**DAY-11 (8-11)**

* **strings:** collection of characters ends with NULL character (\0)
* while declaring the string one extra length should be declared for the NULL character

eg: types of declaration with initialization

char Name[6] = {‘h’,’e’,’l’,’l’,’o’,’\0’};

char Name[6] = “hello”;

char greet[ ] = “how are you”;

char[21]; ---- without initialization

* collection of characters is different from string
* 2d array of characters:

char Name[ ][20]={“Pravallika”,”pravi”,”sri”} // the compiler can understand rows as 3

char Name[5][20] // there are 5 names each should have 20 characters

* **String functions:** header file #include<string.h>
* strcpy, strncpy – to copy string from source to destination
* ptr=strcpy (d1, s1);
* ptr holds the address of d1 as the function returns the base address of destination string
* strcat, strncat – to concat two strings
* Size of destination should be greater than source
* strcat(d1, s1);
* the s1 string is concatenated with d1and even after change the s1 remains same where the function returns the d1 address
* strcmp – comparing two strings
* It will check character by character by checking ASCII value
* If str1 is greater than str2 it returns positive value. If str1 is lesser than str2 it returns negative value. If str1 is equal to str2 it returns zero
* strtok – delimiter is used
* ptr=strtok(s1," ");
* returns address of string first occurred before the delimiter
* strlen – gives length of string excluding the NULL character
* strerror – captures string handling errors
* strstr – to find the first occurrence of the substring in the main string
* p = strstr(s1, s2);
* s2 substring occurrence in s1 string
* returns address of first occurrence of substring
* strchr, strrchr – gives the address of the character occurrence

strstr, strcat, strtok (p4)

why ; is used in structures

* UDT: user defined datatype – to define multiple of its types
* Bunching data types
* Structures:
* Syntax: struct tagname

{

members of structure;

};

* Eg: struct square

{

int length;

int breadth;

}s3,s4; ---- s3,s4 are structure(square) variables

struct square s1,s2; ---- s1,s2 are also structure(square) variables

we can declare variables in either way

* We can declare array of structures
* To declare structure in function void func(struct square s1)
* If we use typedef to the structure name like eg: typedef struct square SQR we can replace it from writing whole thing
* struct square s1,s2,s3 -> SQR s1,s2,s3 (when defined typedef)
* typedef struct square

{

int length;

int breadth;

}SQR1;

* . is used when variables of structure are static

-> is used when variables of structure are pointers

* In C we can’t define a function in structure
* To avoid structure padding it is better to define all the numeric constants at one place (either in the beginning or end) and string constants at one place (either in the beginning or end)

Error: free (): invalid pointer

Aborted code dumped

* As we are moving the pointer to print the desired set and clearing the pointed pointer but not clearing the base address

**DAY-13 (12-11)**

* **Union** - The members of the union can be accessed one at a time.
* It is almost same as structure accept this one property
* Union has the size of the data type which has greater size
* Eg : union item{

int m;

float x;

char c;

};

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

2000 2001 2002 2003

char c

int m

float x

the size of the union in this example is 4 bytes

* **Enum (enumeration) –**

**File handling**

* File is collection of logically related information
* Two types:

1. Sequential file
2. Random access file

* Use special structure named FILE
* FILE \*fp; – file pointer
* FILE \*fopen(char \*name,char \*mode) – to open a file

name of the file, mode you want use in the file - are arguments given to the fopen (mode can be read,write,append)

* Once file is opened it returns the file address to file pointer
* /\*

FILE => DS

fp => file pointer to FILE DS

fopen("name of the file","mode") => opening the file

mode => r,w,a,r+,w+,a+

fclose(fp) => close the opened file

read/write

fprintf,fscanf => formated writing and reading resp

fputs,fgets => unformated i/o operation

fwrite/fread => reading/writing binary objects

\*/

* By using system, we can’t use linux commands
* fseek
* fread();
* fread (
* fread returns the size of how may no of bytes it is read
* fwrite ();/
* fread (
* fwrite returns the size of how may no of bytes it is written

**DAY-14 (14-11)**

* **Multi-threading:**
* Lwt/lwp – light weight thread or light weight process
* Thread –
* Process –
* Concurrency – to perform multiple task(functions) simultaneously
* Concurrency in programming:
* Multithreading: multiple threads within a single process are created to perform different parts of task or handle multiple tasks concurrently.
* threads share the same memory space – leads to race condition (to avoid race condition we need to use lock).
* Multiprocessing: separate processes are created to handle different tasks concurrently
* Each process has own memory space
* Requires inter process communication (IPC) methods like pipes or shared memory
* Asynchronous programming: non-blocking functions are used to handle tasks such as I/O operations without blocking the main thread,
* Concurrency is about dealing with multiple tasks at once or having multiple threads

Parallelism is executing multiple tasks truly simultaneously, achieved by having multiple CPU cores.

**DAY-15 (15-11)**

* Command line arguments
* Command line arguments will be taken into main function
* main(int argc, char \*argv[) 🡪 what are the arguments to be passed

no of arguments to be passed

* Arguments are separated with space, even path can be give as argument in command line

**DAY-16 (18-11)**

* **Data structure:** two types of DS are primitive and non-primitive
* Linear DS – array, stacks, queue,

Non-linear DS - trees

* Single linked list
* Steps to follow:
* Creation
* Initialization
* Joining
* Traversing
* Operations on linked list: deletion, adding, appending, searching
* Count no of nodes present

**DAY-17 (19-11)**

* if we delete a node at beginning of a circular dll then what will be the base node
* to free the allocated memory in structures we use while loop and take two pointers same like node deletion
* **Stack: LIFO**
* When we can see bottom of the stack then stack is said to be empty (bottom = top)
* When limit = top stack is full
* Size/limit/max is to be fixed, bottom is also to be fixed
* Stack moves from bottom to top
* We cant pull or pop elements from the middle and these operations are to be done only from the top
* - Insert/push: to check stack full or not – top is increasing

- delete/remove/pop/pull: to check the stack empty or not – top is decreasing

* To pop an element decrement the top and then pop the element
* **Queue: FIFO**
* If front = rare/max then reset the front and rare (i.e front = 0,rare = 0)
* It is done in the pop operation only

**DAY-18 (21-11)**

* Circular queue, priority queue for objective
* **Searching algorithms:** linear, binary
* Linear / sequential: searching each element one after the other sequentially
* Time consuming. Time complexity O[n]
* Can be searched in any unsorted array/list
* Interval / binary: search in either of the half in sorted list
* Time consuming is less
* Divide and conquer
* Search algorithm can be implemented in iterative or recursive methods
* **Iterative method**

Do until low is equal to high

mid = (low +high)/2

if ( x== arr[mid])

return mid

elif( x > arr[mid])

low = mid + 1

else

high = mid -1

* **Recursive method**

mid = (low +high)/2

if ( x== arr[mid])

return mid

else if( x > arr[mid])

return function(arr, key, mid + 1, high)

else

return function(arr, key, low, mid-1)

* Practical algorithm design issues:
* Time complexity
* Space complexity
* **Sorting:** two types – space (2), stability (2)
* In place: sorting algorithm which does not require any extra space for sorting.

eg: bubble sort

* Referred as sinking sort.
* In bubble sort while in increasing order, in the first iteration the last element is in the correct position
* We need two loops

for(i=0;i<n-1;i++)

for(j=0;j<n-i-1;j++)

* out place: sorting algorithm which require extra space for sorting.

eg: merge sort

* divide and conquer algorithm
* divide the array into two half and recursively continue dividing so that it can’t be divided further
* stable sort:

eg: insertion sort

* easy to implement
* continuous inflow
* unstable sort:

eg: quick sort

* divide and conquer
* we can choose a pivot element(it can be first, last, any element)
* extra space is not required like merge sort
* in ascending order left to pivot are smaller in descending order left to pivot are greater
* selection sort: we repeatedly find the min element and move it to the sorted part of array to make unsorted array sort
* In selection sort while in increasing order, in the first iteration the first (min) element is in the correct position

**DAY-19 (22-11)**

* Algorithm design approach:
* Top-down approach already present solution to gives instructions
* Bottom-up approach
* There are the different types of time complexity:
* Best case
* worst case
* average case
* Big O notation: it is a worst case scenario and describe the execution of time or space
* O(1)🡺 it will always execute in the same time or space regardless of the size of the input data set

Eg: push  and pop operation for a stack

* O(N) 🡺 it describes the algorithm whose performance will grow linearly and in direct proportion to the size of the input data set.

Big O notation will always assume the upper limit

Eg: linear search with unsorted data

* O(N^2) 🡺 it performance is directly proportion to the square of the size of the input data set

This is common with algorithms that are contains nested loops

Eg: 2 loops

* O(logN) 🡺 iterative having the data set described in  binary search example produces a growth curve that peaks at the beginning of and slowly flattens out as the size of the data set

Eg; binary search

* O(NlogN)🡺 logarithmic time

Eg: more advanced sorting algorithm: quick sort, merge sort

* Upper bound means worst case
* Merge sort, quick sort