Data Structure Notes DEMO

Handwritten Notes

For more PDFs and notes visit my telegram page.. Named "BeingPro33"

@beingpro33

* What is data structure? Explain its type and components?

A data structure is the organization of data in a computer's memory or within a file on disk using a specific model. When we working with the huge amount of data of an organization, we must know how to manage or organize data smoothly and electronically.

Data may be arranged in many different ways such as the logical or mathematical model for a particular organization of data is turned as a data structure.

"The organized collection of data is called a Data Structure."

"A data structure is representation of data and the operation allowed on that data."

[Data structure = Organized data + allowed operations]

* Classification of data structure -

i) According to type of data -

a) Homogenous data structure -

In homogenous data structure, the data elements are of the same type like an array data structure.

b) Non-homogenous data structure -

Non-homogenous data structure may not be of same type like struct or abstract data type.

2) A/c to storage allocation time -

a) Static data structure -
Static structures are ones whose size and structure associate memory location are fixed at compile time.
Eg:- Array

b) Dynamic data structure -
Dynamic data structures are ones that expand or shrink as required during the execution of program and their associated memory location change. i,e pointers.

3) On the Basis of storage sequence -

a) Linear data structure -
A data structure is said to be linear if its element form a sequence processing of data item is possible in a linear fashion or one by one sequentially like in an array.

There are two basic ways of representing such a linear structure in memory -

i) One way is to have the linear relationship b/w the elements by means of sequential memory location, such linear structure are called array.

ii) The other way is to have the linear

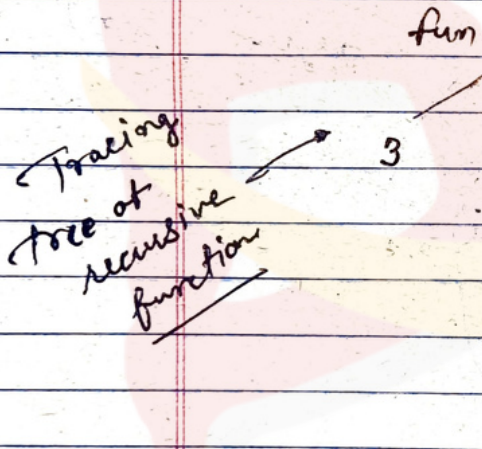
Recursion

infinite

Date
Page

①
```
Void fun1(int n)
{
  if (n > 0)
  {
    printf("%d", n);
    fun1(n-1);
  }
}
Void main()
{
  int x = 3;
  fun1(x);
}
```

(It is called base condition and it must be written in program for terminating the function, otherwise program runs infinites.)



Tracing tree of recursive function

Here printing is done at calling time.

output - 3, 2, 1

(Terminated and now go back to the previous call)

→ Once the call has been terminated, the control should go back to the previous call.

22/03/22

Types of Recursion

Date
Page

i) Tail recursion -

When all the operation will be performed at calling time only and the function will not be performing any operation at returning time. Means everything is performed at calling time.

Eg:-
```
void fun(int n)
{
  if(n>0)
  { printf("%d",n);
    fun(n-1);
  }
}
main()
{
  fun(3);
}
```

(or)
```
fun(n)
{
  if(n>0);
  {
  }
  fun(n-1) # n;
}
```

if will performed at returning time so its not a tail recursion.

Comparision of tail recursion with loop-

```
void fun(int n)
{
  while(n>0)
  printf("%d",n);
    n--;
}
main()
{
  fun(3)
}
```

Time - O(n)

space - O(1)
(if its activation record is only 1)

```
void fun(int n)
{
  if(n>0)
  { printf("%d",n);
    fun(n-1);
  }
}
main()
{
  fun(3);
}
```

time - O(n)

space - O(n)
(if its activation record is n+1).

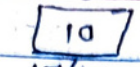
∴ In the case of tail recursion, loop is more efficient.

29/03/2022

Array

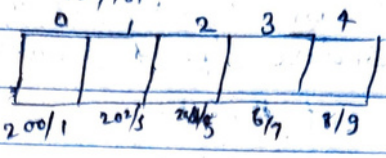
Date
Page

int x = 10;



→ Scalar

int A[5] = A

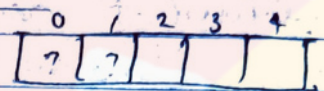


→ Vector

* Arrays are vector variable.

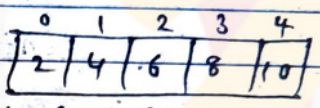
* Declaration and initialization of an array -

int A[5];

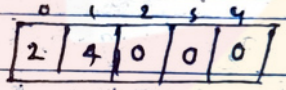


garbage value

int A[5] = {2, 4, 6, 8, 10};

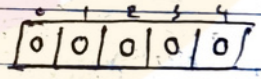


int A[5] = {2, 4};

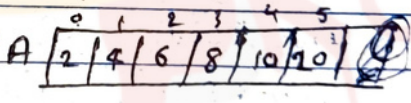


(Remaining are initialize with 0)

int A[5] = {0};

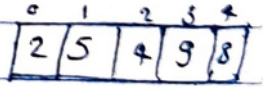


int A[] = {2, 4, 6, 8, 10, 20}



* Accessing of an array -

int A[5] = {2, 5, 4, 9, 8};



① for (i=0; i<5; i++)
{ printf ("%d", A[i]);
}

② printf ("%d", A[2]);

③ " " ", 2[A]);

④ " " ", *(A+2));

13/04/21

Strings

Date / Page

ASCII codes (for only English language)

A - 65	a = 97	0 = 48
B - 66	b = 98	1 = 49
!	!	!
Z = 90	z = 122	9 = 57

↵ = 10 , space = 13 , esc = 27

* ASCII code stores 1 byte in computer memory.
* ASCII codes Ranges from 0 to 127. (Total - 128)

UNICODES (Subset of ASCII)

↓

It works for all language

→ takes memory - 2 bytes (16 bits)

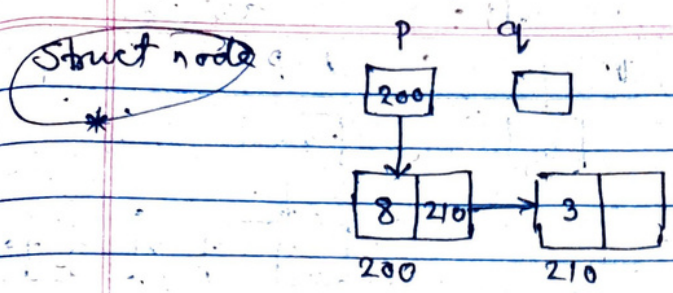
→ These are represented in Hexadecimal form & in four digits.

```
# Char temp;
✓ temp = 'A';
✗ temp = 'AB';
✗ temp = A;
✗ temp = "A";
printf("%d", temp); → 65
printf("%c", temp); → A
```

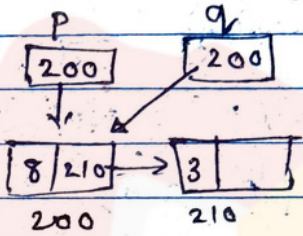
temp

A
65

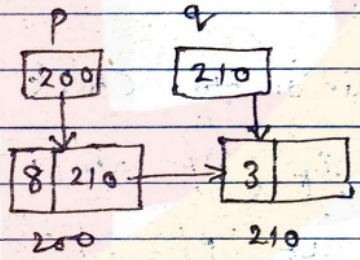
struct node *P, *q



i) q = P; →



ii) q = P → next; →



iii) P = P → next; →