

File name is a string of characters that make up a valid filename for the operating system. It may contain two parts, a primary name and an optional period with extension

Examples

Input.data
store
PROG.C
Student.c
Text.out

Data structure of a file is defined as FILE in the library of standardI/O function definitions. Therefore, all files should be declared as type FILE before they are used. FILE is a defined datatype.

When we open a file, we must specify what we want to do with the file. Example, we may write data to the file or read the already existing data.

Example:

```
FILE *fp;
```

Opening a file:

The general format

```
FILE *fp;
fp=fopen(“filename”,”mode”);
```

The first statement declares the variable fp as a pointer to the data type FILE. As stated earlier, File is a structure that is defined in the I/O Library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. This pointer, which contains all the information about the file, is subsequently used as a communication link between the system and the program.

The second statement also specifies the purpose of opening the file. The mode does this job.

r → open the file for read only.
w → open the file for writing only.
a → open the file for appending data to it.

When trying to open afile, one of the following things may happen:

1. When the mode is ‘writing’, a file with the specified name is created if the file does not exist. The contents are deleted, if the file already exists.
2. When the purpose is “appending”, the file is opened with the current contents safe. A file with the specified name is created if the file does not exists.
3. If the purpose id “reading”, and if it exists, then the file is opened with the current contents safe;otherwise an error occurs.

Consider the following statements:

```
FILE *p1, *p2;
p1=fopen(“data”,”r”);
p2=fopen(“results”,”w”);
```

the file **data** is opened for reading and **results** is opened for writing. In case the results file already exists, its contents are deleted and the files are opened as a new file. If data file does not exist error will occur

Additional modes of operation.

r + → The existing file is opened to the beginning for both reading and writing

w+ → opern for reading and writing(overwrite file).

a+ → open for reading and writing(append if file exists)

Closing a file:

The input output library supports the function to close a file; it is in the following format.

```
fclose(file_pointer);
```

A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer.

Observe the following program.

```
….
FILE *p1 *p2;
p1=fopen (“Input”,”w”);
p2=fopen (“Output”,”r”);
….
…
fclose(p1);
fclose(p2)
```

The above program opens two files and closes them after all operations on them are completed, once a file is closed its file pointer can be reversed on other file.

INPUT/OUTPUT OPERATIONS ON FILES

Following functions are used in input/output operations on files.

- `getc and putc`
- `getw and putw`
- `fprintf and fscanf`

The getc and putc functions are analogous to getchar and putchar functions and handle one character at a time. The putc function writes the character contained in character variable c to the file associated with the pointer fp1.

```
putc(c,fp1);
```

similarly getc function is used to read a character from a file that has been open in read mode.

```
c=getc(fp2).
```

The program shown below displays use of a file operations. The data enter through the keyboard and the program writes it. Character by character, to the file input. The end of the data is indicated by entering an EOF character, which is control-z. the file input is closed at this signal.

```
#include<stdio.h>
#include<conio.h>

void main()
{
FILE *f1;
char c;
clrscr();
printf("Data input output");
f1=fopen("Input.txt","w"); /*Open the file Input*/
while((c=getchar())!=EOF) /*get a character from key board*/
putc(c,f1); /*write a character to input*/
fclose(f1); /*close the file input*/
printf("\nData output\n");
f1=fopen("INPUT.txt","r"); /*Reopen the file input*/
while((c=getc(f1))!=EOF)
printf("%c",c);
fclose(f1);
getch();
}
```

Output

```
Data input output
LALITHA
THENMOZHI
^Z
Data output
LALITHA
THENMOZHI
```

The getw and putw functions:

These are integer-oriented functions. They are similar to get c and putc functions and are used to read and write integer values. These functions would be usefull when we deal with only integer data. The general forms of getw and putw are:

```
putw(integer,fp);
getw(fp);
```

```
/*Example program for using getw and putw functions*/
#include<stdio.h>
#include<conio.h>

void main()
{
FILE *f1;
int c,i;
clrscr();
f1=fopen("Input1.txt","w");
for(i=0;i<5;i++)
{
scanf("%d",&c);
putw(c,f1);
}
fclose(f1);
f1=fopen("Input1.txt","r");
while((c=getw(f1))!=EOF)
printf("%d\n",c);
fclose(f1);
getch();
}
```

Output

```
23
23
45
56
56


23
23
45
56
56
```

The fprintf & fscanf functions:

The fprintf and fscanf functions are identical to printf and scanf functions except that they work on files. The first argument of theses functions is a file pointer which specifies the file to be used. The general form of fprintf is

```
fprintf(fp,”control string”, list);
```

Where fp id a file pointer associated with a file that has been opened for writing. The control string is file output specifications list may include variable, constant and string.

```
fprintf(f1,%s%d%f”,name,age,7.5);
```

Here name is an array variable of type char and age is an int variable The general format of fscanf is

```
fscanf(fp,”controlstring”,list);
```

This statement would cause the reading of items in the control string.

Example:

```
fscanf(f2,”5s%d”,item,&quantity”);
```

Like scanf, fscanf also returns the number of items that are successfully read.

```
/*Program to handle mixed data types*/
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
int number,i;
char item[10];
clrscr();
fp=fopen("input2.txt","w");
fscanf(stdin,"%s%d",item,&number);
fprintf(fp,"%s%d",item,number);
fclose (fp);
fp=fopen("input2.txt","r");
fprintf(stdout,"%s%d",item,number);
fclose(fp);
getch();
}
```

Output

```
Pencil 10

Pencil 10
```

Random access to files:
Sometimes it is required to access only a particular part of the and not the complete file. This can be accomplished by using the following function:

fseek function:
The general format of fseek function is a s follows:

```
fseek(file pointer,offset, position);
```

This function is used to move the file position to a desired location within the file. Fileptr is a pointer to the file concerned. Offset is a number or variable of type long, and position in an integer number. Offset specifies the number of positions (bytes) to be moved from the location specified bt the position. The position can take the 3 values.

Value	Meaning
0	Beginning of the file
1	Current position
2	End of the file.

The offset may be positive or negative. Positive means move forward, negative means move backward.

Example

Statement	Meaning
fseek(fp,0L,0);	Go to the beginning (similar to rewind)
fseek(fp,0L,1);	Stay at the current position. (Rarely used)
fseek(fp,0L,2);	Go to the end of the file, past the last character of the file.
fseek(fp,m,0);	Move to (m+1)th byte in the file.
fseek(fp,m,1);	Go forward by m bytes.
fseek(fp,-m,1);	Go backward by m bytes from the current position.
fseek(fp,-m,2);	Go backward by m bytes from the end. (Position the file to the m th character from the end).

When the operation is successful, fseek returns a zero. If we attempt to move the file pointer beyond the file bouondaries, an error occurs and fseek returns -1 (minus one). It is good practice to check whether an error has occurred or not, before proceeding further.

ftell

ftell takes a file pointer and returns a number of type **long**,that corresponds to the current position. This function is useful in saving the current position of a file.

n=ftell(fp);

n would give the relative offset(in bytes)of the current position. This means that n bytes have already been read(or written).

Example of fseek and ftell

```
#include<stdio.h>
#include<conio.h>
#include<stdio.h>

void main()
{
      FILE *fp;
      long n;
      char c;
      clrscr();
      fp=fopen("g1.txt","w");
      while((c=getchar())!=EOF)
      putc(c,fp);
      printf("No. of characters entered=%ld\n",ftell(fp));
      fclose(fp);
      fp=fopen("g1.txt","r");
      n=0L;
```

```
	while(feof(fp)==0)
	{
		fseek(fp,n,0);
		printf("Position of %c is %ld\n",getc(fp),ftell(fp));
		n=n+5L;
	}
	getch();
}
```

Output:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ^Z
No. Of characters entered =26
Position of A is 0
Position of F is 5
Position of K is 5
Position of P is 15
Position of U is 20
Position of Z is 25
Position of    is 30
```

Explanation

During the first reading, the file pointer crosses the end-of-file mark when the parameter n of fseek(fp,n,0) becomes 30. Therefore, after printing the content of position 30, the loop is terminated. (There is nothing in the position 30)

For reading the file from the end, we use the statement

```
fseek(fp,-1L,2)
```

to position the file pointer to the last character. Since every read causes the position to move forward by one position, we have to move it back by two positions to read the next character. This achieved by the function.

```
fseek(fp,-2L,1);
```

in the while statement. This statement also tests whether the file pointer has crossed the file boundary or not. The loop is terminated as soon as it crosses it.

rewind

rewind takes a file pointer and resets the position to the start of the file.

```
rewind(fp)
n=ftell(fp);
```

would assign 0 to n because the file position has been set to the start of the file by rewind. Remember, the first byte in the file is numbered as 0, second as 1, and so on. This function helps us in reading a file more than once, without having to close and open the file. Remember that whenever a file is opened for reading or writing, a rewind is done implicitly.

Binary files

Binary files are very similar to arrays of structures. Binary files have two features that distinguish them from text files:

- we can instantly use any structure in the file.
- we can change the contents of a structure anywhere in the file.

After opened the binary file, we can read and write a structure or seek a specific position in the file. A file position indicator points to record0 when the file is opened.A read operation reads the structure where the file position indicator is pointing to. After reading the structure the pointer is moved to point at the next structure.A write operation will write to the currently pointed-to structure. After the write operation the file position indicator is moved to point at the next structure.The fseek function will move the file position indicator to the record that is requested. The file position indicator can not only point at the beginning of a structure, but can also point to any byte in the file.

The fread and fwrite function takes four parameters:

- A memory address
- Number of bytes to read per block
- Number of blocks to read
- A file variable

Example of 'write':

```
#include<stdio.h>
#include<conio.h>
struct rec
{
        int x;
};

void main()
{
        int i;
        FILE *fp;
        struct rec my1;
        clrscr();
        fp=fopen("test.txt","w");
        for (i=1; i <= 10;i++)
        {
                my1.x= i;
                fwrite(&my1, sizeof(struct rec), 1,fp);
        }
        fclose(fp);
        fp=fopen("test.txt","r");
        for (i=1; i <= 10;i++)
        {
                fread(&my1, sizeof(struct rec), 1,fp);
```

```
                        printf(“%d\n”,my1.x);
                }
                fclose(fp);
                getch();
        }
```
```
Output
1
2
3
4
5
6
7
8
9
10
```

In this example we declare a structure rec with the members x,y and z of the type integer. In the main function we open (fopen) a file for writing (w). Then we check if the file is open, if not, an error message is displayed and we exit the program. In the “for loop” we fill the structure member **x** with a number. Then we write the record to the file. We do this ten times, thus creating ten records. After writing the ten records, we will close the file.

Questions for Practice:

1. Add two number using pointer and Dynamic memory allocation

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int *ptr,sum=0,i;
// Allocate memory Equivalent to 1 intExampleer
ptr = (int *)malloc(sizeof(int));

for(i=0;i<2;i++)
    {
    printf("Enter number : ");
    scanf("%d",ptr);
    sum = sum + (*ptr);
    }
printf("nSum = %d",sum);
return(0);
}
```

2. Reading & Accessing array using Malloc function

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
clrscr();
int *ptr,*temp;
int i;
ptr = (int *)malloc(4*sizeof(int));  // Allocating 8 bytes
temp = ptr;  // Storing Current Pointer Value
for(i=0;i < 4;i++)
     {
     printf("Enter the Number %d : ",i);
     scanf("%d",ptr);
     ptr++;          // New Location i.e increment Pointer
     }

ptr = temp;
   for(i=0;i < 4;i++)
   {
   printf("\nNumber(%d) : %d",i,*ptr);
   ptr++;
   }
getch();
}
```

Output:

```
Enter the Number 0 : 45
Enter the Number 1 : 35
Enter the Number 2 : 25
Enter the Number 3 : 15
Number(0) : 45
Number(1) : 35
Number(2) : 25
Number(3) : 15
```

3. Explain any five file handling functions.
4. Explain opening and closing a file.
5. Write a c program to delete a file.
6. Write a c program to copy a file from one location to other location.
7. Write a c program which display source code as a output.
8. Write a c program which concatenate two file and write it third file.
9. Write a C program to read name and marks of n number of students from user and store them in a file.
10. Find the Size of File using File Handling Function.
11. Write a c program to open a file and write some text and close its.
12. Write a program to count number of characters in a file.