BUDDHA SERIES

(Unit Wise Solved Question & Answers)

Course – B.Tech (ASH)

College – Buddha Institute of Technology

(AKTU CODE-525)

Department: Applied Science and Humanities

Subject: Programming for Problem Solving (BCS-101/201)

Faculty Name: Shyam Mohan Singh

Unit – 5

Long Answer type question

1-Write a program in C to copy the content of one file into another file.
(B.TECH-(SEM II) 2019-20)

Answer:-

```
include<stdio.h>
#include<conio.h>
void main()
{
FILE *f1,*f2;
char c;
printf("Data is enter in the file named INPUT");
f1=fopen("INPUT","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("INPUT","r");
f2=fopen("OUTPUT","w");
while((c=getc(f1))!=EOF)
putc(c,f2);
fclose(f1);
fclose(f2);
printf("Data is copied into the file named OUTPUT");
f2=fopen("OUTPUT","r");
while((c=getc(f2))!=EOF)
printf("%c",c);
fclose(f2);
getch();
}
```

2- Create a file DATA and store list of 30 integers in it. Using the file DATA ,write a C program which creates two more file name EVEN and ODD to store even integers and odd integers.
(B.TECH-(SEM I) 2020-21)

Answer:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp,*f1,*f2;
```

```
int n,i;
printf(“Contents of INPUT file”);
fp=fopen("INPUT","w");
printf("enter the number");
for(i=0;i<30;i++)
{
scanf("%d",&n);
if(n==-1)
break;
else
putw(n,fp);
}
fclose(fp);
fp=fopen("INPUT","r");
f1=fopen("EVEN","w");
f2=fopen("ODD","w");
while((n=getw(fp))!=EOF)
{
if(n%2==0)
putw(n,f1);
else
putw(n,f2);
}
fclose(fp);
fclose(f1);
fclose(f2);
f1=fopen("EVEN","r");
f2=fopen("ODD","r");
printf(“Contents of EVEN file”)
while((n=getw(f1))!=EOF)
printf("%d",n);
printf(“Contents of ODD file”);
while((n=getw(f2))!=EOF)
printf("%d",n);
fclose(f1);
fclose(f2);
getch();
}
```

3) Discuss various file handling methods used in C in brief.

(B.TECH-(SEM II) 2021-22)

Answer:-

A file is a place on the disk where a group of related data is stored.

The basic operation which can be performed on file is-

- Naming a file
- opening a file
- Reading data from file
- Writing data to file
- Closing a file

Types of file- depending upon the format in which data is stored, file are categories in two ways:

Text files: As the name suggested, text files stored textual information likes alphabets, number, special symbols etc. in actuality, the ASCII code of textual characters is stored in the text files. But, since data is stored in a storage device in the binary format, the text file contents are first converted in the binary format before actually being stored in the storage device.

Binary file: As the name suggested, a binary files stored the information in binary form. Thus, the use of binary file eliminates the need of data conversion from text to binary format for storage purpose. However the data stored in a binary file is not in human understandable form. every executable file generated by C compiler is a binary file.

Defining and opening a file- if we want to store data in a file in the secondary memory, we must specify certain things about the file, to the operating system. They include:

- Filename
- Data structure
- Purpose

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

Opening a file:

The general format of the function used for opening a file is

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

The first statement declares the variable fp as a pointer to the data type FILE. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. fopen() contain the file name and mode (the purpose of opening the file).mode can be one of the following:

r is used to open the file for reading only.
w is used to open the file for writing only.
a is used to open the file for appending data to it.

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

- When the mode is writing, a file with specified name is created if the file does not exist. The contents are deleted, if the file already exists.
- When the purpose is appending the file is opened with the current contents safe. A file with specified name is created if the file does not exist.
- If the purpose is reading and if it exists, then the file is opened with the current contents safe otherwise an error occurs.

Closing a File

A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. it also prevents any accidental misuse of the file. This would close the file associated with the file pointer. The input output library supports the function to close a file. it takes the following form:

fclose (filepointer) ;

4) Write the importance of free () function in Dynamic memory allocation. Explain dynamic memory allocation with the functions use in it.**?**

(B.TECH-(SEM I) 2021-22)

Answer:-

The `free()` function is critical in dynamic memory allocation because it ensures that dynamically allocated memory is properly deallocated when it is no longer needed, thus preventing memory leaks in a program. Without `free()`, memory that has been allocated dynamically using functions like `malloc()`, `calloc()`, or `realloc()` would not be released, causing the program to consume more memory over time and potentially leading to system instability or crashes due to memory exhaustion.

Dynamic Memory Allocation in C

Dynamic memory allocation is the process of allocating memory at runtime, as opposed to static memory allocation, where memory is allocated at compile-time. This allows programs to allocate memory as needed, which is particularly useful when the size of the data structures cannot be determined in advance. The key functions used in dynamic memory allocation are `malloc()`, `calloc()`, `realloc()`, and `free()`.

Functions Used in Dynamic Memory Allocation

1. **`malloc()`** (Memory Allocation):
    - Syntax: `void* malloc(size_t size);`
    - The `malloc()` function is used to allocate a block of memory of a specified size.
    - It returns a pointer to the first byte of the allocated memory. If the memory allocation fails, it returns `NULL`.

- The contents of the allocated memory are not initialized, meaning they contain garbage values.

2- `calloc()` (Contiguous Allocation):

- Syntax: `void* calloc(size_t num, size_t size);`
- The `calloc()` function allocates memory for an array of `num` elements, each of size `size` bytes.
- Unlike `malloc()`, `calloc()` initializes the allocated memory to zero.
- It also returns a pointer to the allocated memory or `NULL` if allocation fails.

5) Define dynamic memory allocation. Differentiate between malloc () and calloc () with proper example**?**

(B.TECH-(SEM II) 2022-23)

Answer:-

Dynamic memory allocation refers to the process of allocating memory during the execution of a program, rather than at compile-time. This allocation is managed by the system's memory manager, and it allows programs to request memory as needed at runtime. The primary advantage of dynamic memory allocation is its flexibility, allowing programs to use memory efficiently depending on the program's requirements at any given moment.

In C, dynamic memory allocation is done using functions like `malloc()`, `calloc()`, `realloc()`, and `free()`. These functions allocate memory from the heap, which is a pool of memory used for dynamic allocation. When memory is no longer needed, it is freed using `free()`.

`malloc()` and `calloc()` are both used to allocate memory dynamically, but they have important differences:

Feature	`malloc()`	`calloc()`
Function Signature	`void* malloc(size_t size);`	`void* calloc(size_t num, size_t size);`
Memory Initialization	Does **not** initialize the memory; the memory block contains garbage values.	Initializes the allocated memory to **zero**.
Number of Arguments	Takes **one argument**: the total size of memory to allocate.	Takes **two arguments**: the number of elements and the size of each element.

Feature	`malloc()`	`calloc()`
Use Case	Suitable when memory initialization is not necessary.	Suitable when you need initialized memory, typically zeroed-out.
Return Value	Returns a pointer to the allocated memory block. If allocation fails, returns `NULL`.	Returns a pointer to the allocated memory block. If allocation fails, returns `NULL`.

6) Define the concept of pointer? Also define the dynamic memory allocation and various functions for dynamic memory allocation, with suitable examples.**?**

(B.TECH-(SEM I) 2023-24)

Answer:-

A **pointer** in C is a variable that stores the **memory address** of another variable. Instead of holding data values directly, pointers hold the address of variables in memory, which allows for indirect manipulation of data. Pointers are powerful in C because they provide the ability to dynamically allocate memory, pass large data structures efficiently, and manipulate memory directly.

A pointer is declared using the `*` operator, and it is assigned an address using the `&` operator.

- `data_type` is the type of data the pointer will point to.
- `*` is used to declare that the variable is a pointer.

Dynamic Memory Allocation

Dynamic memory allocation in C refers to the process of allocating memory at **runtime**, as opposed to static memory allocation, which occurs during compile time. This allows for more flexible memory management, as the size of the memory block can be determined during program execution.

Dynamic memory is allocated from the **heap** (a special area of memory used for dynamic allocation), and the memory is freed when it is no longer needed using the `free()` function.

Functions for Dynamic Memory Allocation in C

The C standard library provides several functions for dynamic memory allocation, which are declared in the `stdlib.h` header file:

1. **`malloc()`** (Memory Allocation):
   - **Syntax**: `void* malloc(size_t size);`
   - Allocates a block of memory of the specified `size` in bytes.
   - The memory is **not initialized**, meaning it contains garbage values.
   - Returns a pointer to the allocated memory block or `NULL` if the allocation fails.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    // Step 1: Dynamically allocate memory for 5 integers using malloc()
    int *arr = (int*) malloc(5 * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    // Step 2: Initialize the array with values
    for (int i = 0; i < 5; i++) {
        arr[i] = i + 1;  // Values will be 1, 2, 3, 4, 5
    }

    // Step 3: Print the values
    printf("Array values after malloc: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Step 4: Dynamically resize the array using realloc()
    arr = (int*) realloc(arr, 10 * sizeof(int));
    if (arr == NULL) {
        printf("Memory reallocation failed!\n");
        return 1;
    }

    // Step 5: Initialize the new elements in the resized array
    for (int i = 5; i < 10; i++) {
        arr[i] = i + 1;  // Values will be 6, 7, 8, 9, 10
    }

    // Step 6: Print the updated array
    printf("Array values after realloc: ");
    for (int i = 0; i < 10; i++) {
        printf("%d ", arr[i]);
    }
```

```
    printf("\n");

    // Step 7: Deallocate the memory using free()
    free(arr);

    return 0;
}
```

7) **Define the structure of a node in linked list.**

(B.TECH-(SEM II) 2020-21)

Answer:-

In a linked list, each element is stored in a **node**. A node typically contains two parts:

1. **Data**: The actual value or data that the node holds.
2. **Pointer (Next)**: A reference (or address) to the next node in the linked list.

In C, the structure of a node in a linked list can be represented using a `struct`. Each node stores the data and a pointer to the next node, allowing the nodes to be connected in a sequence.

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

// Definition of a Node in a Singly Linked List
struct Node {
  int data;
  struct Node* next;
};

int main() {
  // Create a new node
  struct Node* node1 = (struct Node*) malloc(sizeof(struct Node));

  // Initialize the data field of the node
  node1->data = 10;

  // Initialize the next pointer to NULL (since this will be the last node for now)
  node1->next = NULL;

  // Print the data of the node
  printf("Node data: %d\n", node1->data);
```

```
    printf("Next node address: %p\n", node1->next);

    // Free allocated memory
    free(node1);

    return 0;
}
```