C Language

Eng: Mahmoud Ramadan Elshenawy

Topic: C language notes

I wish I was able to help in some way

DATE / / OBJECT C. language.

1) Data Types.

Primary	derived	user defined
char → 1 byte	array []	struct
int → 4 "	Pointer *	union
short → 2 "	functions	enum
long → 4 "		typedef
double → 8 "		
float → 4 "		

* Range of data type

- Signed "+,-" $\longrightarrow -2^{n-1} \longrightarrow 2^{(n-1)} - 1$
- unsigned "+" $\longrightarrow 0 \longrightarrow 2^{n} - 1$

n → refer to number of bits.

* Type casting → convert datatype to another without losing its original meaning.

- implicit → done by compiler.
- Explicit → by user
  - Syntax → ☐ = (datatype) ☐ .

DATE / / OBJECT

* output function

Global	char	string
Printf("%□", □);	Putchar( );	Puts( );

2* operators

Relational operator	Logic operator	assignment operator
<, >, <=, >=, ==, !=	&&, ||, !	+=, -=, *=, /=, %=
bitwise operator	**Conditional operator**	**unary operator**
&, |, ^, ~ << → left shift >> → Right shift	? : □ = (condition) ? □1 : □2 True → □ = □1 False → □ = □2	++, --, ~ "on's complement" Posi fix ↓ Pre fix □++; ++□;

* input function

Global	Char	string

Scanf("%□", &□);	getchar( );
	getche( );
	getch( );
		get( );

D.dre

3# flow control

①	②	③	④

if(condition)
{
≠
}
else
{
≠
}
	for(initiliz; condition, update)
	{
	≠
	}
		while(condition)
		{
		≠
		update exp;
		}
			Do
			{
			#
			* update exp
			} while (exp);

5)

```
switch (variable )
{
   case 1:  OR case '+';
   {
      ≠
      break
   }
}; break;
}
```

6)

goto Cable

Ex → char RE;

RE: ⇝ lable

Printf(" — ");

goto RE;

⇝ under conditional flow of control

Continue , break , return

DATE / / SUBJECT

Functions

advantage → 1- Easy to debug 2- Code Reusability.
3- Reduce code size 4- Calling more than one time

Types → 1- standard → build in Processor "Library"
2- user defined

Fun → Prototype → ReturnType funname (arguments);
→ Calling → funname (variable)
→ definition → ReturnType funname (arguments
{
statements & declarations.
}

Calling

1) Call by value
- Pass value of Parameter during Call
- Copy to function's actual local Argu

2) Call by reference
- Pass Parameter location
- Copy and assign them to local Argu.

---

* Recursive fun → Calling function inside fun
disadvantage: 1- waste memory

2- Large execution Time.

advantage → Solving math Problems.

✱ Types of user defined fun

1- static fun "helper fun"

2- inline fun → Less execution Time

3- API fun.

C Language

more

5# C PreProcessor

1- file inclusion 2- Macro 3- fun Like Macro

1) file inclusion
- → "file1" → in case of double quote
  - → Searching for the file TYPically where source Program found and locate it
- → <file2> → in case of enclosed
  - → Searching for system Path to locate this file

2) Macro :- Text Replacment

Ex: #define AA 10
- → variable name should be Capitalize.

2- Macro to Replace Tiny fun
- adv → Reduce code size

1- #ifdef = #if defined
2- #warning 3- #error
4- #debug
5- #ifndef = #if !defined
6- #if #else #endif → no need for { } omit.
7- #undef → لوعايز يظهر الغلط لما يبدأ build
8- # Pragma → depend on compiler.
- used for Packing.

DATE / / OBJECT

3- fun like Macro

adv : 1- less execution time

disadv : Large space

Ex: #define ADD(v1,v2) (v1+v2)

*for multiline Macro use "\"

Ex → #define Mul(x,y) \

int i=0; \

int x=0; \

i=x;

** Storage classes

	storage	initial value	scope	life
1-auto	stack	carabage	withinblock	end of block
2-extren	data segment	zero	global files	end of program
3-static	data segment	zero	within block	end of program
4-Register	cpu Register	carbage	within block	End of block

C Language

D,qre

* header files
  - files initialized and saved to use it more than once

Note → Make "header guard" to avoid double inclusion

Ex:
```
#ifndef Header_H
#define Header_H
#endif
```

* Modular Programming
  - Separate Program codes into interchangable units and ease of Modification
  - improve maintainability.

DATE / / OBJECT

6 * Pointers

Size → word size.

* Syntax → dataType *Pointer name ;

Ex: int a=5, *ptr ;

ptr = &a ; *ptr = 10 ;

assign → refer to address of a get reference

↳ here it will change value of a to be 10

* Pointer Cast :

dataType *ptr = (dataType*) &variable

Ex: Char x ;

int *ptr = (int*) &x ;

it's important to cast pointer if you will make any operation like step.

* Pointer Types

1- void Pointer "void *

→ Point to some data in storage which doesn't have specific type

in case we need to use void Pointer

we need to cast "specify" to data type

Ex: void *ptr ; ⇝ void Pointer

(int*) ptr ; → change to integer Pointer

C Language

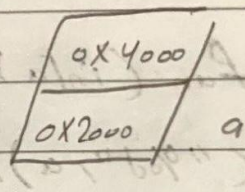
more

2) Null Pointer ⇝ Pointer which Point to nothing

3) Dangling Pointer → Point to memory location that has been deleted "Run Time error"

4) Wild Pointer → Pointer has not been initialized.

Ex: int a = 5;
int *ptr;
ptr = &a; *ptr = 10



printf("%P, %P %d %d %P", &a, ptr, a, *ptr, &ptr);
0X2000 0X2000 10 10 0X4000

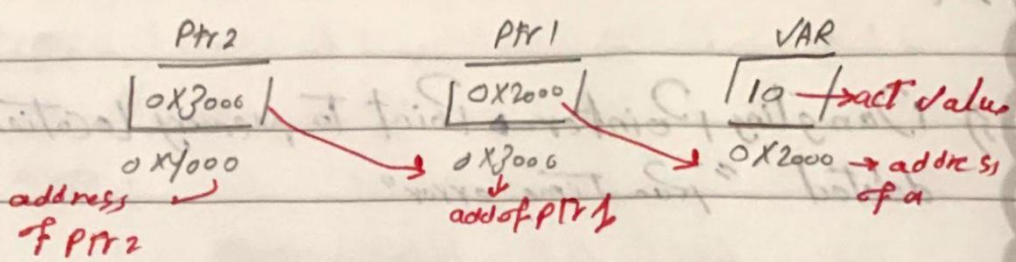
* difference between
1) const char *ptr
→ Pointer to Constant char → Cannot change the value by ptr
↳ but we can change Pointer "const char"
2) Char *const ptr → Constant Pointer to char
- Can not change Pointer
- Can change value

3) Const char * const ptr ⇝ constant Pointer to constant char
→ Can not change Pointer or its value

DATE / / OBJECT

double pointer

→ int **ptr;



* Pointer to fun

Ex1: void (*ptrtofun) (int);

Ex:2

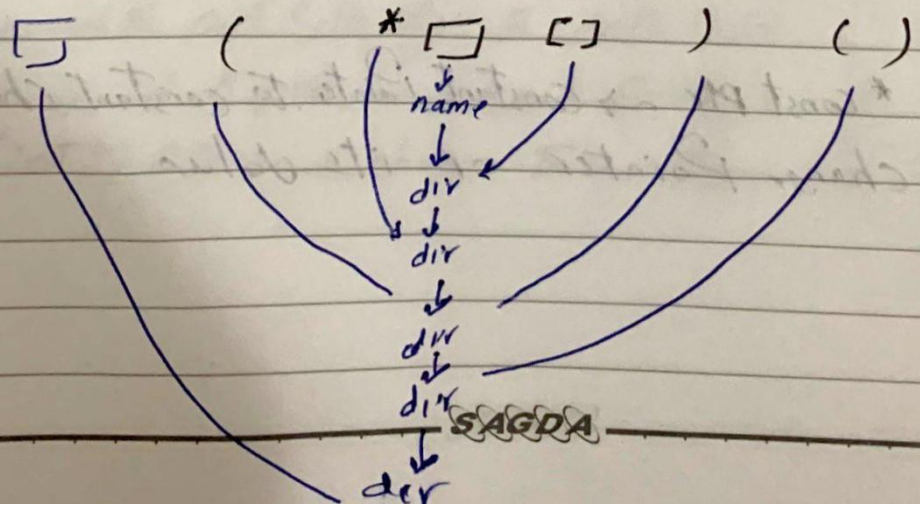
```
void fun (int a)
{
printf("%d", a);
}
int main()
{
void (*ptr) (int) = &fun;
(*ptr) (10);
}
```

* Complex decleration

from left to right for reading

1- int *fun () → fun that return pointer of integer

steps for Read



/ / owecrf

D,qre

7 ** array

syntax → datatype arr-name [ ] ; [No. of element.]

int arr[5] = { □ , □ , □ } ;

*Notes:

1- array name → has no memory address

2- No assign to array name → Er (arr++) XXX

3- size of array = (datatype * No. of elemen)

4- int arr[5] = {1,2} →

* startupcode ويقعد ألتريع carbuge صفر أو باقي يعني لا مالكش

5- int arr[5] = 1; → error

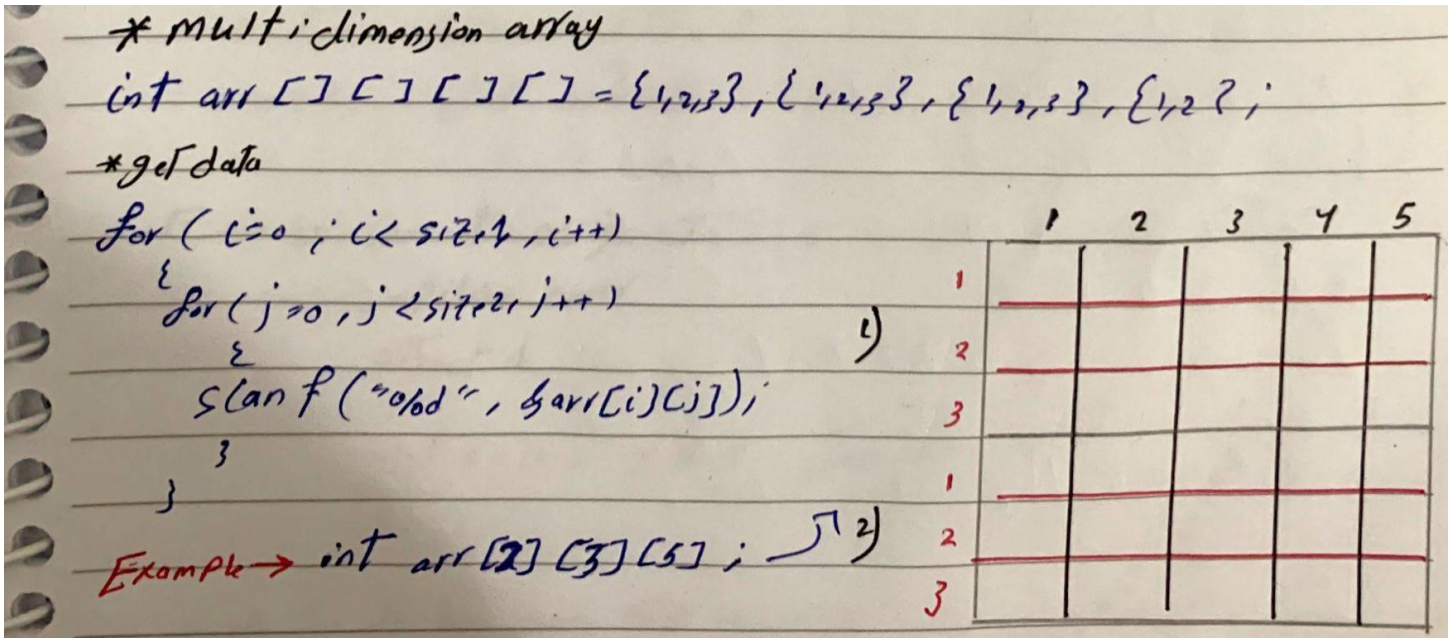
6- int arr[ ] = {1,2,3}; → dependon compiler

هيحط باقي رقم عدد خاناته

*get data

```
for (int i=0 ; i< sizeofarray ; i++)
{ scanf (" %d ", & arr[i]);
}
```

owecr


* multi dimension array
int arr [ ] [ ] [ ] [ ] = {1,2,3}, {1,2,3}, {1,2,3}, {1,2};
*get data
for (i=0 ; i< size4 , i++)
{
for (j=0 , j<size2, j++)
{
scanf ("%d", &arr[i][j]);
}
}
Example → int arr[2] [3] [5] ;
1)
2)
1 2 3 4 5
1
2
3
1
2
3

* array of Pointers.

For first element address
int arr[5];
int *ptr = arr;

For whole array address.
int (*ptr)[5];

* Complex Pointer

Priority → 1) (), [] 2) *, VAR name 3) data Type

Ex

1) char (*ptr)(short) → Pointer To fun that take short and return data type of char

2) char *(*ptr[3])(int *const, const *int)

→ Pointer To array of three element to function That take integer Pointer To constant and constant Pointer to integer and return character Pointer

nre / /

DATE / / OBJECT

8) ** structure

→ user defined data type

→ Syntax

```
struct [structname]
{
    data
};
```

```
struct [structname]
{
    data
} variable ;
```

* to use struct we should create element from struct to use it.

Ex:
```
struct data
{
    int x, y;
};
int main()
{
    struct data P1;
```

To access

```
P1.x = 5;
```

OR `struct data P1 = {.x=5, .y=10};`

```
}
```

→ To Access struct we use dot operator (.)

note → We can not assign to any variable inside struct

→ compiler error

```
struct A
{
int x=10; XXX compiler error
};
```

→ We can declare more than variable over one struct

DATE / / OBJECT

- We can create struct inside struct
  Call nested struct

Ex
```
struct data
  {
  char name[20];
  int ID;
  } i;
int main()
{
struct data P[10];
 int i;
for(i=0;i<10;i++)
      {
      printf("no of is%d :",i)
   scanf("%d", &P[i].ID);
   printf("name");
   scanf("%s", &P[i].name);
  }
```

---

* structure with pointer
  syntax → struct name *ptrtostruct
  struct name P1; ~> ptrtostruct = &P1
  → for access struct
  1- → ~> ptrtostruct → var = value;

  2- (*ptrtostruct).VAR = value;

Dare

owecr

DATE / / OBJECT

* Struct with function

Roles → 1- create struct first

2- creat fun and create element from struct data

note → مينفعش fun تبقى فوق struct هيدي error

لازم create struct في الأول

Ex:

```
Struct data ADD_number()
{
 int x,y;

} ;
void display (struct data object)
{
 Printf ("%d", object.x);
   "   ("%d", object.y);
}
main()
{
struct data P1;
  "     "  P2;
P1= ADD_Number();    P2 = ADD_Number();
display (P1);
  "    (P2);
}
```

DATE / / OBJECT

* Padding & Packing

* default → Padding

Padding

adv - less execution Time
2 - Performance is high

disadv
- waste memory

Packing → collect data side by side

adv → less size
but large in Time execution

→ Packing → by directives

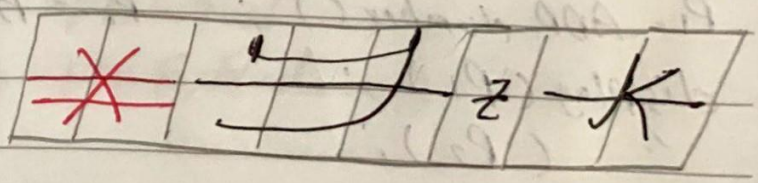
1) * Pragma Pack (1)
(2)
(3)

2) --attribute-- ((Packed))

→ Padding → for reduce cycle make large data type at first

* bitfield → Select no of bits
(:)

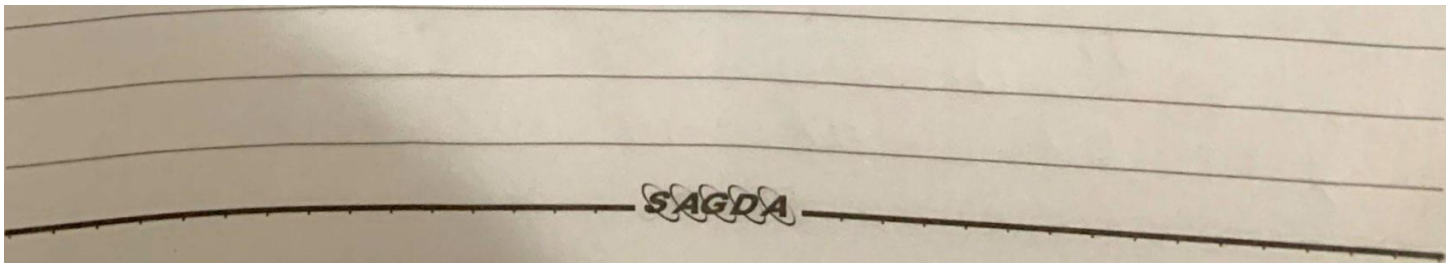
Ex: struct data
{
char x : 2;
" y : 3
" z : 1
" k : 2
};



size → (1) byte

Dare

owecr



nnre / / owecr

9) ** Union

→ reduce code size & make operation before overwrite

→ Reserve biggest data Type

(Ex) union Personal

{ int x;

double y; Size of union = 8 byte

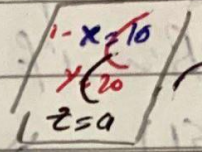
char z;

};

1- x = 10

2- y = 20

3- z = a



↝ overwrite

10) ** Enum

- Size → word size
- important of using flags
- assign first element "zero" and provide in order

Ex:

Enum week

{

Sat, → 0

Sun, → 1

mon, → 2

Tue, → 3

Wen, → 4

Thur, → 5

Friday → 6

}

out

if we assign it will provide in order after it

Ex if we make monday = 22

Then

Tues → 26

Wend → 27

Thru → 28

Friday → 29

11) ** Typedef → for create data Type names

Dare

OBJECT

17) # String → library "string.h"
→ Collection of letters
Functions
1) Strcpy (S1, S2) ⟸
Make copy of S_2 to S_1
2) strcmp (S1, S2) ⟸
return → zero if equal
" → negative if $S1 < S_2$
" → Positive if $S_1 > S_2$
3) strlen (S1) ⟹ return length of S_1

4) strcat (S_1, S_2) → Concatenate S_2 to end of S_1

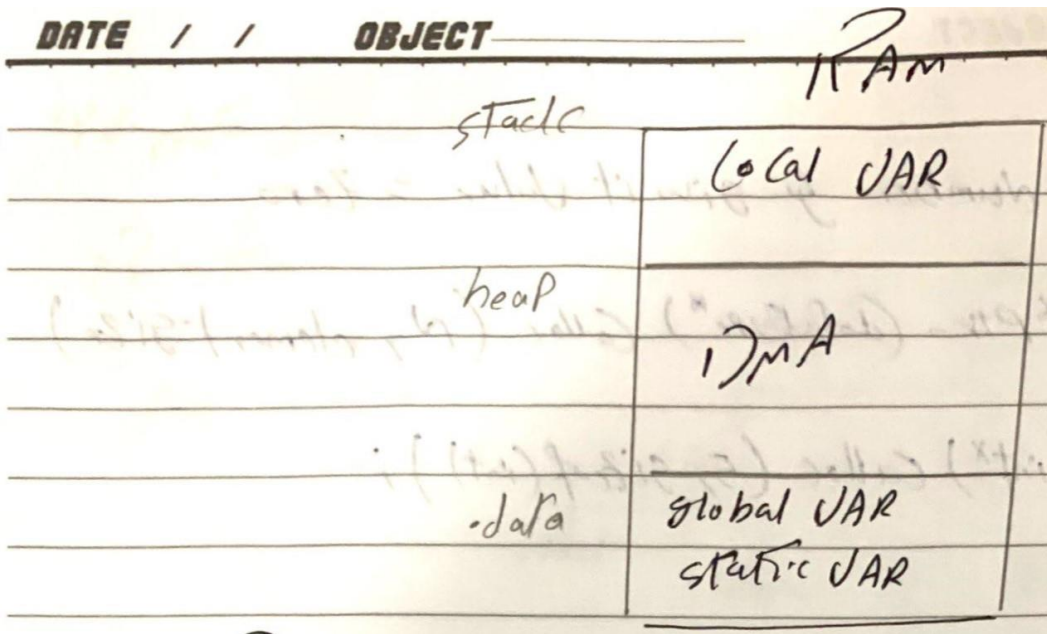
5) strtolc → tolconi zations

6) gets (s) for input
7) Puts (S_1) " out Put

SAGDA

owecr

DATE / / OBJECT



* 13) DMA → dynamic memory allocation
Store in → heap "Run Time"
Library → <stdlib.h>
has specific four Funct
1- malloc() 2- calloc() 3- realloc() 4- free()

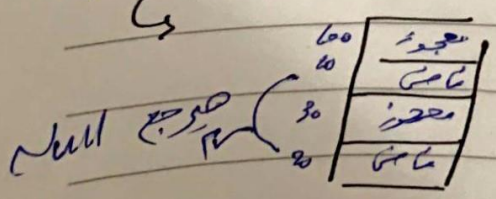
1) Malloc()
reserve location in memory and Return Pointer Point to first element
Syntax
datatype *ptr = (datatype*) malloc (byte size);
Ex
int *ptr = (int*) malloc (5 * sizeof (char));
For check memory reserved
if (ptr != Null)
{ True
}else
{ false
}

size = 5 byte
has garbage data

2) Calloc()
allocate specific number & give it value = zero
syntax
datatype *ptr = (dataType*) Calloc (N, element size)
Ex
int *ptr = (int*) Calloc (5, sizeof(int));

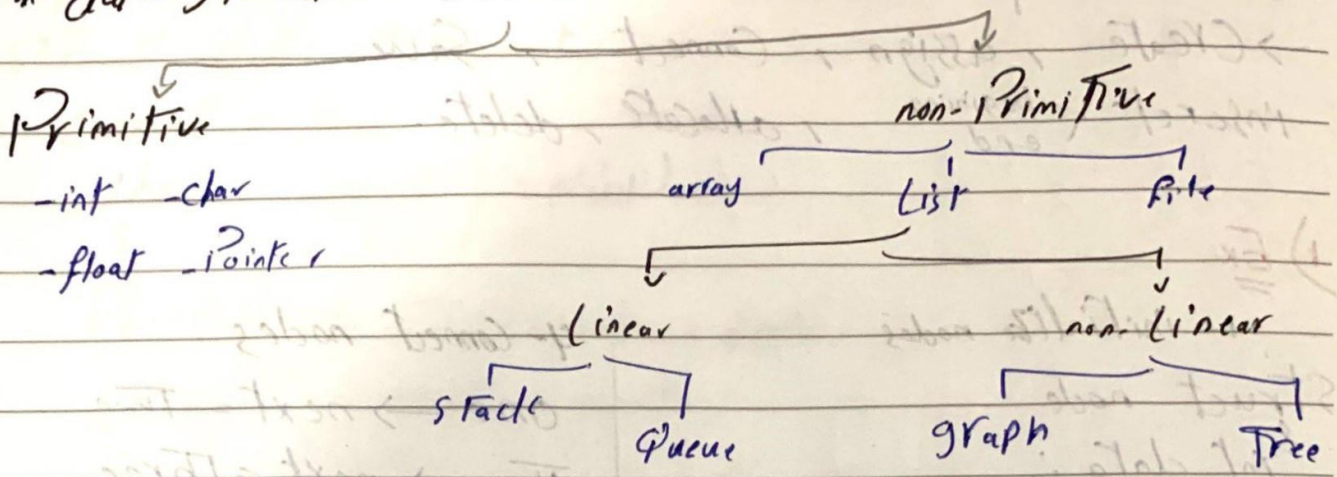
3) realloc()
→ adjust the space reserved
syntax
datatype *ptr = (dataType*) realloc (ptr, newsize);
N, size
Ex
int *ptr = (int*) Calloc (5, size of (int)) → 20 byte
Ptr = realloc (Ptr, (10, size of(int)); → 40 byte

4) free()
→ remove location
syntax
free (Ptr);
Disadvantage
1- memory leak →
2- Memory Fragmentation →



nnre / / owecr

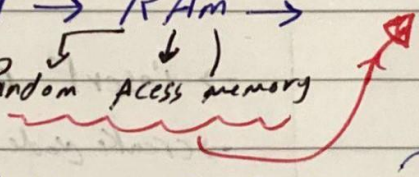
14# data structure Type



→ Linear data has Sequence .
→ non Linear data has no Sequence.

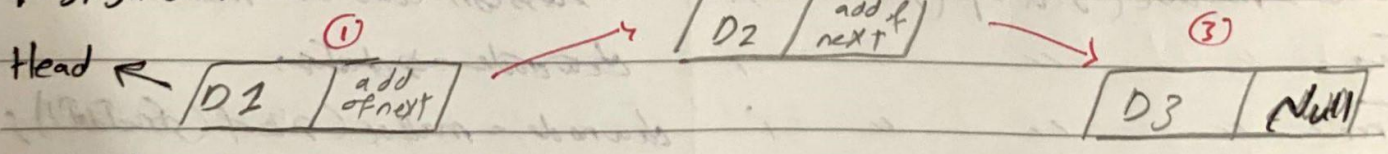
1- Linear data structure " stack, Queue, Linked list "

1- Linked list → RAm → Random Access memory

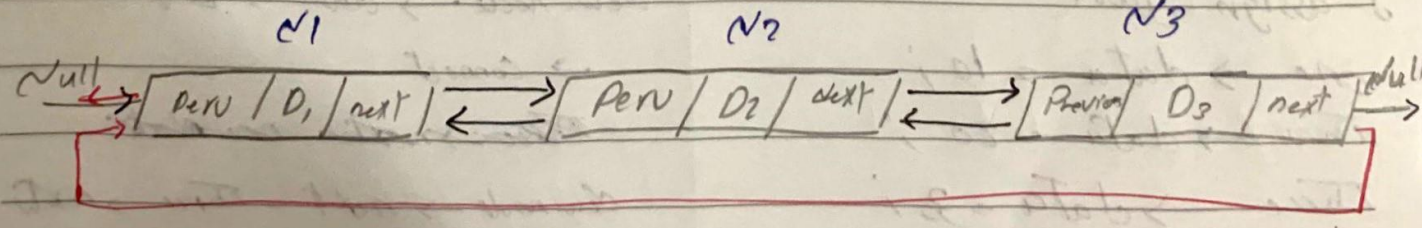


- data Can Link each other by Pointer

1-Single Link



2- doubly Linked list



Can access any node

Dare

DATE / / OBJECT

Operation of Linked list

→ Create, assign, Connect, Save

Insert ← beginning / end, allocate, delete.

1) Ex

1- initialize nodes

```
struct node
{ int data;
struct node *next;
};
```

```
main()
struct node *Head;
" " *one = Null;
" " *Two = Null
" " *Three = Null
```

2-allocate memory

```
one = malloc(size of(struct node));
Two = " " " ;
Three = " " " ;
```

3-assign data

```
one -> data = 10;
Two -> data = 20;
Three -> data = 30;
```

4- Connect nodes

```
one -> next = Two
Two -> next = Three
Three -> next = Null
```

5) Save address

```
Head = one;
```

→ insert at end

→ create node

```
struct data * New node = Null;
```

→ assign allo cate memor

~~New node -> data;~~

```
New node = malloc(size of(struct Data));
```

→ assign data

```
New. node -> data = 40;
```

→ Connect

~~Three -> next = new node;~~

```
New node -> next = Three -> next;
Three -> next = New node;
```

owecr

DATE / / OBJECT

* for Print data

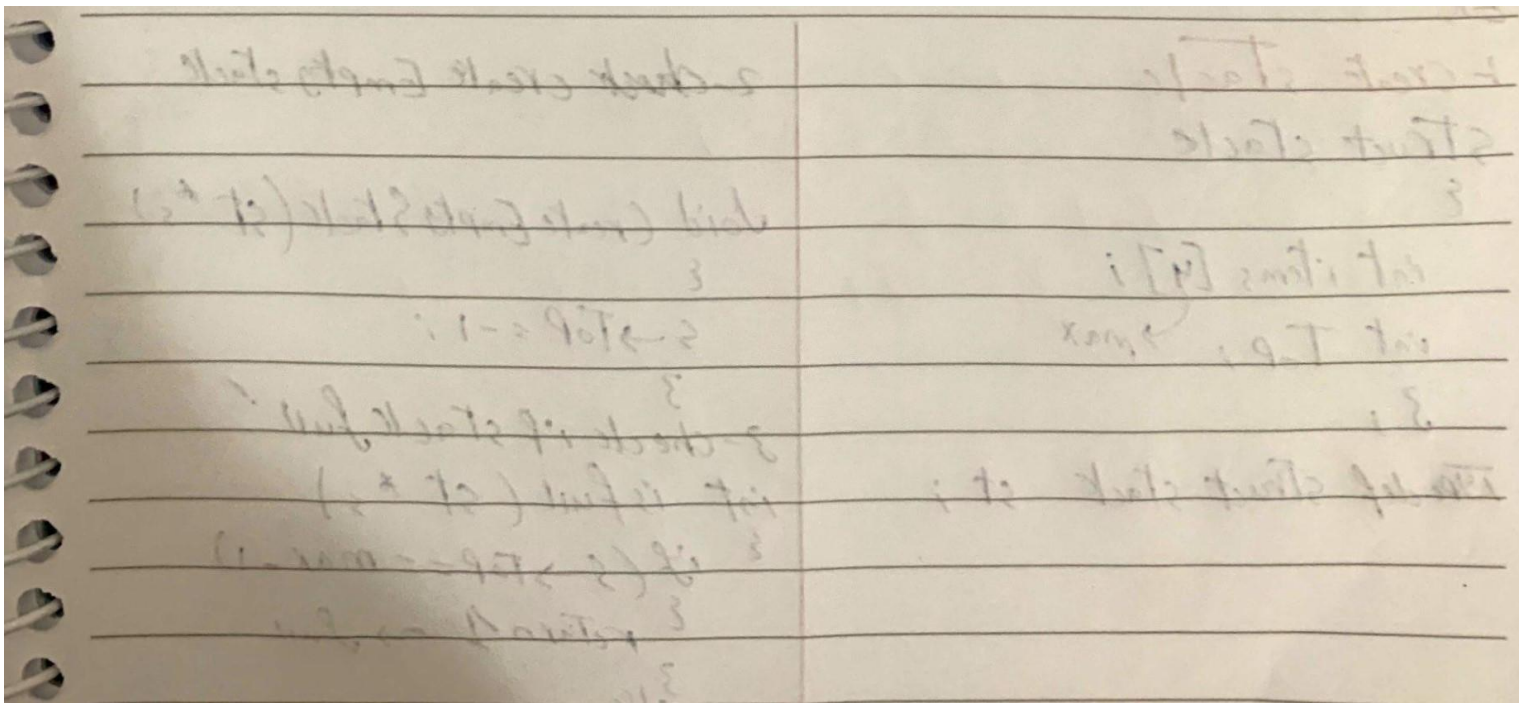
```
struct data *current ;
  current = Head ;
  while ( current != Null )
   {
   Printf ( "%d", current -> data) ;
   current = current -> next ;
   }
```

→ insert at first

→ insert at delete node

"check my git hub" ⇒ mahmoud ramadan ⇐

nnre / / owecr



owecr

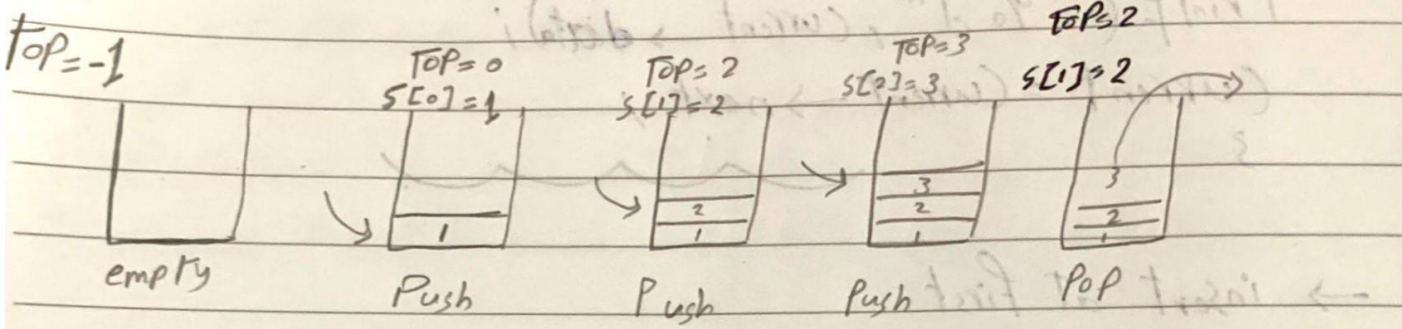
DATE / / OBJECT

2- Stack → ADT → abstract data type

→ Pile of Plates kept on top of each other.

LIFO → Last in first out

operation → Push , Pop , is empty , is full



Push → TOP++ Pop - TOP--

steps

1- initialize Pointer Called TOP and assign it to -1

So can check both increment "push" decrement "pop"

Ex

1- create stack

```
struct stack
{
   int items [4];
   int Top;
};
typedef struct stack st;
```

2- check create Empty stack

```
void CreateEmptyStack (st *s)
{
   s->TOP = -1;
}
```

3- check if stack full

```
int isfull (st *s)
{ if (s->TOP == max-1)
   { return 1 → full
   }else
   { return 0 → not full
   }
```

SAGDA

nnre / / owecr

4) check for empty

```
int IsEmpty (st *s)
{
if (s->top == -1)
  {
  return 1;
  }
 else
 {
 return 0;
  }
}
```

5- Push "add"

```
void Push (st *s, int newitem)
 {
   if (isfull(s))
     {
     Printf ("stack full");
     }
     else
     {
  } s->top++;
    s->items [s->top] = newitem;
   }
}
```

6- Pop item

```
void Pop (st *s)
 {
if(isEmpty (s))
  {
  Printf "stack Empty");
  }
 else
 {
  s->top--;
  }
}
```

owecr

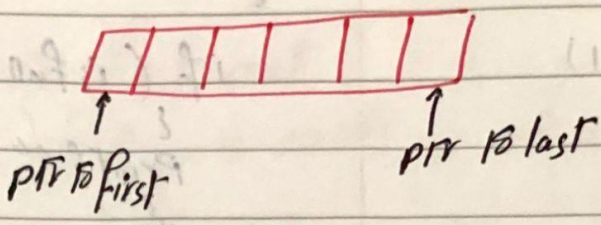
nnre / / owecr

Dare / / OBJECT

3) Queue →

FIFO → first in first out

operation
→ enqueue
→ dequeue
→ is empty
is full



* work principle

Two pointer → first & last

Set value of last = -1 & first = -1

1- Enqueue
- check is queue full
- Set value of first ptr = 0
- increase last ptr by 1
- add element in position of last ptr

2- Dequeue
- check is queue Empty
- return value pointed by first
- increase first by 1
- for last element set first & last to -1

owecr

DATE / / OBJECT

Ex

```
#define SIZE 5
void enqueue(int);
void dequeue(void);
int item[SIZE], First = -1, last = -1
main()
{
dequeue();
enqueue(1);
" (2);
" (3);
" (4);
" (5);
" (6); → Queue full
```

```
void enqueue(int value)
{
  if (last == size-1)
    { "queue full"
    }
  else
    {
      if (First == -1)
        { First = 0
        last++;
        item[last] = value;
        }
    }
}
void dequeue(void)
{
  if (frist == -1)
    {
      queue empty
    }
  else
  {
    First++;
    if (First > last)
      {
        first = last = -1
      }
  }
}
```

Notes for C

1- Same name ⟶ Combination error
else if created within block

2- Printf ("%d", x, y, z); ⟶ Print x
" " , (x, y, z)) ⟶ " z

3- continue; ⟶ skip one loop
4- break; ⟶ exit loop
5- while (1); ⟶ infinite loop
6- if (condition); condition
□; ⟶ if valid ⟶ execute this line
□; ⟶ valid forever.

7) if () - else ⟶ Cannot unplug them by logic

```
if ( )
{
}
≠ logic (XXX)
else
{
}
```

8) Can not assign string from struct should use strcpy fun

9) extren int x;
int x; ⟶ error x ⟶ undefined

10)

Dare

DATE / / OBJECT

10) Volatile → 1- for access by Hw or Reg "in/G/view dues
2- save from compiler optimization.

11) Size of Pointer = word size

owecr