INTRODUCTION TO C

Invention of C

C is a programming language developed at AT & T's Bell Lab of USA in1972 by "Dennis Ritchie".

USE OF C

C is used mainly because it produces code that runs nearly as fast as code written in assembly language.

Some applications of C :

- Operating Systems
- Language Compilers
- Assemblers
- Language Interpreters
- Text Editors
- Modern Programs

Features of C:

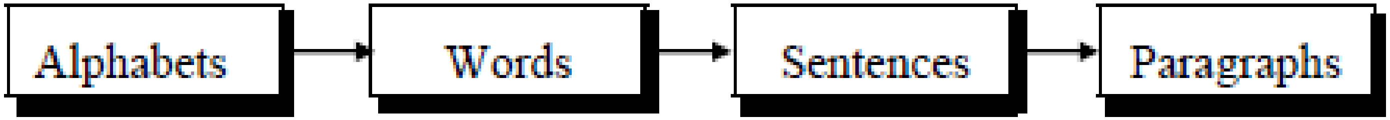
- Efficiency, high performance and high quality software.
- flexibility and power
- Stability and small size code
- Provide functionality through rich set of function libraries
- Gateway for other professional languages like C → C++ → Java

ANSI C

- **A**merican **N**ational **S**tandards **I**nstitute (ANSI C),

appointed a committee which approved a version of C in December 1989, which meets some standards.

Steps in learning English language:



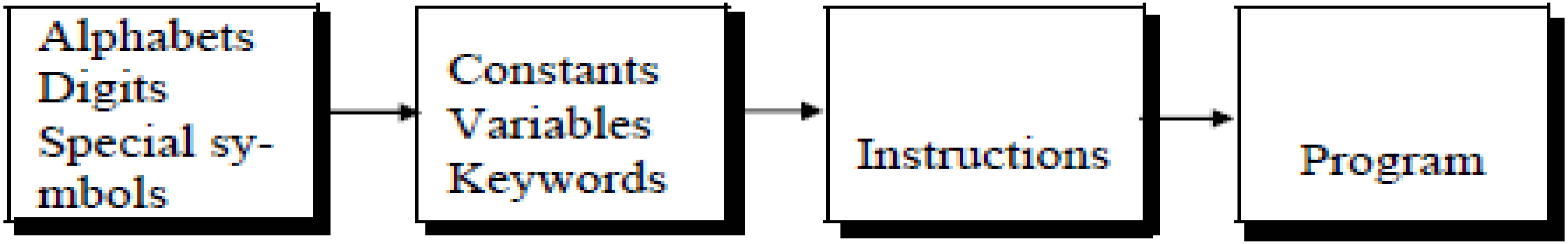
Alphabets

Words

Sentences

Paragraphs

Steps in learning C:



Alphabets
Digits
Special symbols

Constants
Variables
Keywords

Instructions

Program

VARIABLES

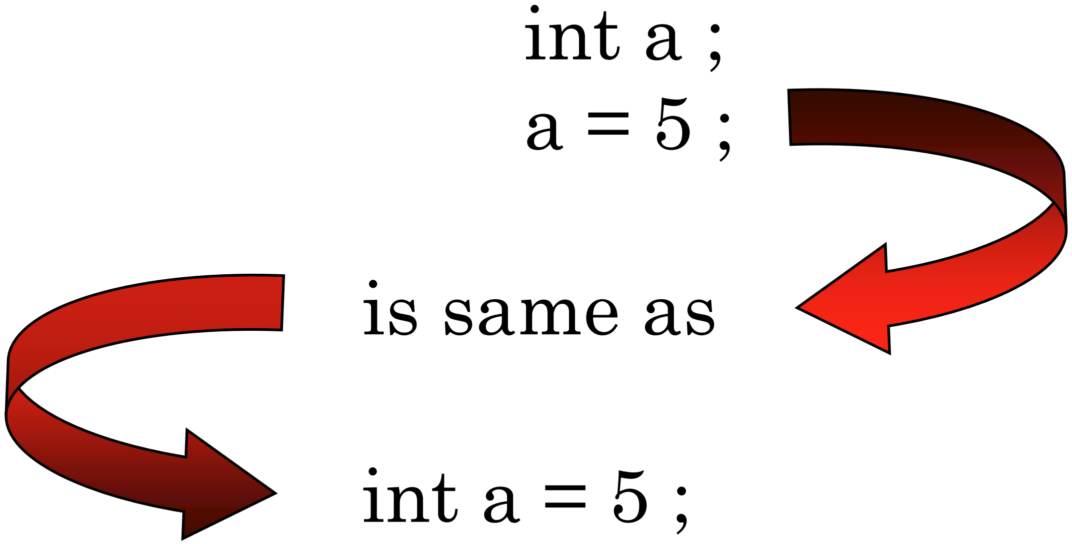
"A variable is an entity that may change."

- Variable names are the names given to location in memory.
- These location may contain integer, real or character constant.
- A particular type of variable can hold only that type of constant.

Rules

- A variable name is any combination of **alphabets, digits or underscores**.
- The first character in the variable name must be an alphabet or underscore.
- **Commas or blanks are not allowed** within a variable name.
- **No special symbol other than an underscore** can be used in a variable name.
- Ex.: si_int
  m_hra
  pop_e_89
  Thes

Type Declaration, A few Subtleties



```
int a ;
a = 5 ;
```

is same as

```
int a = 5 ;
```

è **int a = 5, b = 10, c = a + b * 5 % 2 ;**

Here order is important

Constants

- A constant is a entity that does not change.

- Types of C constants:

1. Numeric Constants
2. Character Constants

INTEGER CONSTANT

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.
- The allowable range for integer constants is -32768 to 32767.

Example: 426 ,
+782,
-7605

Real Constant

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.

- Exponential real constants: (mantissa +exponent)

$$.000342 = \underset{\text{mantissa}}{\underline{3.42}} \quad e \quad \underset{\text{exponent}}{\underline{-4}}$$

- Range: -3.4e38 to 3.4e38

Ex.: +325.34, 426.0, -48.5792, +3.2e-5

Character Constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

- Maximum length of a character constant can be 1 character.

Ex.: 'A'
'I'
'5'
'='

Exercise:

- Identify the type of constants

  ’+’

  786

  786.0

  ’e’

  7

  ’7’

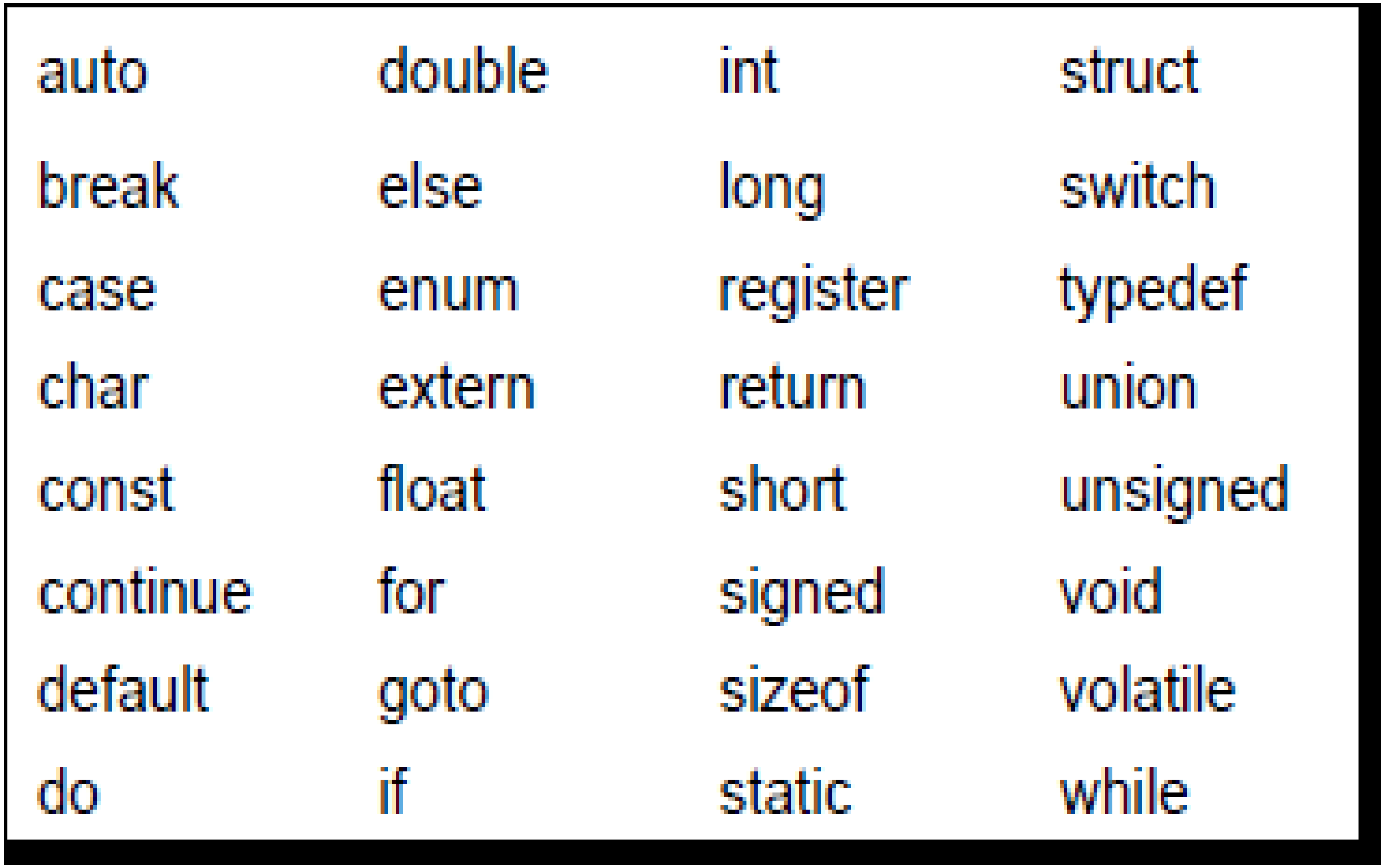
  ’E’

Keywords

- Keywords are the words whose meaning has already been explained to the C compiler.

- The keywords **cannot be used as variable names.**

- There are only 32 keywords available in C.



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

Simple C Program

```
/* A first C Program*/

#include <stdio.h>

void main()

{
        printf("Hello World \n");

}
```

Simple C Program

- **Line 1: #include <stdio.h>**

- Preprocessor directives are executed by the **C preprocessor**.

- In this case, the **directive #include** tells the preprocessor to include code from the file **stdio.h**

- This file contains declarations for functions that the program needs to use. A declaration for the **printf** function is in this file.

Simple C Program

Line 2: int main()

- This statement declares the **main function**.
- C program must always have one main function.
- A function is a self-contained module of code that can accomplish some task.
- The “int" specifies the return type of main. In this case, integer value is returned to the operating system.

Simple C Program

Line 3: {

- This opening bracket denotes the start of the program.

SIMPLE C PROGRAM

Line 4: printf("Hello World \n");

- **Printf** function from a standard C library that is used to print strings to the standard output, normally your screen.
- The compiler links code from these standard libraries to the code you have written to produce the final executable.
- The **"\n"** is a special format modifier that tells the **printf** to put a line feed at the end of the line.

Simple C Program

Line 5: }

- This closing bracket denotes the end of the program.

C code to Multiply two numbers

```
#include <stdio.h>
int main(  )
{
    int value1, value2, product ;
    printf(“Enter two integer values:”) ;
    scanf(“%d %d”, &value1, &value2) ;
    product = value1 * value2 ;
    printf(“Product = %d\n”, product) ;
    return(0);
}
```

FUNDAMENTAL DATA TYPES

Type	Storage size	Value range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615
float	4 byte	1.2E-38 to 3.4E+38
double	8 byte	2.3E-308 to 1.7E+308

Format specifier:

Format specifier	Type of value
%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lu	Unsigned long integer

FORMAT SPECIFIER:

- %d (print as a decimal integer)
- %6d (print as a decimal integer with a width of at least 6 wide)
- %f (print as a floating point)
- %4f (print as a floating point with a width of at least 4 wide)
- %.4f (print as a floating point with a precision of four digits after the decimal point)
- %3.2f (print as a floating point at least 3 wide and a precision of 2)

PRINTF & SCANF (EXAMPLE1)

```
int a;
float b;
scanf("%d  %f",&a,&b);
printf("%d %f", a, b);



double db=17.8;
char ch;
long int l;
scanf("%c  %lf  %ld",&ch,&db,&l);
```

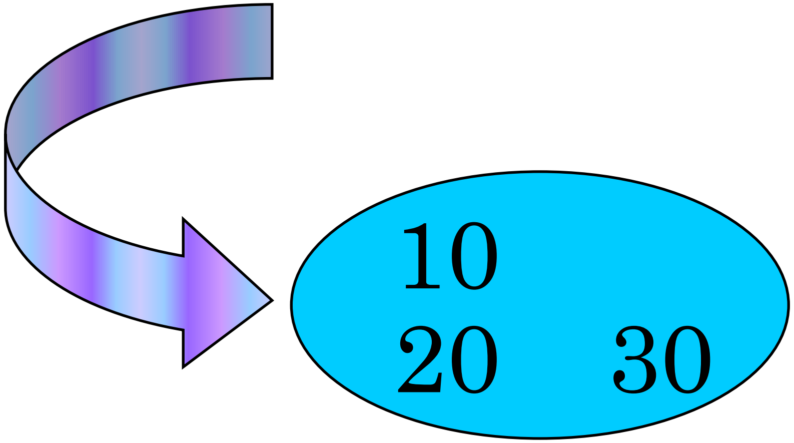
PRINTF & SCANF (EXAMPLE2)

```
int i=645;
float f=3.12;
long int l=3475;


printf("i = %d \n",i);
printf("i = %5d \n",i);
printf("i = %05d \n",i);
printf("f = %f \n",f);
printf("f = %2.2f \n",f);
printf("ld = %ld \n",l);
```

```
i = 645

i =   645

i = 00645

f = 3.120000

f = 3.12

ld = 3475
```

Escape Sequences

- \n – Newline
- \t - Tab
- \a – audible alert (bell)
- printf (”%d\n%d\t%d”, 10, 20, 30) ;



```
10
20    30
```

```
main( )
{
printf ( "You\tmust\tbe\tcrazy\nto\thate\tthis\tbook" ) ;
}
```

And here’s the output...

```
You      must    be      crazy
to      hate     this    book
```

Arithmetic Instructions

- To perform arithmetic operations between constants and variables.

***, /, -, +**	**arithmetic operators.**
=	**assignment operator.**

Increment

prefix	**++ a**
postfix	**a++**

Decrement

prefix	**--a**
postfix	**a--**

Shorthand operators

+=   -=

- prefix: increment/decrement takes place before the value is used in expression evaluation.

- Postfix: increment/decrement takes place after the value is used in expression evaluation.

Legal Arithmetic Operations

Operand1	Operand2	Result
int	int	int
float	float	float
int	float	float
float	int	float

Try This

```
float  a ;
a = 5 / 2 ;          →  2.0
a = 5.0 / 2 ;        →  2.5
a = 5 / 2.0 ;        →  2.5
a = 5.0 / 2.0 ;      →  2.5
a = 2 / 5 ;          →  0.0
a = 2.0 / 5 ;        →  0.4
a = 2 / 5.0 ;        →  0.4
a = 2.0 / 5.0 ;      →  0.4
```

Try This

int a ;

a = 5 / 2 ; → 2

a = 5.0 / 2 ; → 2

a = 5 / 2.0 ; → 2

a = 5.0 / 2.0 ; → 2

a = 2 / 5 ; → 0

a = 2.0 / 5 ; → 0

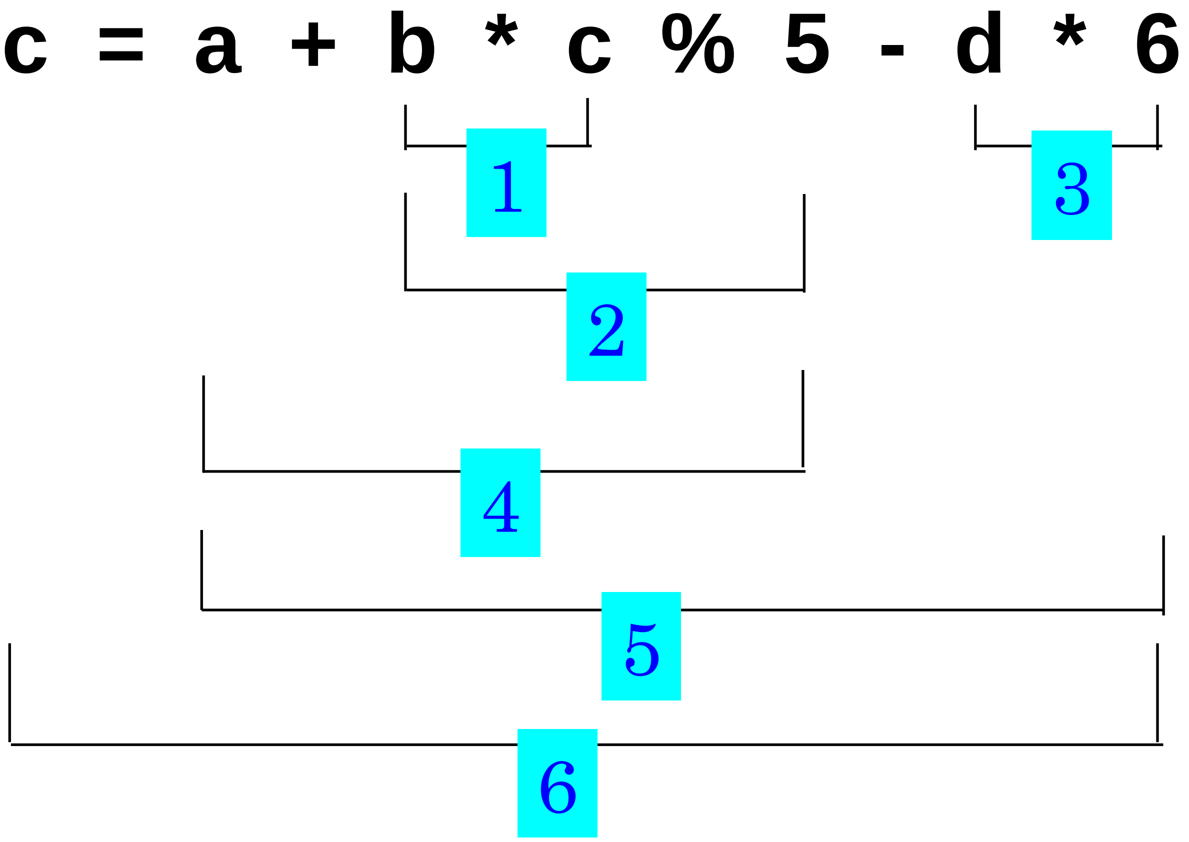
a = 2 / 5.0 ; → 0

a = 2.0 / 5.0 ; → 0

C Operator Precedence Table

Operator	Description	Associativity
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of *type*)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
| |	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	

HEIRARCHY/PRIORITY/PRECEDENCE



c = a + b * c % 5 - d * 6

Operators

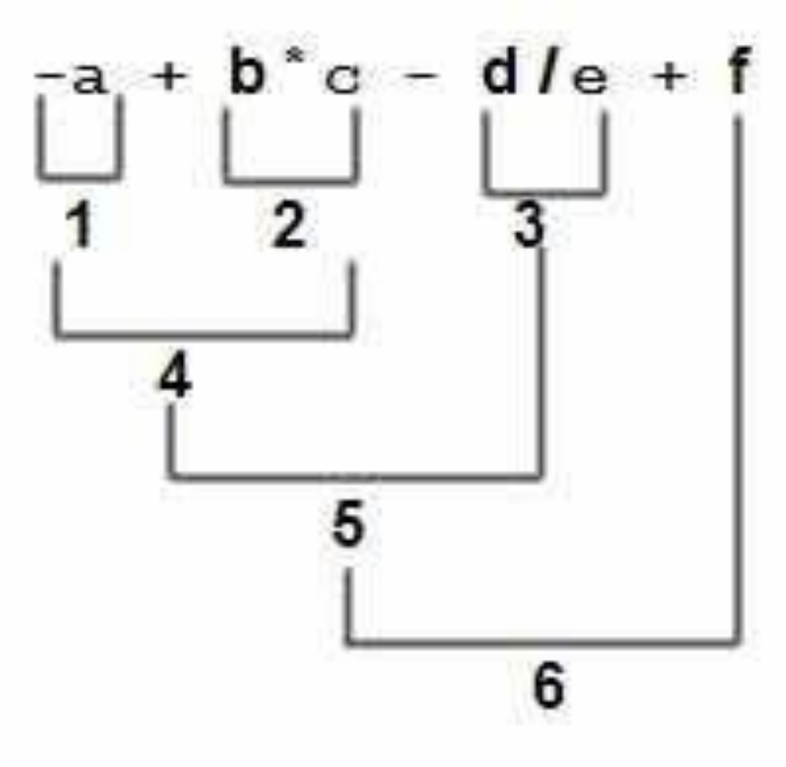
* / %

+ -

= (left to right)

a= 1, b=5, c=1, d=1

HEIRARCHY/PRIORITY/PRECEDENCE



```
int i=10,j=20,k;

k = i++ * j++;
printf("i++ * j++ = %d \n",k);

i=10;
j=20;
k = ++i * ++j;
printf("++i * ++j = %d",k);
```

OUTPUT:

```
        i++ * j++ = 200
        ++i * ++j = 231
```

Bitwise Operators in C (&, |, ^,~, >>, <<)

```
unsigned int a = 12, b = 25;

printf("a=%d, b=%d \n",a,b);

printf("a&b = %d \n",a&b);

printf("a|b=%d\n",a|b);

printf("a^b=%d\n",a^b);

printf("~a=%d \n",~a);

printf("b<<1=%d \n",b<<1);

printf("b>>1=%d",b>>1);
```

For bitwise & operator:

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

```
  00001100
& 00011001
 ________
 00001000  = 8 (In decimal)
```

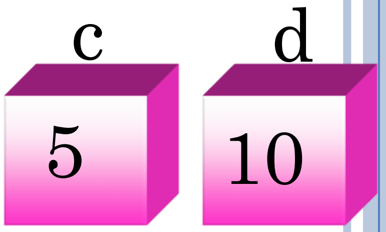
Calculate Power Using pow() Function

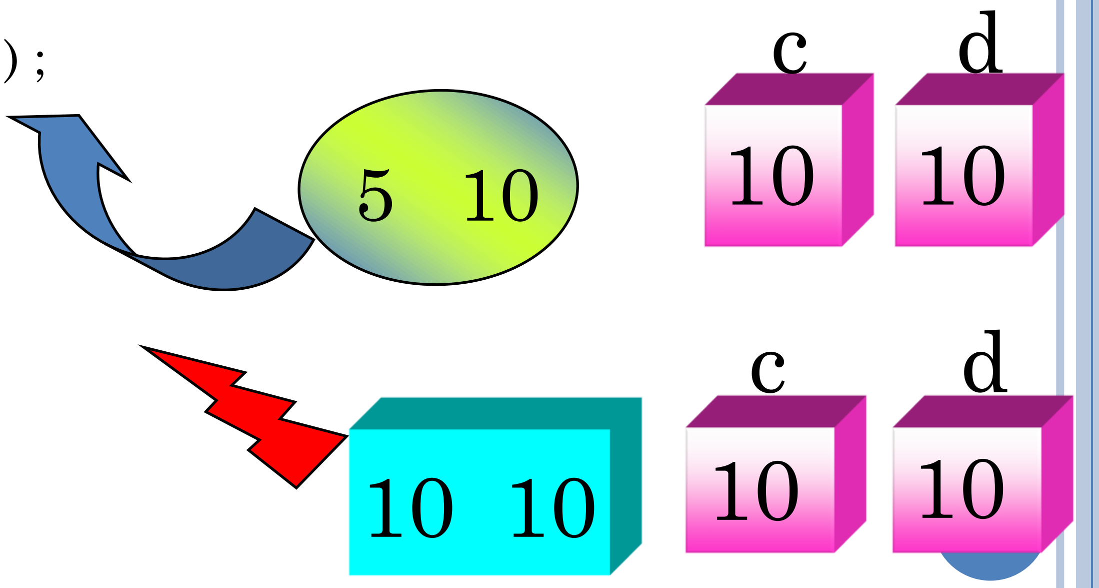
```
#include<stdio.h>
#include<math.h>
int main()
{
    int a,b,c;
    printf("Enter numbers");
    scanf("%d %d", &a,&b);
    c=pow(a,b);
    printf("Result = %d",c);
    return 0;
}
```

math.h -> abs(), sqrt(), sin(), cos(), tan(), etc

INTERCHANGING CONTENTS OF TWO VARIABLES

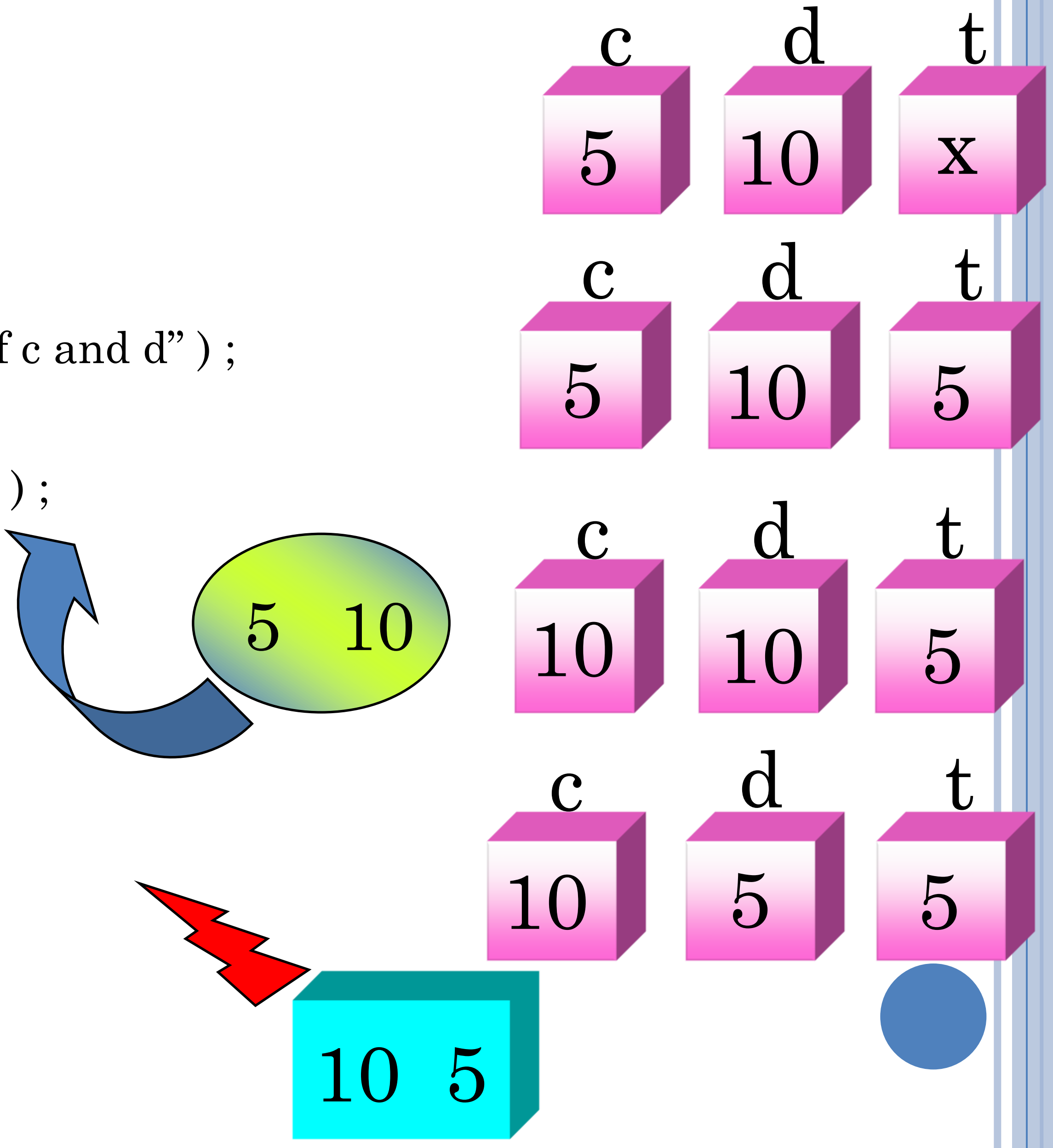
```
int main( )
{
    int c, d ;
    printf ( ”Enter values of c and d” ) ;

    scanf ( ”%d%d”, &c, &d ) ;

    c = d  ;

    d = c ;

    printf ( ”%d %d”, c, d ) ;
}
```





Interchanging Contents of Two Variables

```
main( )
{
    int c, d , t ;
    printf ( "Enter values of c and d" ) ;

    scanf ( "%d%d", &c, &d ) ;

    t = c ;

    c = d ;

    d = t ;

   printf ( "%d %d", c, d ) ;
}
```



WAP TO SWAP TWO NUMBERS WITHOUT USING THIRD VARIABLE

```
#include<stdio.h>
 int main()
{
          int a=10, b=20;
          printf("Before swap a=%d b=%d",a,b);
          a=a+b;          //a=30 (10+20)
          b=a-b;          //b=10 (30-20)
          a=a-b;          //a=20 (30-10)
          printf("\nAfter swap a=%d b=%d",a,b);
return 0;
}
```

TYPEDEF

- Allows user to define identifier that would represent an existing data type.

- e.g.

```
 typedef   unsigned  int   type1;

 type1 i,j,k;
```

ENUMERATED DATA TYPE

- **Enumeration** (or **enum**) is a user defined **data type in C**.

- It is mainly used to assign names to integral **constants**, the names make a program easy to read and maintain.

- **enum** State {Working = 1, Failed = 0};

```
#include<stdio.h>
enum week{Monday, Tuesday, Wednesday, Thursday, Friday, Saturday,
Sunday};
int main( ) {
int i;
scanf("%d",&i);
switch(i)
{
case Monday:
case Tuesday:
case Wednesday:
case Thursday:
case Friday:
                printf("Weekdays");
                 break;
case Saturday:
case Sunday:
                printf("Weekend");
}
return 0;
}
```

Enumerated data type: Example

```
include<stdio.h>
enum week{Mon=10, Tue, Wed, Thur, Fri=10, Sat=16, Sun};
enum day{Mond, Tues, Wedn, Thurs, Frid=18, Satu=11, Sund};

int main()
{
printf("The value of enum week: %d %d %d %d %d %d %d",Mon,
Tue, Wed, Thur, Fri, Sat, Sun);


printf("The default value of enum day: %d %d %d %d %d %d
%d",Mond , Tues, Wedn, Thurs, Frid, Satu, Sund);
}
```

OUTPUT:

```
The value of enum week: 10 11 12 13 10 16 17
The default value of enum day: 0 1 2 3 18 11 12
```