PROBLEM SOLVING USING C

UNIT-1

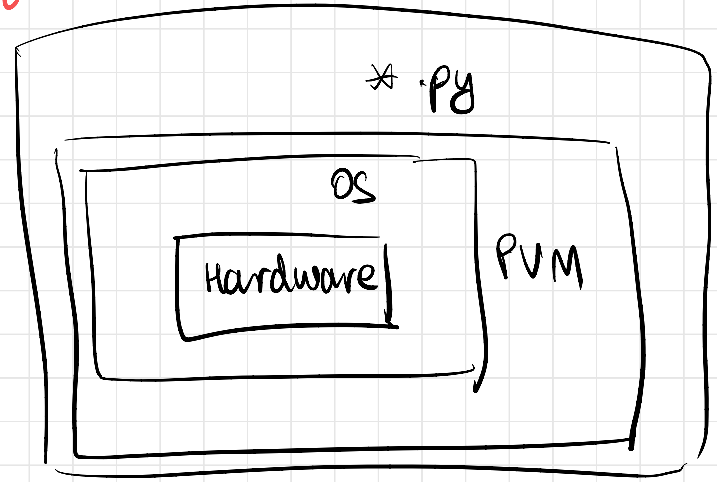
COUNTING

Vibha Masti

Feedback/corrections: vibha@pesu.pes.edu

Bhadri Prasad (L,O) (D,B) B112 opp

Python



C

- closer to hardware
- not as difficult as Python
- not as user-friendly

- C compilers use ASCII, not UTF 8

Structure of a C Programme

① Preprocessor directives

instruction to compiler; preprocessor

#include <stdio.h>

angular brackets for built-ins (" double quotes" for own files - modules)

any no. of header files

standard input/output header file

NOT LIKE import IN PYTHON

function headers

- stdio.h : contains function declarations
  header file with .h extension

- C files have .c extension
- gcc compiler using choice of editor
- minGW 64 or 32 for windows

② user-defined functions

- declarations here, not definitions
- starting point of execution of programme

③ global variables, constants

- comments: // — single line
- /* ... */ — multi-line

4 main

- beginning of programme
- no variable / function / file name with main to be used
- in C, code blocks are defined by curly braces {}, unlike python (indentation & :)
- indentation not important syntactically

return type — entry point — double quotes

```
int main() {
    printf("Hello world!\n");
    return 0;
}
```

function from stdio.h

all statements end with semicolon (expects ; at eos) syntax requires (syntax error)

- before a file is saved, only in RAM
- after saving, on HD
- to execute in python,

`python filename.py` — directly compiles & executes

- in C, compilation creates .obj file (assembly)

`gcc test.c`

$$\begin{tikzcd} \text{gcc test.c} \arrow[d, "\text{compiling}"] & & \text{a.out} \\ \text{test.obj} \arrow[rr, "\text{linking}"] & & \text{test.exe} \end{tikzcd}$$

test.exe – size larger than *.c

- to display o/p (printf), should interact with hardware
- linking shows location of printf (pointer)
- this location stores definition of function
- #include functions are linked while linking
- default return type: int

```
main () {
  ...
}
```

- even if no integer returned from main, executes with no warnings
- returns to shell
- successful completion of programme: `return 0;`

```
int main() {
      printf ("Hello world!\n");
}
```

no warning

- o/p still shown with warnings
- no return type: `void` return type

not recommended

```
void main() {
      printf("Hello World\n");
}
```

printf statements

```
#include <stdio.h>

int main() {
      printf("Hello, World\n");
      return 0;
}
```

std out

```
Hello, World
```

```
#include <stdio.h>

int main() {
      printf("%s", "Hello, World");
      return 0;
}
```

std out

```
Hello, World
```

```
#include <stdio.h>

int main() {
      printf("%s %s", "Hello");
      return 0;
}
```

std out

```
HelloP⍰g
```

ASCII characters stored after "Hello"

Datatypes in C

1) char
2) float
3) double
4) int
5) void

compiler decides no. of memory locations for each datatype

CHAR - 1 byte

ASCII : 8-bit / 1 byte char

UTF-8 : Unicode Transformation Form , 8-bit

UTF-16 : Unicode Transformation Form , 16-bit

Unicode is 16bit by default
65 535 symbols

ASCII for

0 :	0x30	48
9 :		58
A :	0x 65	
Z :	0x 91	
a :	0x 97	
z :	0x BD	

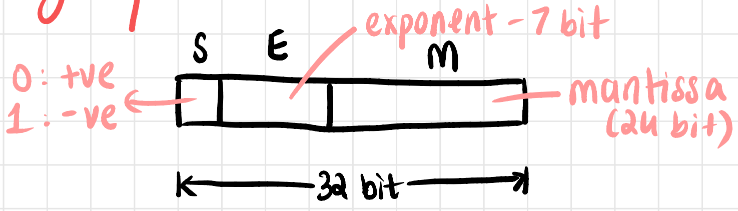
Homework .

FLOAT - 4 bytes

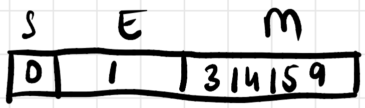
- floating point number
- decimal point can move wherever

$$0.314159 \times 10^{1} = 3.14159$$
$$3.14159 \times 10^{0} = 3.14159$$
$$31.4159 \times 10^{-1} = 3.14159$$

single precision number (SPN)



• for the number pi = 3.14159 = 0.314159×10^1



$5.8321 \times 10^{-6} = 0.58321 \times 10^{-5}$ ← 2s complement

* excess 7 notation

Q1. $-0.15629 \times 10^{+8}$

$S = 1$

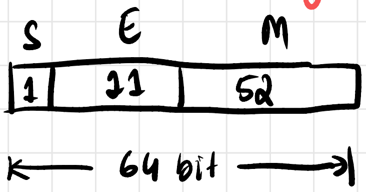
$E = (8)_{10} = 1000 = 0001000$ (7 bits)

$M = (15629)_{10} =$

flops : floating point operations per second

• in C, SPN are stored as floats

DOUBLE – 8 bytes



INTEGER

- stores integers
- today, 4 bytes (used to be 2)

- short : 1 byte
- int : 4 bytes
- long : 8 bytes

Long integers (64-bit)

- no. of ints : 2^{64} (long)
- unsigned : 0 to $2^{64}-1$
- signed : -2^{63} to 0 to $2^{63}-1$

32-bit int

-2^{31} to 0 to $2^{31}-1$

Variable declarations

```
int main() {
    int a;       —— single int
    int b,c;     —— comma-separated multiple declaration

    a = 10;   ┐
    b = 20;   ├── definition
    c = a+b;  ┘
// using printf on ints
① printf(" Sum of a and b is ", c);

// does not print c
// output:
//
// Sum of a and b is

// for C, the type of variable needs to be specified
// in the printf string
② printf(" Sum of a and b is %d\n", c);

// output
//
// Sum of a and b is 30


③ printf(" Sum of %d ", a, "and %d ", b, " is %d\n", c);

// output
//
// Sum of 10
// does not print beyond a as it expects only int
// unlike python
// must use required control strings
④ printf("Sum of %d and %d is %d\n", a, b, c);

// output
//
// Sum of 10 and 20 is 30
}
```

Control strings for printf (Controls type of i/p or o/p)

1) int
   - %d → decimal
   - %x → hexadecimal
   - %o → octal
   - %ld → long int

2) float & double
   %f

3) char & char array (string)
   - %c — char
   - %s — string

Datatypes

int	4	(integer)
float	4	(SPN)
double	8	(DPN)
char	1	(ASCII)

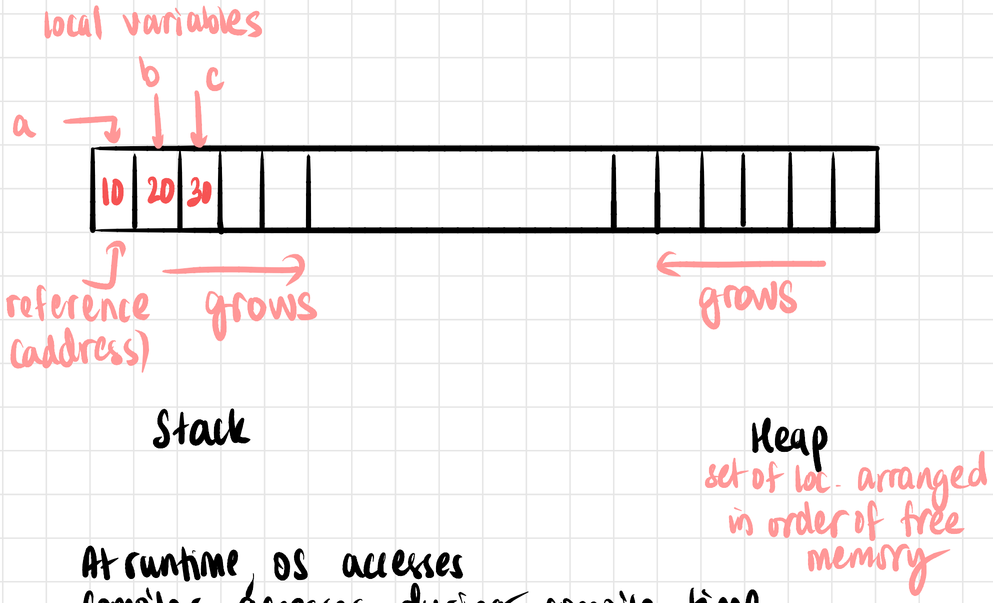
Dependent on compiler

Q: What happens here?

```
int a,b,c;

a=10;
b=20;

c=a+b;

printf("\n Sum of %d and %d is %d\n", a,b);
```

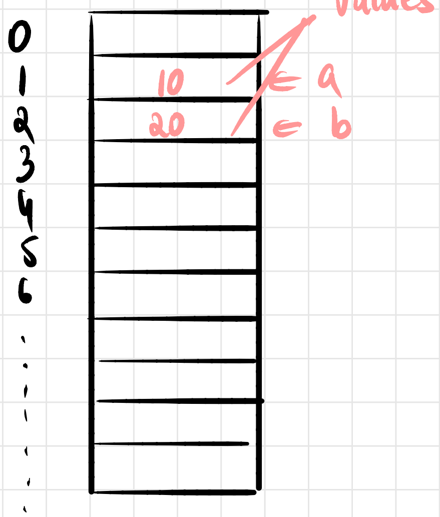
- Output is junk (garbage)
- Picks next location contents

- Brief intro to stack & heap



At runtime, OS accesses
Compiler accesses during compile time

Memory



Operators

1. +, -, *, /, % — arithmatic (modulus ↓ %)

2. <, >, <=, >=, != , == — relational

3. = — assignment (right to left)

4. +=, -=, *=, /=, %= — shorthand

5. AND → &&, OR → ||, NOT → ! — logical

6. Bitwise operators:

addition / or → |
multiplication / and → &
xor → ^
bitwise shift left → <<
bitwise shift right → >>
bitwise complement → ~ (ones complement)

7. incrementor → ++] right to left
decrementor → --

Prefix / Pre increment
```
int a=10;
++a;   ← increments & returns inc value
```

Postfix / Post increment
```
int a=10;
a++;   ← increments & returns original value
```

```
int a=10, b=10;

printf("%d\n", a++);   // Output: 10

printf("%d\n", ++b);   // Output: 11
```

• Works same for decrement

8. (type) — casting] right to left

9. * — dereference] right to left
& — address of

Associativity

1. $4 * 8/7 + 3 \% 4$ arithmatic $\xrightarrow{}$ L to R

(using stack)

$$\left[\left(\left(4*8\right)/7\right)+\left(3\%4\right)\right] = (32/7) + (3\%4)$$

integer division

$$= 4 + (3\%4)$$

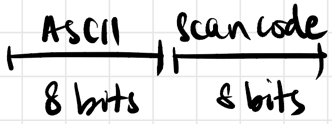
$$= 4+3 = 7$$

try: `float c;`
`c=3/4;` ⟶ c=0.000...

scanf

whatever is typed on keyboard is has a scan code unique to each key

scan code is 16 bit data



Syntax

type of data to store

address of location to store

```
scanf("control string", &variable);
```

whitespaces not necessary

```
eg: int c, d;
    scanf("%d %d", &c, &d);
```

Q: Write a programme to read no. of hours per week an employee has worked and the gross pay he gets per week calculate average per hour.

```
float hours, weekpay;

printf("Enter no. of hours per week: ");
scanf("%f", &hours);
printf("Enter gross pay per week: ");
scanf("%f", &weekpay);

float perhour = weekpay/hours;
```

```
int rupee = (int) perhour;
int paise = (perhour - rupee)*100;

printf("Employee gets paid an average of %f
        rupees and %f paise every hour\n",
         rupee, paise);

return 0;
```

Types of Statements

1. Simple statement
- in Python: expression

```
a = 10;
x = x + y*y;
printf("____");
scanf("____");
```

2. Compound statement
- in Python: block

```
{ ____ }
```

3. Control statement

a. Decision statement

i. if statement:

```
if (condition) statement;
            or
if (condition)
        statement;
        or
if condition {
        statement 1;
        statement 2;
             :
}
```

ii. if-else statement

```
if (condition) {
        statements;
}
else {
        statements;
}
```

iii. nested if

```
if (condition) {
        statement;
}
else if (condition)
        statement;

        else if (condition)
                statement;
                    :
                    :
                else statement;
```

iv. ternary operator

```
(condition)? (stm for true): (stm for false);

big = (a > b)? a : b;
```

Q: Read a number. Check if it is odd or even

```
int n;
printf("Enter a number: ");
scanf("%d", &n);

// using bitwise

if (n & 1)
    printf("%d is odd\n", n);

else
    printf("%d is even\n", n);


// using % 2

if (n %2)
    prinf("%d is odd\n", n);

else
    printf("%d is even\n", n);
```

Q. Check if no is +ve, -ve or 0.

```
int n;

printf("Enter a no: ");
scanf("%d", &n);

//method 1
if (n > 0)
    printf("%d is +ve\n", n);

else if (n < 0)
    printf("%d is -ve\n", n);

else
    printf("%d is 0\n", n);


//method 2

if (n & 0x80000000)
    printf("%d is -ve\n", n);

else if (n == ~n)
    printf("%d is 0\n", n);

else
    printf("%d is +ve\n", n);
```

Q: Read hexadecimal no. and display its decimal equivalent

```
//use %x in scanf
// takes 0-9 and ABCDEF

int num;
printf("Enter a number in hex: ");
scanf("%x", &num);

printf("The no. in hexa is %x\n", num);
// case-insensitive

printf("The dec equivalent is %d\n", num);
printf ("The bin equivalent is %b\n", num);
```

check this] homework

Q: Read int and display hex equivalent

```
int num;
printf("Enter dec no: ");
scanf("%d", &num);

printf("Hex equivalent of %d is %x\n", num, num);
```

Q: Read hex no. & check if +ve, -ve, 0

//check &

// ABC12345

```
  1010 1011 1100 0001 0010 0011 0100 0101
MSB ←
& 0x8 0    0    0    0    0    0    0
```

// code - try with %d

```
int num;
printf("Enter a no: ");
scanf("%x", &num);

if (num & 0x80000000)
        printf("%x is -ve\n", num);
else
        printf("%x is +ve\n", num);
```

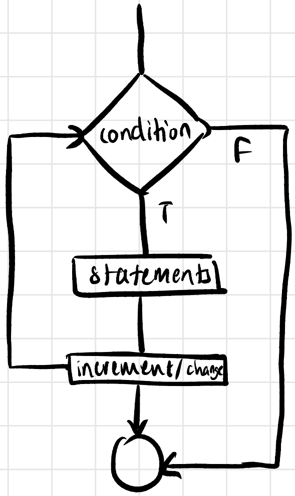
NOTE: unsigned ints do not have a signed bit

4. Iterative statement (loops)

1. while statement

syntax:

```
while (condition) {
        S1;
        S2;
        :
}
```



Q: (H/W) count no. of 1s in a number's binary representation

called checksum - security check
parity check: even parity (even 1s)
odd parity (odd 1s)

```
// using bin operators

int n, count = 0;
printf("Enter a no: ");
scanf("%d", &n);

int copy = n;
```

```
while (copy) {
    if (copy & 1) ++count;
    copy = copy>>1;
}
printf("No.of 1's in %d (bin) is %d", n, count);
```

1101	1111	1100	0000	1010	0000	0110	0001
D	F	C	0	A	0	6	1

```
//using hex input
int n;
printf(" Enter a no: ");
scanf("%x", &n);
printf(" Hex=%, Dec=%d", n, n);
int copy=n, count=0;
while(copy) {
    if (copy &1) ++count;
    copy = copy>>1;
}
printf("No.of 1's in %x (bin) is %d", n, count);
```

Reading characters

```
char ch;
```

Method 1

```
printf("Enter char: ");
scanf("%c", &ch);

//display ASCII equivalent

printf("%c %d (dec) %x (hex) \n", ch, ch, ch);
```

Method 2

NOTE: ' ' → char
" " → string

```
printf("Enter char: ");
ch = getchar();      //scanf
putchar(ch);         //printf
```

Method 3

warning

```
printf("%c", getch());
```

Method 4

echo, warning

```
printf("%c", getche());
```

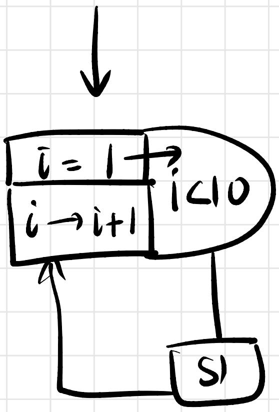
Q:- Read two numbers/digits using getch and convert it to int.

```
char dig;
int n =0;

while ((dig =getchar())!= '\n'){
    n *= 10;
    n += (dig-'0');
}

printf("No. is %d\n", n);

//reading no. of digits
```

2. For statement

syntax

```
for(initialisation; condition; incrementation) {
      S1;
      S2;
}
```



```
for (int i=0; i<10; ++i) {
      printf("%d. Hello\n", i+10);
}
```

Q: Find the sum of all odd no.s b/w 10 and 100

```
int sum = 0;
for (int i=11; i<100; i+=2) {
    sum += i;
}

printf("sum = %d", sum);
```

Q:- Find the sum of all even no.s b/w 10 & 100

```
int sum = 0;
for (int i=12; i<100; i+=2) {
    sum += i;
}

printf("sum = %d", sum);
```

Q: Check if a number is prime

```
int num, isprime = 0;
printf("Enter no: ");
scanf("%d", &num);

int root = sqrt(num);
if (i==2) isprime = 1;
else if (i%2 ==0) isprime = 0;

else {
    isprime = 1;
    for (int i=3; i<=root; i+=2) {
        if (num %i ==0)  isprime =0

    }
```

NESTED FOR LOOP

Q: Generate primes from 1 to 100

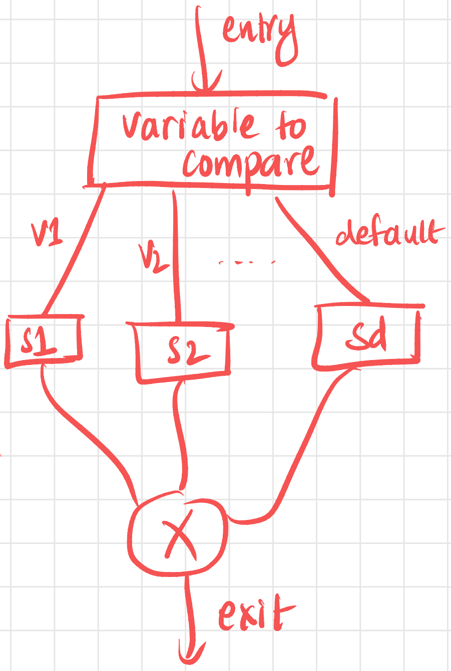
```
    printf("2\n");
    for (int i=3; i<100 ; i+=2) {
        int root= sqrt (i);
        isprime = 1;
        for (int j=3; j<= root, j+=2) {
            if(i% j ==0) [
                isprime =0;
                break;
            }
        ]
        if (isprime) printf("%d\n",i);
    }
```

5. switch statement

syntax

```
switch (var) {

    case v1:  s1;
              break;

    case V2 :  S2;
               break;

    default :  sd;
}
```

if not present, will fall through



• used for menu-based / driver programmes.

Q: Write a programme to calculate (use switch)

1. area of circle
2. area of rectangle / square
3. volume of sphere
4. volume of cylinder

```
int main() {
char ch;
float r, l, b, h;

int menu = 1;
```

```
while(menu){

    printf("Enter choice (1,2,3 or 4)\n");
    printf("1. Area of circle\n");
    printf("2. Area of rectangle\n");
    printf("3 Volume of sphere\n");
    printf("4. volume of cylinder\n");
    printf("5. Exit\n");
    printf("\n Your choice: ");
    scanf("%c", &ch);

    switch (ch) {
        case '1': {
                printf("Enter radius: ");
                scanf("%f", &r);
                printf("Area of circle = %.2f\n",
                                    3.14 * r*r);

                break;
        }
        case '2': {
                printf("Enter l: ");
                scanf("%f", &l);
                printf("Enter b: ");
                scanf("%f" &b);
                printf("Area of rect = %.2f\n", l*b);
                break;
        }
        case '3': {
                printf("Enter radius: ");
                scanf("%f", &r);
                printf("Volume of sphere = %.2f\n" 3.14/3
                                    *3.14*r*r*d);

                break;
        }
```

```
            case '4': {
                    printf("Enter r : ");
                    scanf("%f", &r);
                    printf("Enter h : ");
                    scanf("%f", &h);
                    printf("Volume of cylinder = %.2f",
                                        3.14*r*r*h);

                    break;
            }
            case '5': {
                    menu = 0;
                    break;
            }
            default: {
                    printf("Invalid choice\n");
            }
        }
}
```

Q: Read character by character till a word is read (char count and no. of words)

```
int main() {
        int count;
        char ch;
        int cc = 0;

        while (ch = getchar()) {

            switch (ch) {
                case ' ':
                case '\n':
                case '\t':
```

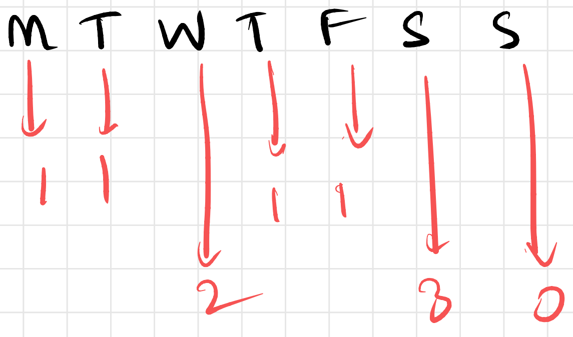
```
            case '#':
            case ',':
            case '!':
            case '?':
            case '.':
                     cont=0;
                     break;

      }

      if(!cont) break;

      ++c;
  }
```

enum

- labelling

```
enum week{sun, mon, tue, wed, thu, fri, sat};
```

- new pattern



```
enum week{mon=1, tue=1, wed=2, thu=1,
          fri=1, sat=3, sun=0};
```

```
enum week day = sun;
printf("%d\n", day);
```

by default: 0, 1, 2, 3 . . . .

- if any value explicitly defined, all following value continue counting
- for SQL, oracle, n-way trees implemented in C/Cpp