JULY '08

11 FRIDAY

DAY 193-173 | WK 28

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	
8	9	10	11	12	13	
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

8

9

10

11

11

12

1

2

3

4 man 1 → Linux command

man 2 → system calls

5 man 3 → library function

6

NOTES

OPERATORS

→ Presidence decides which operation will be done first in an expression, associativity decides the order of evaluation
45- operators

	Presidence	Associativity
1st Highest		
( )	parentheses	L to R
[ ]	bracket	
.	dot	
→	arrow	
2nd HP		
++	plus plus	R to L
--	minus minus	
!	not	
~	1's compliment	
&	address of	
*	indirection	
sizeof		
3rd HP		
/	division	L to R
*	multiplication	
%	modulo	
4th HP		L to R
+	plus	
-	minus	

JULY '08

14 MONDAY

DAY 196-170 | WK 29

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

5th HP

Bitwise operators:
- >> right shift
- << left shift

L to R

6th HP

Relational operators:
- > greater than
- < less than
- >= greater than equal to
- <= less than equal to

L to R

7th HP

- == equal to
- != not equal to

L to R

8th HP & (AND)
9th HP ^ (XOR) [similar then 0]
10th HP | (OR)

Bitwise (L to R)

11th HP || (OR)
12th HP && (AND)

Logical (L to R)

13th HP ?: (conditional) R to L

14th HP
= (assignment)
+ =
- =
/ =
% =
* =
& =

compound assignment

R to L

NOTES

| =
^ =
>> =
<< =

15th HP (,) comma L to R

string - forming operator
string - merging operator

+ unary plus
- unary minus

⊙ Relational operators → return value 0 or 1
⊙ Logical operators (!, ||, &&) → return value 0 or 1

* Conditional (? :)

```
main()
{
    int k=10;
    if (k>=5)
        printf("Hello");
    else
        printf("Bye");
}
```

equivalent ←→

```
main()
{
    int k=10;
    k>=5 ? printf("Hello");
         printf("Bye");
}
```

32 keywords

auto
break
char const case continue
do double default
extern else enum
float for
goto
int if
Long
return register
short sizeof struct signed switch static
unsigned union
typedef
void volatile
while

→ variable name and function name is called an identifier, which cannot be a keyword.

watch code change

- post ++ → don't increment 'i' value only use
- pre -- → first increment (i) value then assign

→ operands + operator = arithematic expression

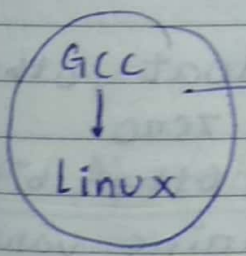
5 * 9 → binary operator
-5 → unary operator
5>=6 ? 5:6 → ternary operator

NOTES

DATA TYPES

datatypes (decides type of variable)		modifiers (decides size and range of variable)	qualifiers
bytes			
1 char	scalar	signed	const
4 int	scalar	unsigned	volatile
4 float	scalar	short	
8 double	scalar	long	
void	non-scalar		
enum	non-scalar		
struct	non-scalar		
union	non-scalar		

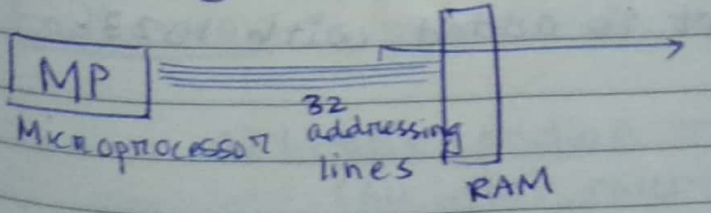
→ Memory — a) Logical
b) Virtual
c) Physical (RAM)



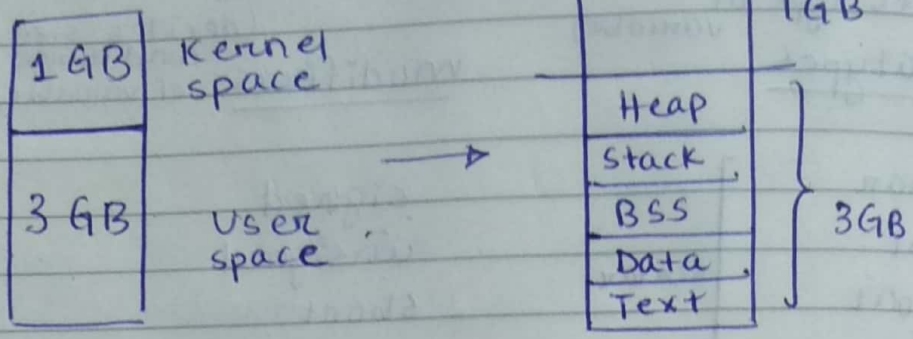
→ can access 4GB of memory
X 386 (80386 processor)

→ A set of wires connected from one device to another device in computer = bus

Two dedicated buses b/w microprocessor and RAM-
① Address bus
② Data bus



2^{32} = 4GB can be accessed



* Float abstraction / Double abstraction

no - cycle like unsigned char

4 bytes (=32 bits)

1 | 8 | 23

Sign | Exponent | Mantissa

→ In case of non-recurring float, other bits of mantissa filled with zero.

→ In case of recurring float, other bits of mantissa filled with recurring value.

(*) $1000.010 = 1.000010 \times e^{(3)}$ exp.

→ In case of float type, the exponent is added with 127, but in case of double type exponent is added with 1023.

NOTES

$$\begin{array}{r} 127 \\ 3 \\ \hline 130 = 10000010 \end{array}$$

→ 0 ⇒ sign +ve
1 ⇒ sign -ve

float storage

```
{
float x = 12.50
unsigned char *p = &x;
printf ("%d ", *(p+0));
printf ("%d ", *(p+1));
printf ("%d", *(p+2));
printf ("%d", *(p+3));
}
```

double

8 bytes = 64 bits

1	11	52
sign	exp.	mantissa

* INT Abstraction

short int ⇒ 2 bytes
int = long int ⇒ 4 bytes
long long int ⇒ 8 bytes

(Actually int depends on compiler)

Endianess = process of byte ordering

SUNDAY 20

a) Little endian → DCBA order
(All intel microprocessors)

b) Big endian → ABCD order
(IBM, Motorola, Spark)

JULY '08

21 MONDAY

DAY 203-163 | WK 30

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	
8	9	10	11	12	13	
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

90000

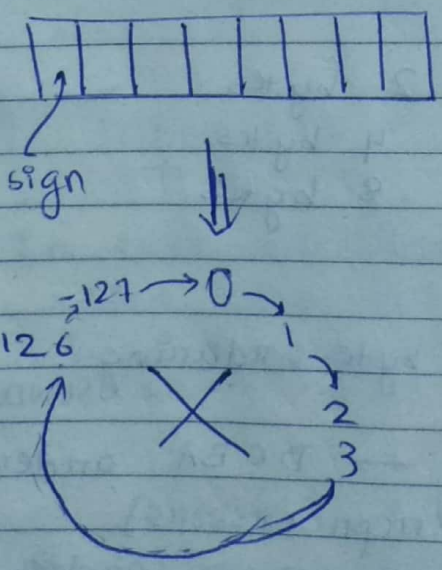
Bigo: 00000000 00000001 01011111 10010000

0 1 95 144

Little: 144 95 1 0

```
// main()
{
    long int x = 90000;
    unsigned char *p = &x;
    printf("%d", *(p+0));
    printf("%d", *(p+1));
    printf("%d", *(p+2));
    printf("%d", *(p+3));
}
```

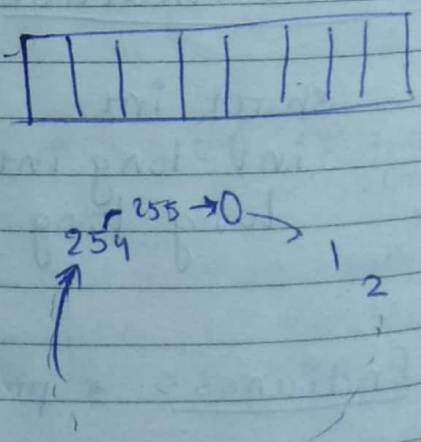
* CHAR Abstraction



Signed

-127, -126, ..., 0, 1, 2, 3, ...

-126, -127

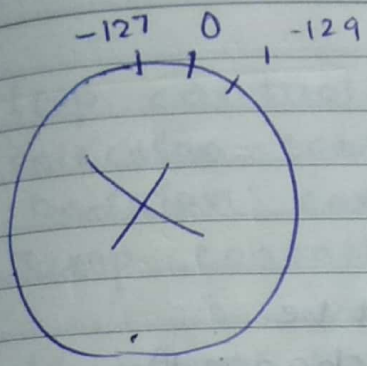


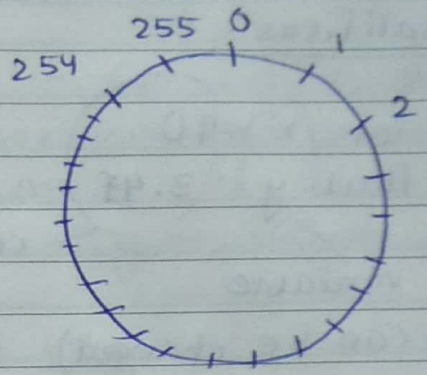
Unsigned

254, 255, 0, 1, 2, ...

... 254, 255

NOTES





Signed : To overcome problem of '-0', ANSI decided all negative data to be stored in memory in 2's compliment

→ 1's compliment +1 = 2's compliment

→ 2's compliment of char '-0' = -128

(1) 0000000

↓ 1's compliment

1111111

\+ 1

↓ 2's compliment

10000000 = 128 → (+) → -128

(*) char x = -130 ; ⇒ x = 126

-128 -129 -130

↳127 ↳126

JULY '08

23 WEDNESDAY

DAY 205-161 WK 30

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

* Qualifiers

```
int x = 90
float y = 3.4f
```

x, y → variable (value can be changed)

90, 3.4f → constants (value can't be changed)

* ASCII value - contains 65536 characters

char x = 'A' ≡ char x = 65

	0 – 9	A – Z	a – z
ASCII:	47 - 58	65 – 90	97 - 122

Enter → 10
Space bar → 32
Esc → 27
Tab → 9
Backspace = 8

'nul' is the very first ASCII character in the character set.

W/Get ASCII value

```
main ()
{
    char x;
    x = getchar ();
    printf ("%d\n", x);
}
```

NOTES

CONTROL STRUCTURE

(a) Loop control structure - for, while, do while
(b) selection control structure - switch, case, default
(c) Decision control structure - if, else
(d) Jump control structure - goto, break, return, continue.

All control structures are keywords.

→ A single statement falls under the default scope of a control structure. To put multiple statement under the scope, use curly braces.

→ Hanging if (error)

→ If-else statement ; curly braces must be in if

```
int i = 5;
if(i>=3)
        printf("Hello");
        printf("Bye");   → hanging if
else                       neither if nor
        printf("Ok");      else part.
```

→ for, while : entry control loop structure
do while : at exit ~~video~~ control loop.

JULY '08

25 FRIDAY

DAY 207-159 | WK 30

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Jump control

(1) break - transfers control just outside the loop

(2) return - tranfers the control to the end of the function. (like main())

(3) continue - transfers the control again to the beginning of the loop!

(4) Selection control

(switch-case)

→ Execution in switch-case starts always from match-case.

→ 'break' statement prevents the falling of control from match-case to next case.

→ If there is no match-case, then the default case will be executed.

→ switch-case is faster than if-else, because switch case execution is directly start from match case but execution in if-else is from beginning.

NOTES

%p → print pointer address

JULY '08

28 MONDAY

DAY 210-156 | WK 31

JUNE '08

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

POINTER

→ Pointer is a variable which refers address, address is a location in memory, which is 4 bytes long. And hence pointer size is 4 bytes irrespective of data type.

→ A pointer having n-indirection can hold the address of a pointer having (n-1) indirection.

```
int  k = 90;
int *p = &k;
int **q = &p;
int ***r = &q;
```

} Chain of pointers

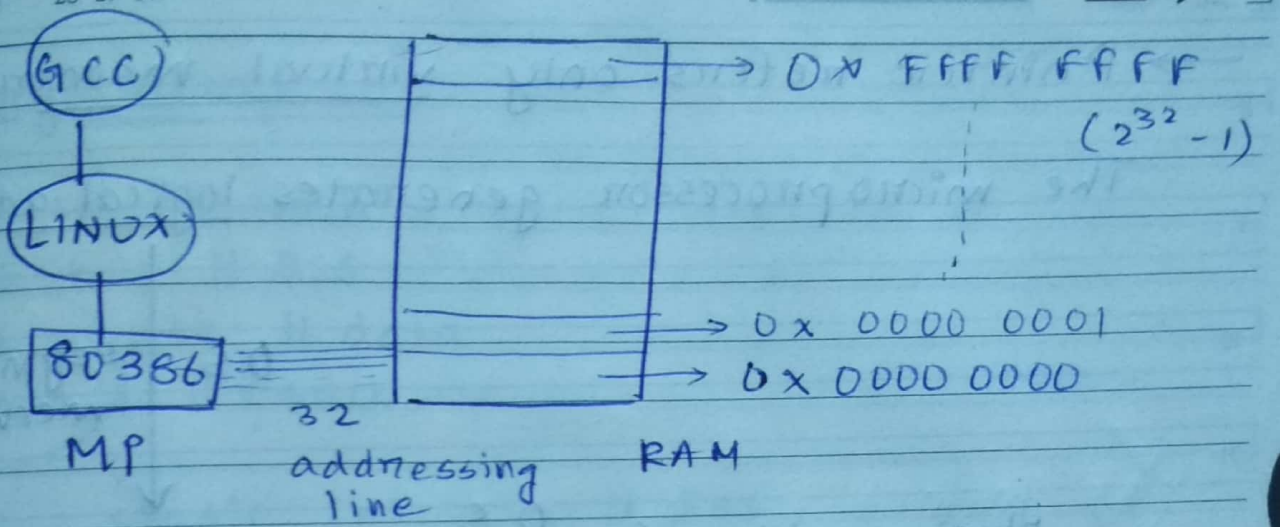
	***r	**q	*p	k
Value	0x1804933	0x080495ec	0x08049644	90
Address:	0x48066553	0x1804933	0x080495ec	0x08049644

→ Different appearance of pointer

```
main()
{
    int *p          // p is a pointer
    int *q[5]       // q is an array of pointers
    int (*r)[5]     // r is a pointer to array
    int *s()        // s is a function returns pointer
    int (*t)()      // t is a function pointer
```

*p-- ⇒ Here address is decremented

(*p)-- ⇒ Here value is decremented



→ Doing read or write operation using pointer the concept is called dereference.

→ If a pointer refers an invalid address (unauthorised = reversed by OS) in memory then called dangling pointer

→ If a pointer doesnot refer any address, in memory, it is called a wild pointer.

→ If a pointer of type 'void' is called a generic pointer. (void * p)

→ If a pointer refers starting address (or zero location) in memory, then it is called null pointer.

```
char *p = 0x 0000 0000 ;
char *p = 0;
char *p = NULL;
(All the p's are null pointer above)
```

30 WEDNESDAY
DAY 212-154 WK 31

→ Pointer refers only virtual memory

The microprocessor generates logical address

OS (segmentation mechanism)

virtual address

OS (paging mechanism)

physical address

Knowing these is called memory management

* Appln of NULL pointer

(i) It represents a stack is empty or not
(2) It " " queue " " " "
(3) " " the last node of linklist.
(4) " " " leaf " " tree.

* Appln of function pointer

```
int (*p)();
p = printf; // !(*printf)
```

now replace printf by (*p)("...");

NOTES

* Memory allotment

```
#include "stdio.h"
int k ; // Bss
int m = 10; // data
main() // stack
{
        static int a ; // Bss
        static int b=10; // data
        int c ; // stack
        int d = 10; // stack
        malloc (10) ; // heap
        calloc (1,10) ; // heap
}
int sum()   // stack.
```

User defined function

* Arithematic of pointer

i) Address ± Number = Address

ii) Address - adress = number

iii) Address + 1 = Next Address according to datatypes.

```
char *p = 500;
int *q = 500;
double *r = 500;
int (*s)[5] = 500;
p++;
q++;
```

```
r++;
s++;

printf("%d %d %d %d", p, q, r, s)

// 501  504  508  520
```

```
main()
{
    long int x;
    unsigned char *p = &x;

    *(p+0) = 144;
    *(p+1) = 95;
    *(p+2) = 1;
    *(p+3) = 0;
    printf("%d", x);      // 90000
}
```

x = 0 | 1 | 95 | 144 (Little)

*(p+3) *(p+2) *(p+1) *(p+0)

→ malloc(20) ≡ calloc(4,5)
≡ calloc(5,4)
≡ calloc(2,10)

memory allocation in 1-dimensional array equivalent

memory allocation in 2-dimensional array equivalent

NOTES

If a pointer refers address in same segments memory called a near pointer, while if a pointer refers address in other segments memory called a far pointer.

Appln. of pointer

1. Dynamic memory allocation
2. Hardware programming
3. Implementation of array
4. Implementation of function pointer.

TUESDAY 05
WK 32 | DAY 218-148

FUNCTION

Monolithic	Modular	Object-oriented (style)
C	C	C++
C++	C++	Java
		C#

→ Every function allocates memory in stack area. Stack size is limited. stack overflow becomes segmentation fault. Every functions are push and pop in stack in a LIFO order. (Last-in-First-out)

Monolithic

```
{
int a, b, c ;
printf ("Enter 2 nos. ");
scanf ("%d%d", &a, &b);
while (c=a%b)!=0)
{
    a=b;
    b=c;
}
printf ("%d", b);
}
```

Modular (Gives clean image)

```
int gcd(int, int) // function declrn
{
    int a, b, x;

    x=gcd(a, b); // function call
    printf("%d", x)
}
int gcd(int a, int b)// fn defn
{   int c;
    while ((c=(a%b)!=0))
    {
        a=b;
        b=c;
    }
    return b;
}
```

NOTES

06 WEDNESDAY
DAY 219-147 | WK 32

→ When control transfer to a function it takes time, when function is push into stack it takes time, when control return from a function it takes time, when function pop-up from stack it takes time, all together is called function overheading.

Most (cost of fn overheading)

```
int i
for (i=0; i<5; i++)
{
    bbsm();
}
```

```
bbsm()
{
    printf("Hi");
}
```

CTC → BBSR
5 times
(1 item each time)

Lesser

```
int i
bbsm()
```

CTC → BBSR
1 time
(5 items each time)

```
bbsm()
{
    int i;
    for (i=0; i<5; i++)
    {
        printf("Hi");
    }
```

Least (c++)

```
int bbsm()
main()
{
    bbsm();
}
```

```
inline int bbsm()
{
}
```

CTC → BBSR
online shop.

* Function recursion fR

→ When a function is called by itself, it is known as function recursion.

```
main()
{
    main();
}
```

→ Any problem solved using func. recursion could be solved by loop control structure and vice-versa.

Replace while in loop by if in fR and repeat the function call in fR

```
fn()
{
  while(i<=j)
  {
    i++
  }
}
```

→

```
fn()
{
  if(i<=j)
  {
    i++;
    fn();
  }
}
```

→ ① fR allocates memory but loop control structure doesn't allocate memory.
② Loop is faster than fR.

→ When a function is defined inside a class, it is called a method. All the methods are functions but vice-versa isn't true.

→ If multiple functions are defined having same name, called function overloading. (a c++ feature)

(*) Parameter

```
main()
{
    int x, a = 10, b = 20;
    x = sum(a,b)   // a & b → actual parameter
    printf("%d", x);
}
int sum(int p, int q) // here p&q → formal parameter
{
    ____
}
```

→ Actual parameter allocates memory but formal parameter doesn't.

(*) Function calling convention

a. call by value
b. call by address
c. call by reference (c++)

} function parameter technique

NOTES

→ To access local variable in another function, function calling convention is used.

BY VALUE

```
int swap(int, int);
main()
{
    int a=10, b=30;
    printf("Before %d %d", a, b);
    swap(a, b);
    printf("After %d %d", a, b);
}

int swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

BY ADDRESS

```
int swap(int *, int *)
{
    swap(&a, &b);
}

int swap(int *p, int *q)
{
    int temp = *p;
    *p = *q;
    *q = temp;
}
```

LIBRARY

→ In a program, fn declaration, definition including main() all are present in same file, while the c-program can't be used again, while some function may not be even utilised. But all fns are copy to the program. Thus memory to be occupied increases.

→ To overcome above problem, library is used.

Library	LINUX	Windows	Microsoft DOS
static	.a	.lib	.lib
dynamic	.so	.dll	X

Create a static library (all the fn. from library is copied to the program)

① gcc -c def.c -W (compile)

② ar csr sample.a def.c (create static library)

③ gcc test.c sample.a (link with static library)

④ ./a.out (to run the program)

Create a dynamic library (the funct. used in program are copy to program

① gcc -c -fPIC def.c -W (compile)

② gcc -shared -Wl,-soname, sample2.so -o sample2.so def.o (create dynamic lib.)

③ gcc test.c sample2.so (linking)

④ export LD_LIBRARY_PATH = `pwd` (set library path)

⑤ ./a.out (run)

→ A library is a pre-compiled object file which contain function definition in hidden format. Header file contains function declaration.

NOTES

13 WEDNESDAY
DAY 226-140 | WK 33

ARRAY

* Time complexity

→ It is a mechanism which measures the efficiency of an algorithm.

→ The efficiency of an algorithm is represented using a mathematical notation Big O.

→ Big O represents the complexity of an algorithm only in worst case situation; not in the best or average case situation.

Big O notation

O(1) → No loop
O(n) → one loop
O(n²) → Two nested loop
O(n³) → Three nested loop.
O(log n)
O(log(log n))
O(n log n)

→ Array is a variable which can store more than one similar types of elements. It is used to create a list.

→ No. of brackets represents number of dimension to an array.

NOTES

```
x[5]    // 1-D
y[3][5] // 2-D
z[2][3][5] // 3-D.
```

→ n-dimensional array is collection of (n-1) dimensional array.

" " → string constant
' ' → char

stack size - 10,240 KB
= 10,240 x 1024 Bytes } → Max^M memory can be allocable.

→ Accessing elements of array (traverse)
- pointer : *(x+i)
  subscript : x[i]

- x[3] = *(x+3) = 3[x] = *(3+x) refers same address.
  i[x-1] = x[i-1]

- x[i][3] = *(x[i]+3) = *(*(x+i)+3)

→ String : A string is a character array terminated with null character '\0'.

char a[] = "abcdef"
↳ "...." contains \0 at end by default.

- size of () measures length of a string including null character but strlen() measures the length of a string without null character.

- strcpy() copy the whole string including null character but strncpy() copy only specified number of character without null character.

NOTES

JULY '08

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- strcmp ( )

		GCC	ICC
0 = strcmp("abc", "abc")	=	0	0
strcmp("akc", "abc")	=	1	diff of ASCII k - b.
strcmp("abc", "akc")	=	-1	diff of ASCII b - K.

→ Row order traversal is faster than column order traversal because elements of an array present in row order.

→ memset is a f^n used to set the value of an array to a constant.

// memset (x , 0 , sizeof(x))

- x → array name
- 0 → to which array is initialized
- sizeof(x) → length of array

→ Array limitation in 'C' :

a) does not support negative indexing

b) array in C cannot hold disimilar kind of elements.

NOTES → a wild pointer can't store anything

ex : char *p

strcpy (p, "man");

printf ("%s", p);

}

→ segmentation fault.

18 MONDAY
DAY 231-135 | WK 34

STORAGE CLASS

* Types of storage class

→ extern
→ static
→ register
→ auto

Scopes in C

→ program scope
→ file scope
→ function scope
→ block scope

→ Storage class is a data structure used by the compiler which decides scope, storage, life and default initial value of a variable.

@ auto

→ default storage class of a local variable

int k; ≡ auto int k;

→ all auto variables automatically allocates and deallocates memory when a function is pushed and popped into stack.

```
int x = 10;   // weak variable
main()
{
    int x=90;  // strong variable
    printf("%d", x);
}
```

O/p → (90)

NOTES

(b) register

→ registers are the memory elements used by the microprocessor, m/p takes 1ns to read and write into register.
(m/p)

→ It has no address thats why a pointer can't refer to a register variable.

→ registers are suitable to implement in a loop counter.

```
main()
{
    int x;
    for(x=1; x<=8000; x++)
        printf("%d", x)
}
```

8000 x 250 ns

or

```
register int x
```

8000 x 1 ns

(c) extern

↳ tells the compiler either the variable is defined in same file or other files of a program.

↳ extern variable can't be initialised since it is a variable declaration.
↳ doesn't allocate memory but definition does.

NOTES

test.c

```
extern int x
printf("%d", x)
```

def.c

```
int x=90
```

gcc test.c def.c
↳ ./a.out = o/p = 90

20 WEDNESDAY
DAY 233-133 WK 34

(d) static

↳ means a variable can't be accessed in another file

↳ also a local variable, if static written before it, means it avoids reinitialized during function recursive call.

```
int k=90;
{
    bbsm()
}
bbsm()
{
    extern int k;  // if static ⇒ o/p = error.
    printf("%d ", k);
}
```

o/p → 90

AUGUST '08

22 FRIDAY

DAY 235-131 WK 34

JULY '08

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

PREPROCESSOR

Preprocessor directive (all begin with #)

(1) Macro expansion directives

#define is used to define a macro;
#undef is used to undefine a macro;

'cpp' read the source file & generate intermediate source file before compilation

source (test.c) —CPP preprocessing→ Intermediate (test.i)

```
#define MAX 10+2
main()
{
   int k = MAX/2 ;
   printf ("%d", k);
}
```

```
main()
{
   int k = 10+2/2;
   printf ("%d", k);
}
```

- Macro call is preprocessed but a function call is not preprocessed.

source		intermediate
SUM(5,6)	if function call →	sum(5,6)
SUM(5,6)	if macro call →	5+6

NOTES → typedef command is used to change name of a datatype

```
typedef int raja;
raja x;
```

→ typedef can change only datatypes but MACRO can change any types

#define ~~printf~~ pf printf
#define nani int

→ Also macro is preprocessed but typedef is not preprocessed.

(2) File inclusion directive (#include)

→ main objective of #include is copy the contents of include file and insert in source file.

"ravana.h" → searches both pwd as well as standard include path. (ie. /usr/include)

<ravana.h> → searches only standard include path

(3) Conditional directive

SUNDAY 24

#if, #else, #endif, #ifdef, #ifndef, #elif

NOTES

→ Nested comment is not allowed in 'C'. To overcome the problem of nested comment, conditional directives are used

JULY '08

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

(4) Miscellaneous directive (#pragma)

#pragma pack(1) removes the slack byte in structure or union.

gcc ⟶ when you involve gcc, it normally does preprocessing, compilation, assembly and linking.

cpp ⟶ (man cpp) The C preprocessor, often known as cpp, is a macro processor that is used automatically by the C-compiler to transform your program before compilation.

Os :- Linux
Compiler :- gcc
preprocessor :- cpp
assembler - as
linker - ld

NOTES

STRUCTURE AND UNION

```
Struct student  → structure name
{
    int roll;
    char name[20];      } data members
    int age;
};
main()
{
    struct student p={1,"s.das",23};   → structure variable
    printf("%d %s %d", p.roll, p.name, p.age);
}
```

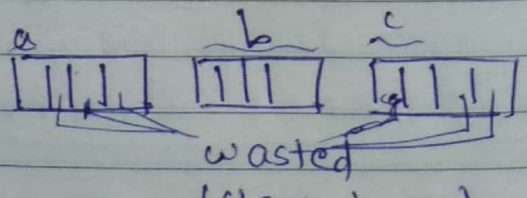
→ The data member of structure can only be accessed by structure variable.

• Memory allocation

→ Structure allocates memory blockwise. The block size depends on the longest data member of a structure.

```
char a;
int b;
short int c;
```

sizeof(struct student)); → to know size



#pragma pack(1) → removes slack bytes.

NOTES

→ slack byte is an undefined byte present inside the structure.

(a) size of a struct = size of all the data members present inside structure.

(b) size of an union = size of longest data member present inside union.

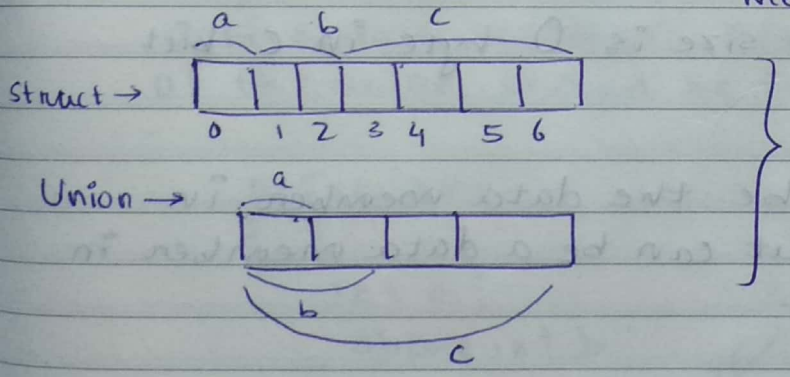
→ Each datamember of a structure begins at a different position in memory but each data member of union begins at same position in memory.

command –

printf ("%d ", offset of (union xxx, c))

↓

shows the location of member 'c'.



→ Union is a mechanism which permits to share a common memory and it is suitable for locking mechanism. (Mall-database example)

→ Structure is a mechanism which groups different data member to form a record. It is suitable to create database, create linklist, create bitfield.

→ A structure is nested within another structure is called a nested structure.

→ If a structure is nested within same structure, but nested structure variable is a pointer, that is called a self-referential structure.

```
struct xxx
{
    int roll;
    struct xxx *M;
};
```

↓ to create linklist.

Diff. blw C and c++ structure

→ Empty structure size is 0 byte in C but 1 byte in C++.

→ Function can't be the data member in C-structure but can be a data member in C++ structure.

→ 'private' and 'public' access qualifier is not allowed in C-structure but allowed in C++ structure.

→ Inheritance is not allowed in C structure but allowed in C++ structure.

NOTES

* Bitfield (not applicable to 'float')

```
struct xxx
{
    unsigned char x : 2;
    unsigned short int b : 9;
    unsigned int c : 5;
};
```

→ no. of bits to store data ≥0 ≤8 (Max size bits of unsigned char.)

→ Bit-field suitable to create protocol header, file header, font design.

→ Very first data member of a union is called active data member. Active data member of the union should be the longest data member.

```
union xxx
{
    int c;
    short int b;
    char a;
}
```

01 MONDAY
DAY 245-121 WK 36

Decimal to IEEE 754

263.3

2 | 263	
2 | 131	1
2 | 65	1
2 | 32	1
2 | 16	0
2 | 8	0
2 | 4	0
2 | 2	0
2 | 1	0
0	1

263 : 1000 0011 1

0.3 : 0100 1100 1100 .....

I) 263.3

= 1000 0011 1 . 0100 1100 1100

= 1.0000 0111 0100 1100 1100 .. $\times 2^8$

0.3

0.3 × 2	0.6	0
0.6 × 2	1.2	1
0.2 × 2	0.4	0
0.4 × 2	0.8	0
0.8 × 2	1.6	1
0.6 × 2	1.2	1
0.2 × 2	0.4	0

2 | 135	
2 | 67	1
2 | 33	1
2 | 16	1
2 | 8	0
2 | 4	0
2 | 2	0
2 | 1	0
0	1

e = 127 + 8 = 135 = $(10000111)_2$
(bias)

Representation

S (1) e (8) m (23)

0 1000 0111 0000 0111 0100 1100 1100 110

0100 0011 1000 0011 1010 0110 0110 0110

4 3 8 3 A 6 6 6

→ 0 x 4383A666

NOTES

IEEE 754 to Decimal

$$A = (\overbrace{1000\ 1000}^{e}\ \overbrace{1000\ 1000\ 1000\ 0000\ 0000\ 0000}^{m})_2$$

s

$\rightarrow (-1)^{s} \cdot (1+m) \cdot 2^{e}$

$s = 1_2$

$e = (0001\ 0001)_2 = 17 - 127 = -110$

$m = (0001\ 0001\ 0000\ 0000\ 0000\ 000)_2$

$-1\ -2\ -3\ -4\ -5\ -6\ -7\ -8\ \ldots$

$m = 2^{-4} + 2^{-8} = 0.066406250_{10}$

$\rightarrow$ Value $= -1 \cdot (1 + 0.066406250) \cdot 2^{-110}$ ✓

SEPTEMBER '08

03 WEDNESDAY

DAY 247-119 | WK 36

AUGUST '08

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

SEPTEMBER '08

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

8 # STRINGS

replace by size reqd.

heap ← `char *city =(char *) malloc (sizeof (char) *size);`

↳ dynamic memory allocation

10 stack ← `char city[10]; // static memory allocation`

11 → Every string must be terminated by a 'null'.

→ strings can be initialized at the time of

11 of array declaration or using a loop in program.

12 ✓ `char *name = {'A', 'L', 'a', 'b', 'a', 'm', 'a', '\0'};`

(static memory allocation)

1

✓ `char string[10] = "Alabama"`

2 `char string[] = "Alabama"`

dynamic ← `char *name = "Alabama"`

3

4 → The no. of characters in the string exceed the maximum size, the string is truncated.

5

1. `char str1[] = "Hello"`

6 `char str2[10];`

`str2 = str1;` (// comp. error)

2. `char *s = "Good Morning"`

NOTES `char *p;`

`p = s; //works fine`

3. char str1 [] = "Hello";
   str1 = "bye" ; // Give error.

4. char *s = "Hello"
   s = "bye" // works fine

⇝ scanf ≡ gets
⇝ puts(name) ≡ printf("%s", name)

→ ASCII of '\0' = 0 ; ASCII of '0' is 48.

```
while (name[i] != '\0')
{
    printf("%c", name[i])
    i++;
}
```

(OR)

```
char *ptr = "King";
while (*ptr != '\0')
{
    printf("%c", *ptr);
    *ptr++;
}
```

→ Once the base address is obtained in ptr, *ptr would yield the value at this address.

%s ⟶ format specification for printing out string.

→ puts() can display only one string at a time
gets() is capable of receiving only one string at a time; but it can receive a multi-word string

Standard library String Functions-

✓ strlen : finds length of a string
strlwr : converts to lowercase
strupr : converts to uppercase
✓ strcat : appends one string at the end of another
strncat : appends first n characters of a string at the end of another
✓ strcpy : copies a string into another
strncpy : copies first n-characters of two strings
✓ strcmp : compares two strings
strncmp : " first n-characters of two strings
stricmp / strcmpi : " two strings irrespective of case
strrev : Reverses string.

(A) Strlen()

→ Counts the number of characters present in a string
→ we pass the base address of string concerned.

```
char arr[]= "Name"
strlen(arr);
OR
strlen("Name");
```
} syntax.

NOTES

(B) strcpy()

→ copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function.

Syntax: strcpy(char *t, const char *s);

so that string would remain constant
↳ const qualifier

ensures that your program does not inadvertently alter a variable that you intended to be a constant

Use of const

(1) char *p = "Hello" // pointer is variable, string is variable
*p = 'M'; // works
p = "Bye" // works

(2) const char *q = "Hello" (OR) char const *q = "Hello"
(string is fixed pointer is not)
*q = 'M'; // error
q = "Bye"; // works

(3) char * const t = "Hello" // pointer is fixed string is not
*t = 'M' // works
t = "Bye" // error

SUNDAY 7

(4) const char* const u = "Hello" // string & pointer fixed
*u = 'M'; // error
u = "Bye" // error

(c) strcat()

→ concatenates the source string at the end of the target.

strcat (target, source)

(d) strcmp()

compares two strings till →
a) there is a mismatch
b) one string ends.

strcmp (str1, str2)
return value = str1 - str2

* 2D Array of characters

char m[6][10] = {
"@kshay",
"parag",
"raman",
"srinivas",
"gopal",
"rajesh"
};

&m[1][0]

&m[4][0]

(@ Many loss of memory)

NOTES * Array of pointers to strings :

char *names[] = {
"akshay",
"parag",
"raman"
};

```
char *temp;
temp = names[2];
names[2] = names[3];
names[3] = temp;
```
} easy exchange

else in char array -

```
for(i=0; i<=9; i++)
{
    t = names[2][i];
    names[2][i] = names[3][i];
    names[3][i] = t;
}
```
} 10 exchanges

Prob:

```
char *names[6];
```
↓
we can't initialize by scanf!

⇒ int *j

This declaration tells the compiler that j will be used to store the address of an integer value. In other words j points to an integer.

	i	j
value-	3	65524
address-	65524	65522

⇒ Every time a pointer is incremented, it points to an address of immediately next location of its type.

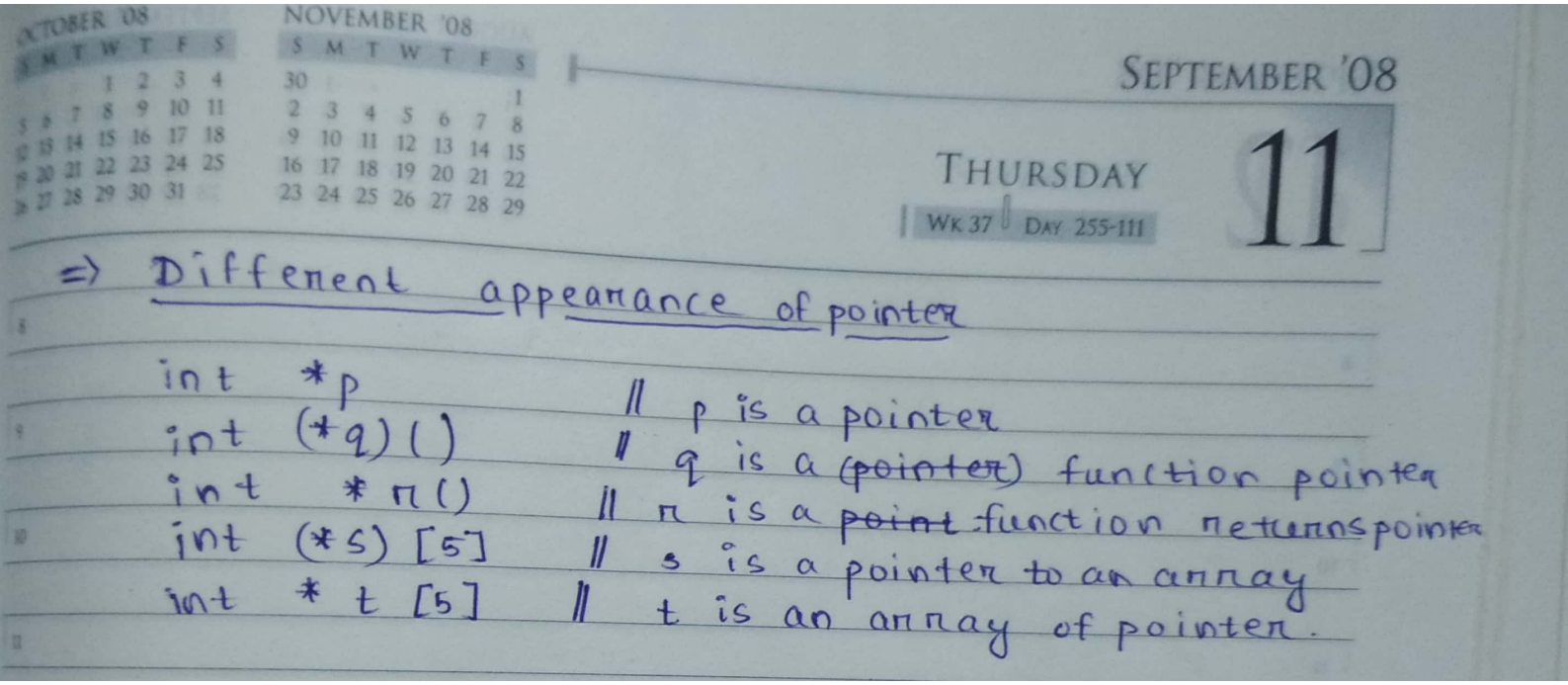
⇒ 1) Addition of positive and negative numbers never result in a wrong answer.

2) If sum of two positive numbers is less than zero, or sum of two negative numbers is greater than zero, then also there is an <u>overflow</u>.

⇒ Pass address to function, if both these conditions are satisfied

- the parameter value will be modified inside the function body.
- the modified value is needed in the calling function after the called function returns.

NOTES



OCTOBER '08

S	M	T	W	T	F	S
		1	2	3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

NOVEMBER '08

S	M	T	W	T	F	S
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

SEPTEMBER '08

THURSDAY 11

WK 37 | DAY 255-111

=> Different appearance of pointer

```
int  *p          // p is a pointer
int  (*q)()      // q is a (pointer) function pointer
int  *r()        // r is a point function returns pointer
int  (*s)[5]     // s is a pointer to an array
int  *t[5]       // t is an array of pointer.
```