Introduction to C-Programming

Er. Shiva K. Shrestha (HoD)

Department of Computer Engineering,

Khwopa College of Engineering

Why C?

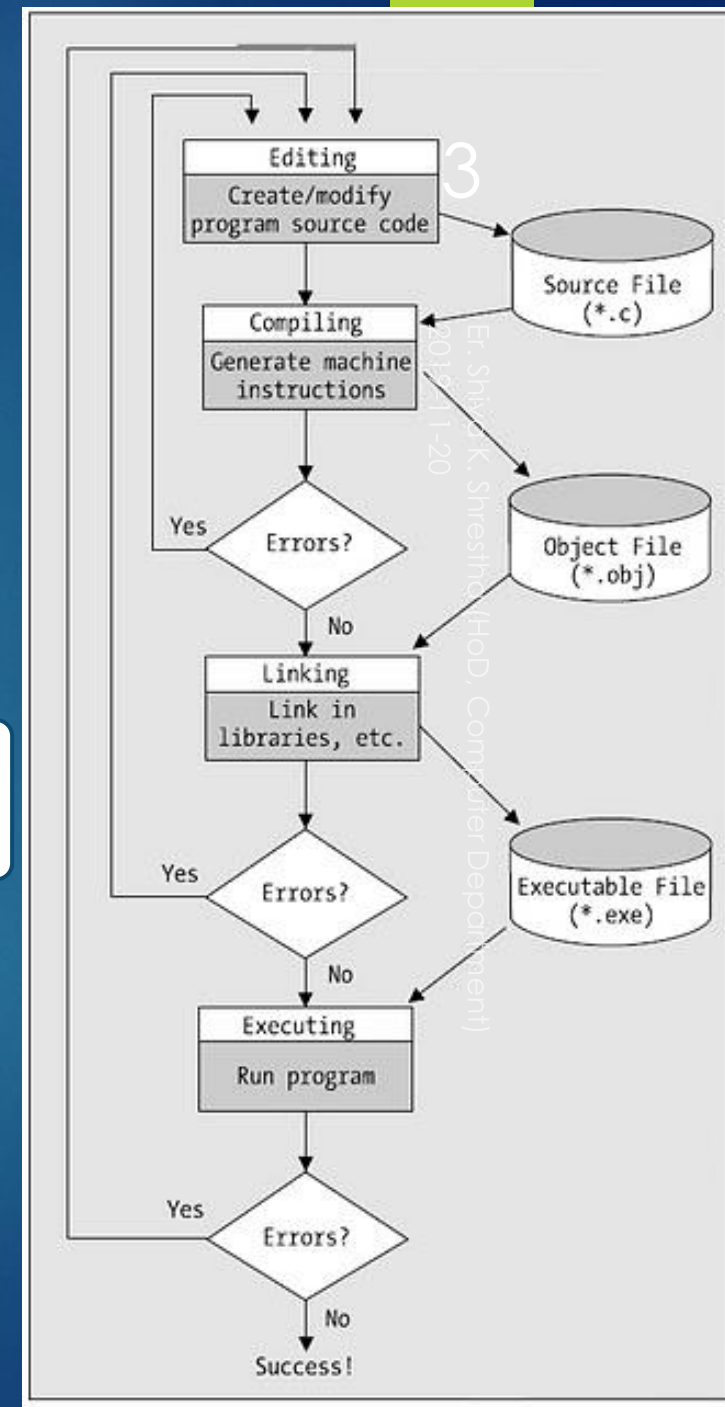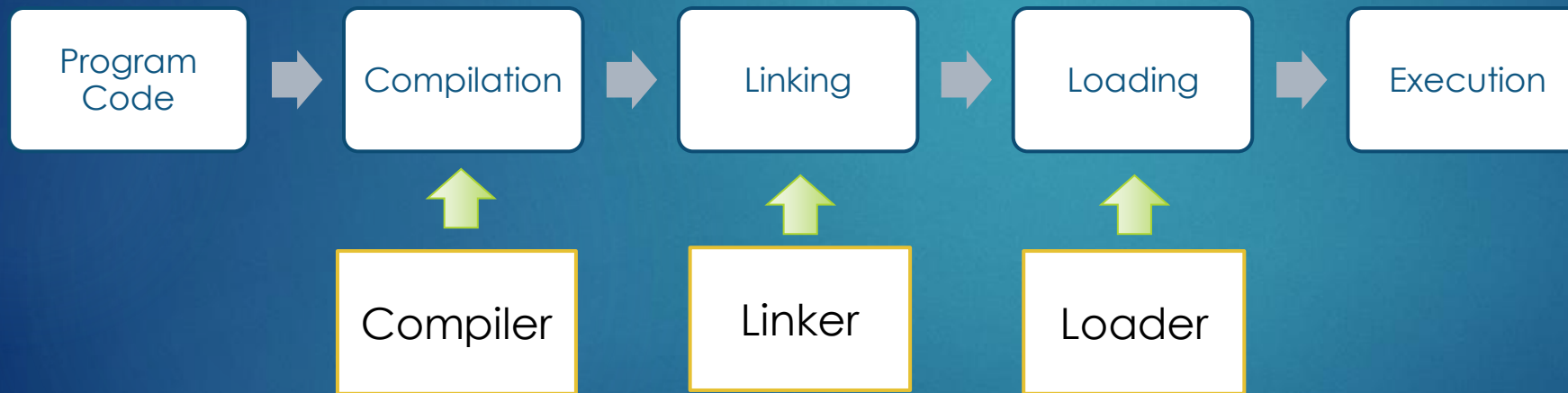## C Features

- Simple
- Portability
- Syntax Based
- Case Sensitive
- Structure Oriented
- Middle Level
- Use of Pointer
- Compiler Based
- Fast & Efficient
- Powerful

Uses of C

- OS
- Compilers
- Device Drivers
- Interpreters
- Embedded System

Executing a C Program

Compilation & Execution

Basic Structure of C Program

1. Documentation Section
2. Link Section
3. Definition Section
4. Global Declaration Section
5. main() Function Section
6. Subprogram Section

Documentation Section
Link Section
Definition Section
Global Declaration Section

main() Function Section
{
    Declaration Part
    Executable Part
}

Subprogram Section
    Function 1
    Function 2
    Function 3
    -
    -
    -
    Function n

Sample C Program

```
// Name of Program                    Documentation section

#include<stdio.h>
#include<conio.h>                      Link section

#define max 10                         Definition section
    void add();                        Global declaration section
    int x=10;

    int main()                         main () Function section / Entry Point
    {   int a=10;                       variable declaration
    printf("Hello Main");
        return 0;                       Body of Main function

    }

void add(){
    printf("Hello add");                Function Declaration
        }
```

Class Works

- WAP to display message "Plane Hijacked".
- WAP to calculate sum of three entered numbers with 7.
- WAP to display pass or fail as per OM entered by user.

Character Set

Categories

▶ Letters or Alphabets

    ▶ Uppercase Letter (A to Z), Lowercase Letter (a to z)

▶ Digits (0 to 9)

▶ Special Characters

▶ White Spaces

The **special characters** are listed below:

+	-	*	/	=	%	&	#	!	?	^
"	'	~	\	|	<	>	(	)	[	]
{	}	:	;	.	,	_	(blank space)			

Tokens/Identifiers

- Identifiers
- Variables
- Keywords
- Constants
- String
- Operators
- Special Symbols

Identifier is the word used for name given to variables, functions, constants etc. It can also be defined as a set of combinations of one or more letters or digits used to identify constants, variables, functions etc.

Rules for Valid Identifiers

1. First character should always be letter.
2. Both uppercase and lowercase letters may be used and are recognized as distinct.
3. Only underscore (_) is allowed among special symbols and is treated as letter.
4. Space and other symbols are not valid.
5. Proper naming conventions should be followed.
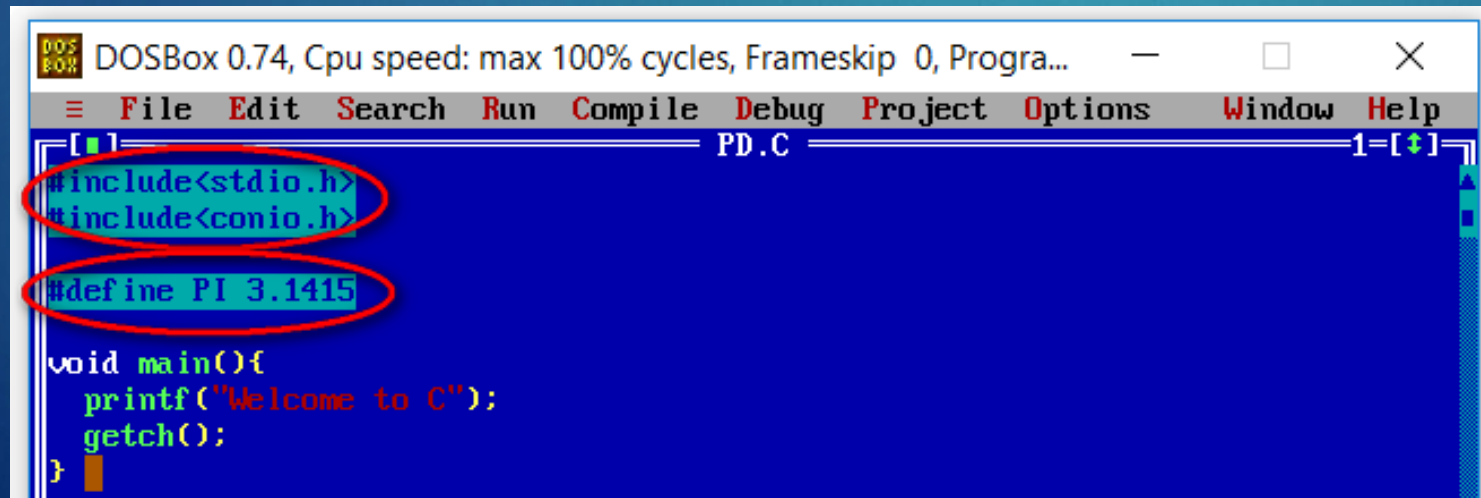6. Identifier's should not any keyword

32 keywords available in C

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

Valid	Count
	test23
	high_balance
	_test
Invalid	4th
	order-no
	error flag
	hi!there

Preprocessor Directives

Types of Preprocessor Directives (Mainly)

- ▶ File Inclusion
  - ▶ #include

- ▶ Macro Substitution
  - ▶ #define

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Progra...   —   □   ×
≡   File   Edit   Search   Run   Compile   Debug   Project   Options   Window   Help
[■]═══════════════════════════ PD.C ═══════════════════════════1═[↕]
#include<stdio.h>
#include<conio.h>

#define PI 3.1415

void main(){
    printf("Welcome to C");
    getch();
}
```

Preprocessor Directives

| File Inclusion Directive | #include |
| Macro Substitution Directive | #define |

Conditional Directive
- #if
- #elif
- #else
- #endif
- #ifdef
- #ifndef
- #under

Miscellaneous Directives
- #pragma
- #error
- #line
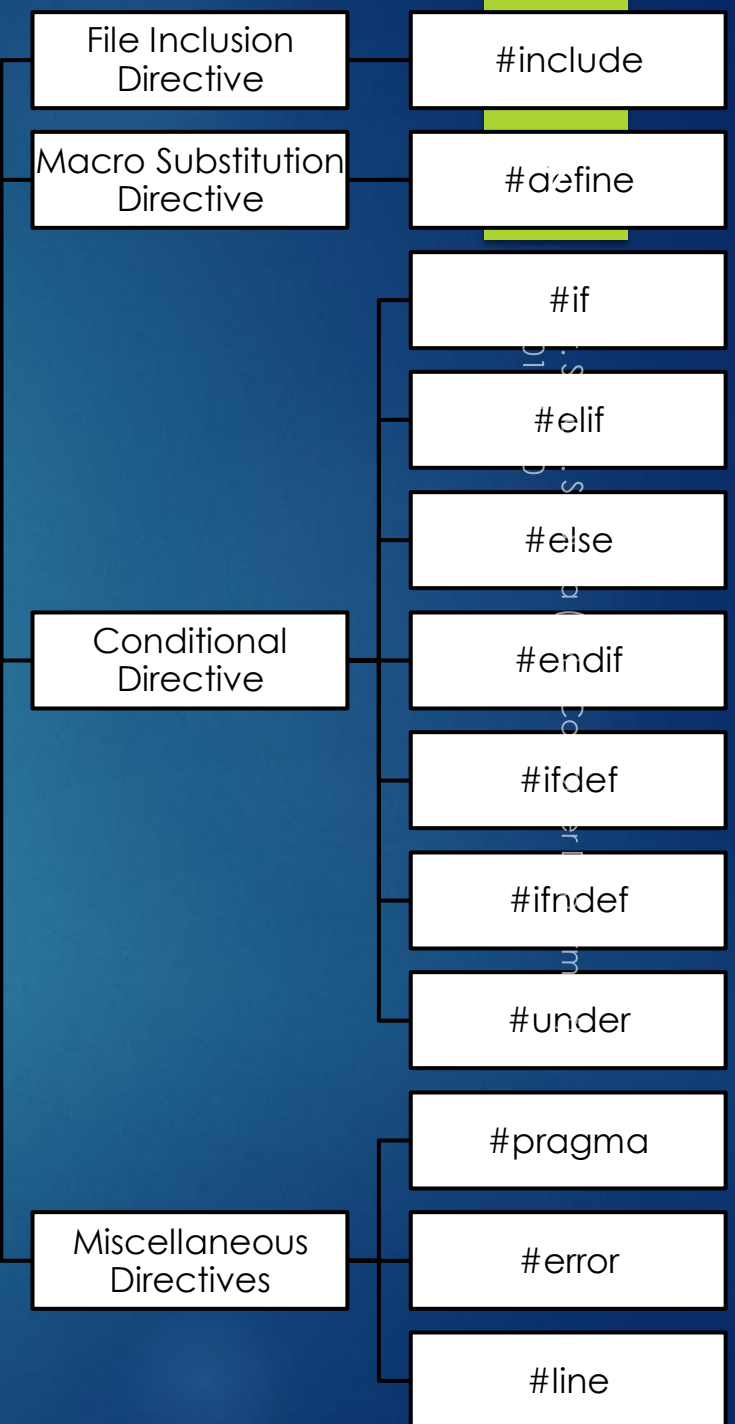
Variables

▶ A variable is a named location in memory that is used to hold a value that may be modified by the program.

General syntax:

  typename var1, var2;

▶ e.g. int a, b;

  float c,d;

  char e;

Scope of Variables

(Where Variables are declared?)

▶ Variables will be declared in three basic places: *inside functions, in the definition of function parameters*, and *outside of all functions*.

▶ These are local variables, global variables, and formal parameters.

```
int x=10;          // Global x
voi main()
{
    int x=20;          // X Local to Block 1
    - - - -
    - - - -
        {
            int x=30;  // X Local to Block 2
            - - - -
            - - - -
            - - - -
        }
}

void funct()
{
    int x=40;          // X Local to Block 3
    - - - - -
    - - - - -
    - - - - -
}
```

① ② ③

Local Variables

► Variables that are declared inside a function are called local variables.

► For Example

```
void func1( ){
    int x;
    x=10;
}
void func2( ){
    int x;
    x=-199;
}
```

```c
/* Local Variable: Example#1 */
#include<stdio.h>
#include<conio.h>
int main(){
    int outerVariable = 10;
    clrscr();

    {
        printf("I can access outerVariable in inner block = %d\n", outerVariable);
    }

    printf("I can also access outerVariable in same block = %d", outerVariable);

    getch();
}
```
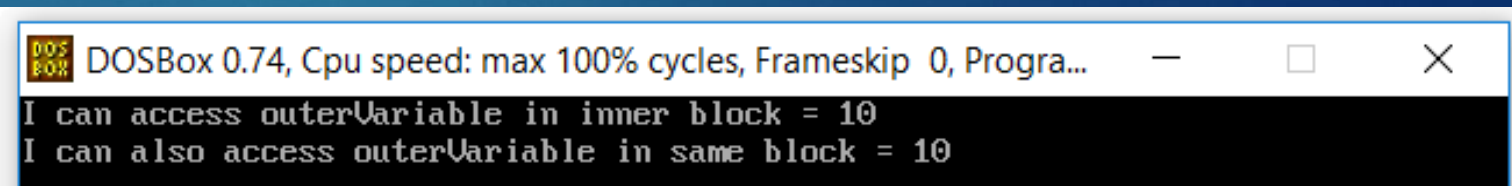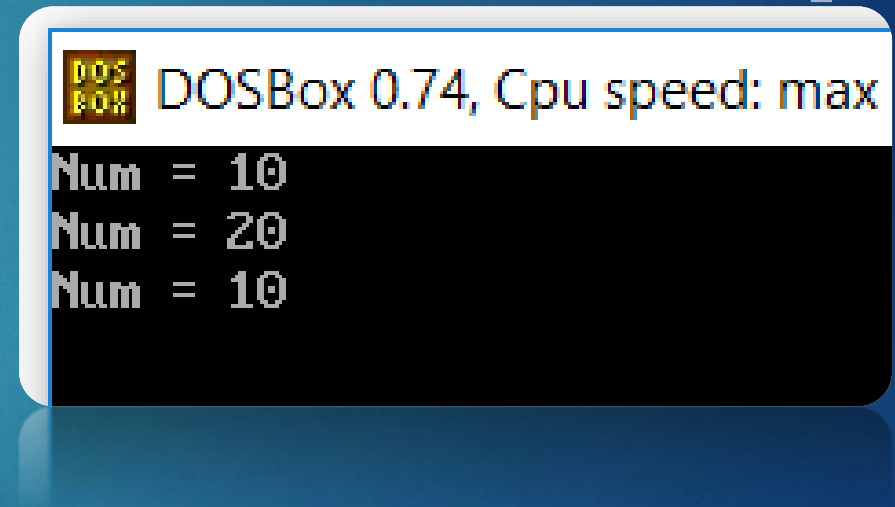
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...   —   □   ✕

```
I can access outerVariable in inner block = 10
I can also access outerVariable in same block = 10
```

Local Variable (2)
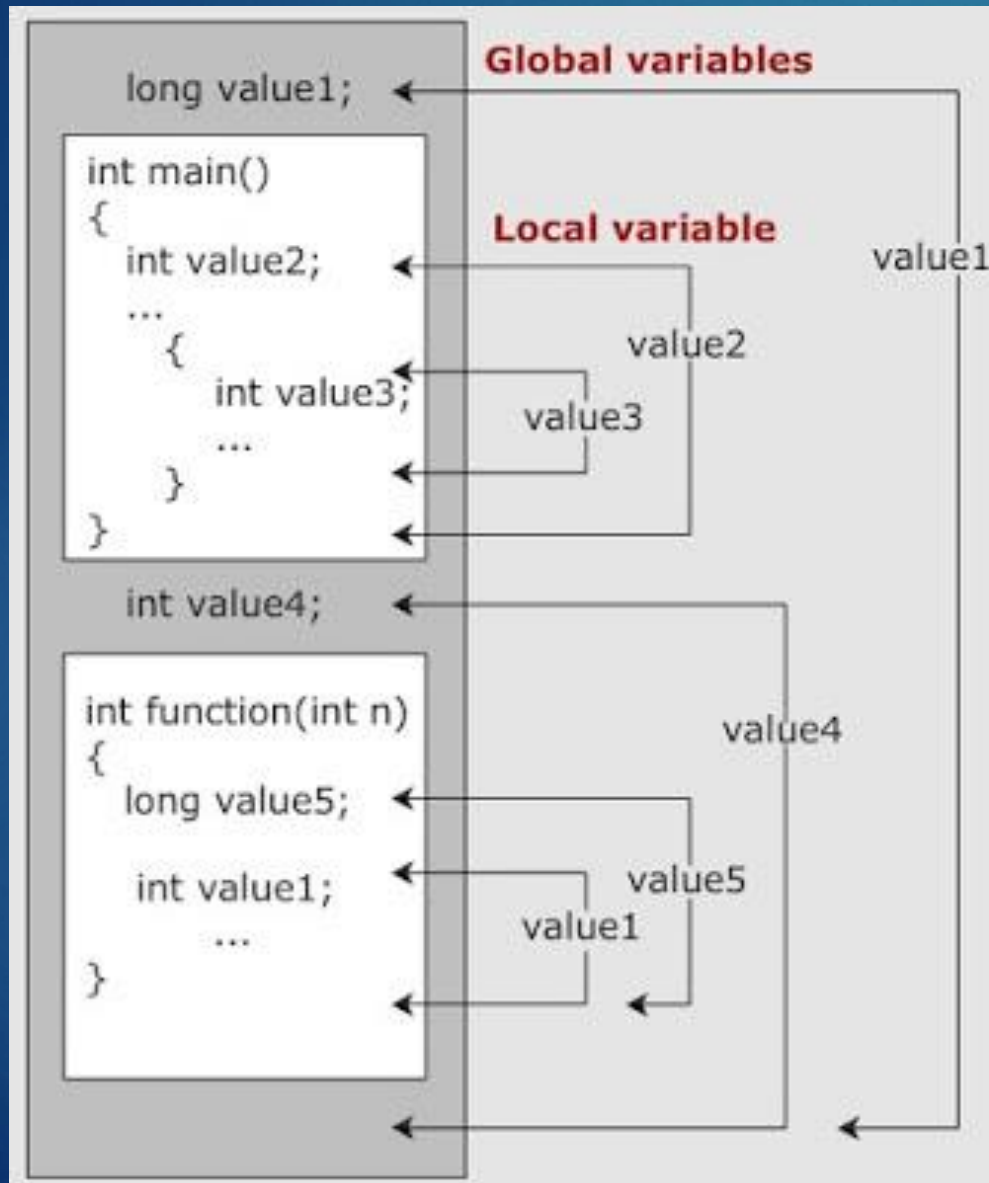
```c
#include<stdio.h>
void main(){
    int num = 10;
    // I cannot declare num again in same block
    // int num;
    printf("Num = %d\n", num);
    {
        // I can declare num again in inner block
        int num = 20;
        printf("Num = %d\n", num);
    }
    printf("Num = %d\n", num);
}
```

DOSBox 0.74, Cpu speed: max
Num = 10
Num = 20
Num = 10

Global Variables


```c
1   /* Global Variable: Example#1 */
2   #include<stdio.h>
3   #include<conio.h>
4   int count; /* count is global */
5   void func1( );
6   void func2( );
7   void main( ){
8       count=100;
9       func1( );
10      getch( );
11  }
12  void func1( ){
13      int temp;
14      temp=count;
15      func2( );
16      printf("\ncount = %d",count);
17  }
18  void func2( ){
19      int count=50;
20      printf("Value of count:%d",count);
21  }
```

```
Value of count:50
count = 100
```

Formal Parameters

If a function is to use arguments, it must declare variables that will accept the values of the arguments.


```
/* Print two numbers in order. */

void Sort ( int a, int b )
{
  int temp;

  if ( a > b )
  {
    printf("%d %d\n", b, a);
  }
  else
  {
    printf("%d %d\n", a, b);
  }
}
```

Formal parameters

Actual parameters

```
int main()
{
  int x = 3, y = 5;

  Sort ( 10, 9 );
  Sort ( y, x + 4 );
  return 0;
}
```

11
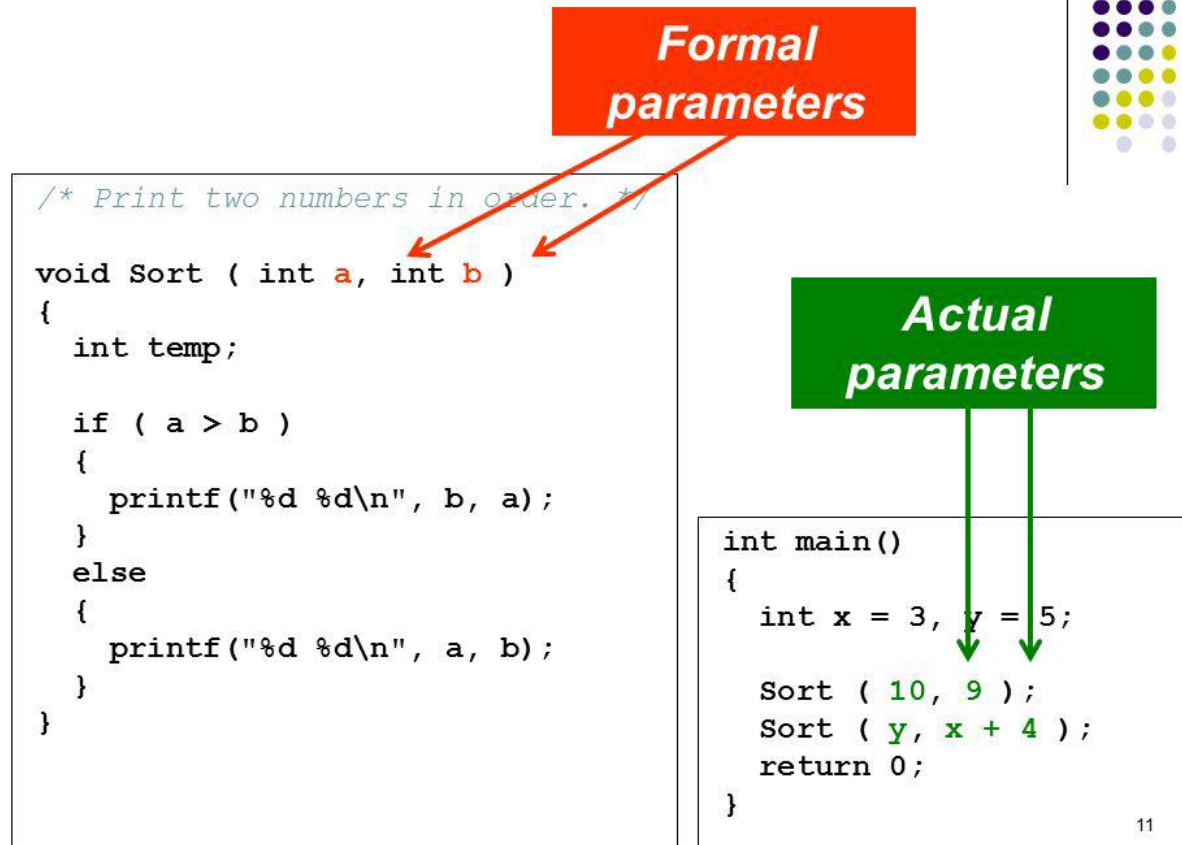
```
1   /* Formal Parameter */
2   #include<stdio.h>
3   #include<conio.h>
4   int square(int x);
5
6   void main(){
7     int t=5;
8     t=square(t);
9     printf("Value of t:%d",t);
10    getch();
11   }
12
13   int square(int x){
14     x=x*x;
15     return x;
16   }
17
```
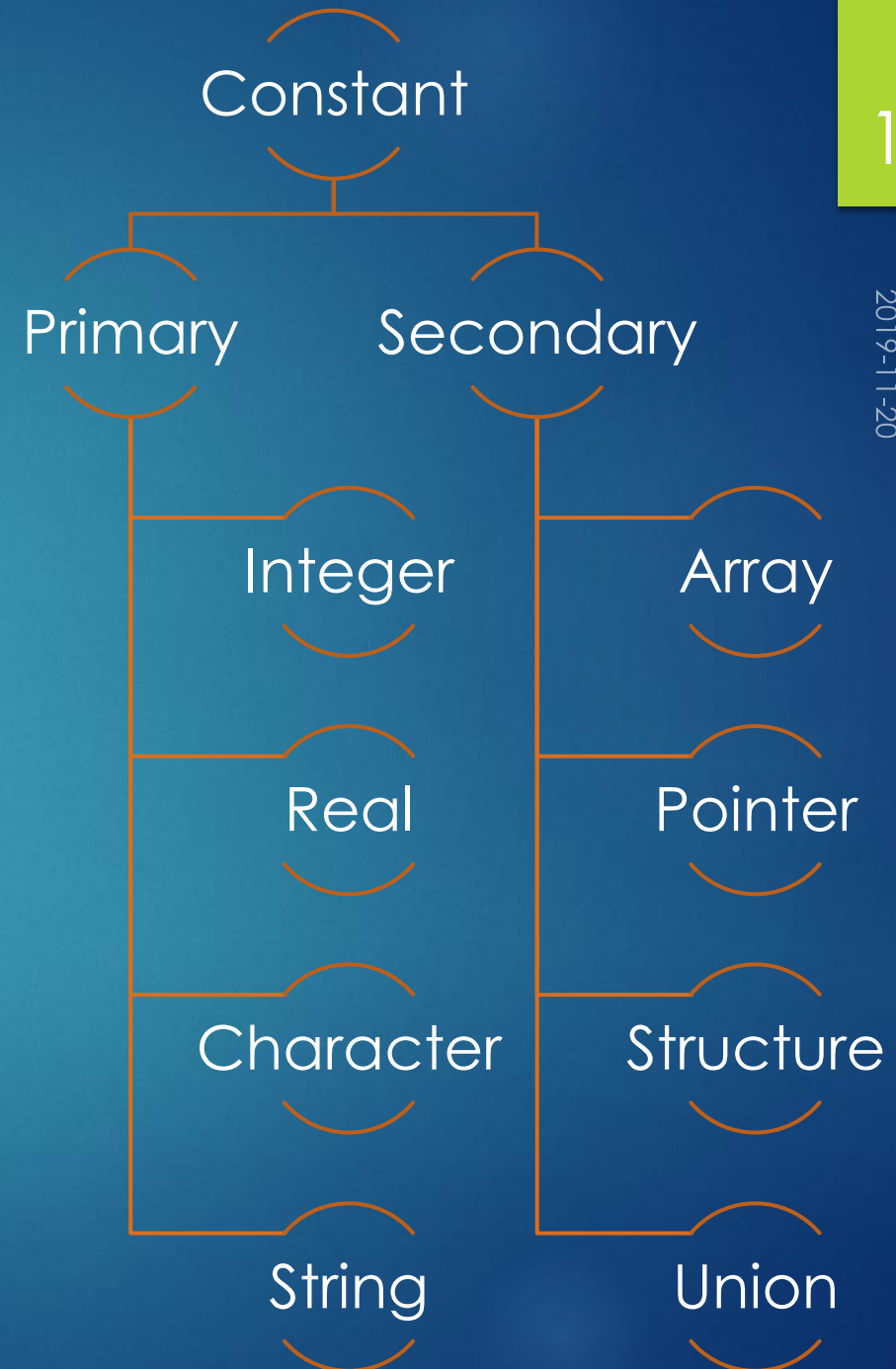
Value of t:25

Constants

- Constant – it does not change during the execution of a program
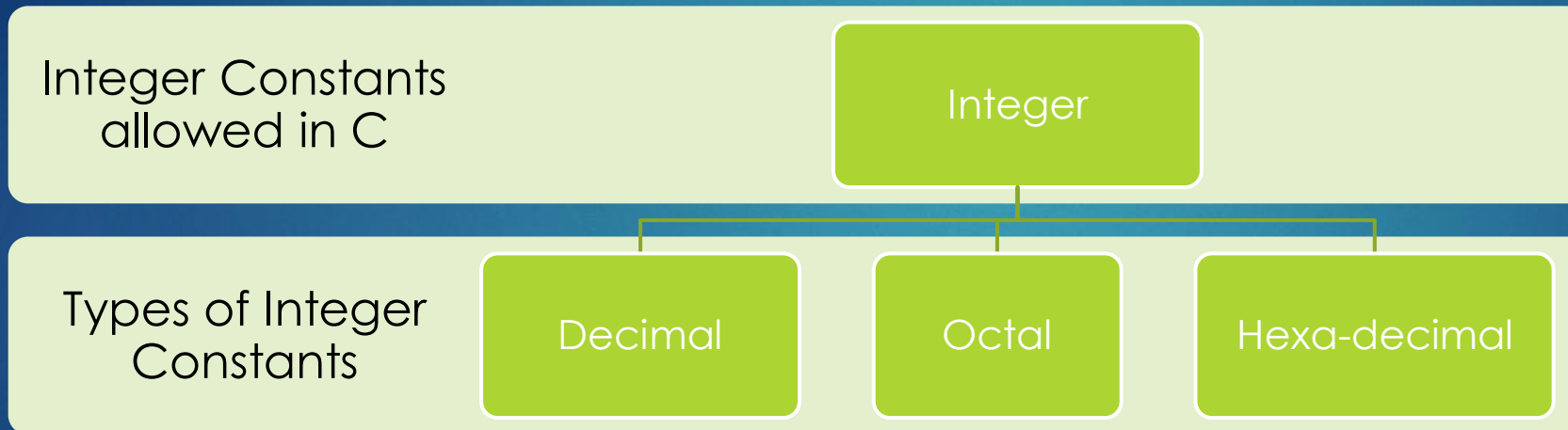
- For Example

  #define PI 3.1415

  const float pi=3.14159;

Constant
- Primary
  - Integer
  - Real
  - Character
  - String
- Secondary
  - Array
  - Pointer
  - Structure
  - Union

Primary Constants
Integer Constants

Integer constants are whole numbers without any fractional parts.

Integer Constants allowed in C	Integer		
Types of Integer Constants	Decimal	Octal	Hexa-decimal

Valid	1234
	3456
	-9746
Invalid	11.
	45,35
	$12
	X248

Primary Constants
Integer Constants: Decimal Integer Constant

Decimal integer constants:

▶ The allowable digits in a decimal integer constant are 0,1,2,3,4,5,6,7,8,9.

▶ The first digit should not be 0.

▶ Should at least one digit.

Valid	1234
	3456
	-9746
Invalid	11.
	45,35
	$12
	025
	x248

Primary Constants
Integer Constants: Octal Integer Constant

Octal integer constants:

▶ The allowable digits in a octal constant are 0,1,2,3,4,5,6,7

▶ Must have at least one digit and start with digit 0.

Valid	0245
	-0476
	+04013
Invalid	25 (does not start with 0)
	0387(8 is not an octal digit)
	04.32(Decimal point is not allowed)

Primary Constants
Integer Constants: Hexadecimal Integer Constant

Er. Shiva K. Shrestha (HoD, Computer Department)
2019-11-20

Hexadecimal integer constants:

▶ The allowable digits in a hexadecimal constant are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

▶ Must have at least one digit and start with 0x or 0X.

Valid	0x14AF
	0X34680
	-0x2673E
Invalid	0345 (Must start with 0x)
	0x45H3 (H is not a hexadecimal digit)

Primary Constants

Floating Point/Real Constants

Rules for constructing real constants:

▶ At least one digit to the left and right of the decimal point.

▶ It must have a decimal point.

▶ It could be either positive or negative

▶ If no sign precedes an floating point constant is assumed to be positive.

▶ No commas or blank spaces are allowed.

▶ For too large or small, the mantissa may have upto 7 digits & exponent may be between -38 and +38.

Valid	+325.34
	426.0
	32E-29
	35e+36
Invalid	1 (decimal point missing)
	1. (No digit following decimal digit)
	-1/2 (Symbol / is illegal)
	.5 (No digit to the left of the decimal)
	56,8.94 (Comma not allowed)
	-125.9e5.5 (exponent cannot be fraction)
	0.158e+954 (exponent too large)

Er. Sh
2019

Character Constants

▶ A character constant is a single alphabet, single digit, or a single special symbol enclosed within single quotes. Remember golden rule: single character single quote.

▶ e.g. 'A'   '5'   '+'   '?'

String Constants

▶ A string constant is a sequence of alphanumeric characters enclosed in double quotation marks whose maximum length is 255 characters. For e.g. const char txt [ ] ="This is C and C++ Programming language.";

▶ e.g. "green", "My name &&&!!!   ???", "$2525"

Declaration of Constants

▶ Value of an identifier defined as constant, cannot be altered during program execution.

▶ Generally, the constant is associated with an identifier and that name is used instead of actual number.

General Syntax

 const typename identifier = const;

 e.g.   **const float pi=3.14159;**

▶ **Another way to define constants** is with the #define preprocessor which has the advantage that it does not use ant storage (but who count bytes these days?).

▶ **#define Identifier Value**

▶ **#define PI 3.1415**

Escape Sequences

▶ backslash symbol (\) is considered an "escape character": it causes an escape from the normal interpretation of string, so that the character is recognized as one having a special meaning.

Escape Sequence	Purpose
\n	New Line (Line feed)
\b	Backspace
\t	Horizontal Tab
\r	Carriage Return
\f	Form feed
\'	Single quote
\"	Double quote
\\	Backslash
\a	Alert
\?	Question mark

Data Types

1. Primary Data Types
   - int
   - char
   - float
   - double
2. Derived Data Types
   - structures,
   - unions,
   - enumerations
3. User-defined Data Types

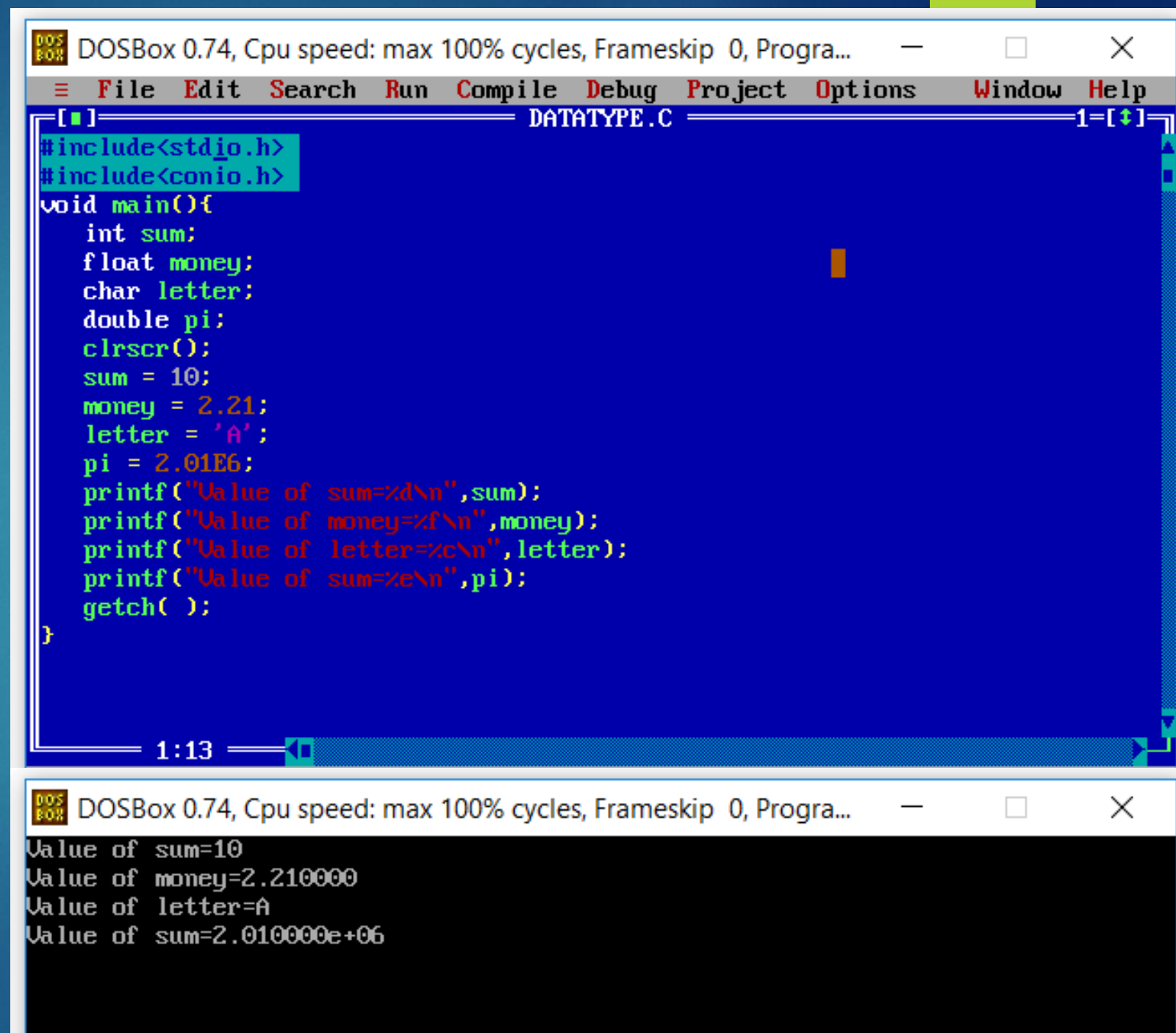# Primary Data Types

- **int**
  - int a;
  - a = 5;

- **float**
  - float miles;
  - miles=5.6;

- **double**
  - double big;
  - big = 312E7;

- **char**
  - char letter;
  - letter = 'y';

Modifying the Basic Data Types

The lists of modifiers are:

- signed
- unsigned
- short
- long

▶ signed, unsigned, short, long can be used with int.

▶ signed, unsigned can be used with char.

▶ long can be used with double.

The amount of storage allocated is not fixed. ANSI has the following rules

short int <= int <= long int

float <= double <= long double

Data Type	Category	Range	Memory Space	Format Specifier
int	short signed	- 32,768 to 32,767	2 Byte	%d or %i
	short unsigned	0 to 65535	2 Byte	%u
	signed	- 32,768 to 32,767	2 Byte	%d
	unsigned	0 to 65535	2 Byte	%u
	long signed	-2,147,483,648 to 2,147,483,647	4 Byte	%ld
	long unsigned	0 to 4,294,967,295	4 Byte	%lu
	hexadecimal			%x
	unsigned octal			%u
float	float	$1.2*10^{-38}$ to $3.4*10^{+38}$	4 Byte	%f
double	double	$1.7*10^{-308}$ to $3.4*10^{+308}$	8 Byte	%lf
	long double	$3.4*10^{-4932}$ to $1.1*10^{+4932}$	10 Byte	%Lf
char	signed char	- 128 to 127	1 Byte	%c
	unsigned char	0 to 255	1 Byte	%c

User-Defined Data Types

Syntax:

typedef existing_data_type new_name_for_existing_data_type;

Note:- We'll study example of typedef & Derived Data Type in Chapter 8.

Operators

1. Unary operators
   - -, ++, --, sizeof, (type)
   - e.g. -743, -a, -root, a++, a--, ++a, --a
2. Binary operators
   - Arithmetic - +, -, *, /, %
   - Assignment - =, +=, -=, *=, /=, %=
   - Relational - <, >, <=, >=, ==, !=
   - Logical - &&, ||
3. Tertiary operators
   - (?:)

Unary Operator - sizeof

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0,

```
≡  File   Edit   Search   Run   Compile   Debug   Proje
═[■]══════════════════════════════ FINDSIZE.C ═══════
#include<stdio.h>
#include<conio.h>
void main(){
    int sum;
    float money;
    char letter;
    double sn;
    clrscr();
    printf("Size of sum=%d\n",sizeof(sum));
    printf("Size of money=%d\n",sizeof(money));
    printf("Size of letter=%d\n",sizeof(letter));
    printf("Size of sn=%d\n",sizeof(sn));
    getch( );
}
```

DOSBox 0.74, Cpu speed: max 100%

```
Size of sum=2
Size of money=4
Size of letter=1
Size of sn=8
```

1. Unary Operator – Typecast (type)

Typecast operator is also unary operator. It is used to change the data type of a variable.

Syntax: **(type) expression**

▶ e.g. (float) x/2;

main(){

    int var1; float var2;

    var2= (float) var1;

}

▶ Unary operators have higher precedence. Also, the associativity of the unary operator is right to left.

Er. Shiva K. Shrestha (HoD, Computer Department)
2019-11-20

Binary Operators
Arithmetic Operators

+   for addition

-       for subtraction

*   for multiplication

/   for division

%   for remainder operation, modulo operator

Arithmetic Expressions

```
main(){
    int a=10,b=3; int c,d,e,f;
    c=a+b;
    d=a-b;
    e=a*b;
    f=a%b;
    printf("Value of c=%d\n",c);
    printf("Value of d=%d\n",d);
    printf("Value of e=%d\n",e);
    printf("Value of f=%d\n",f);
}
```

Binary Operators
Assignment Operators

=   for assignment

+=   add with destination and assign

-=   deduct from destination and assign

*=   multiply with destination and assign

/=   divide the destination and assign

%=   assign the remainder after dividing the destination

 exp1+=exp2   is equivalent to exp1=exp1+exp2

 exp1-=exp2   is equivalent to exp1=exp1-exp2

 exp1*=exp2   is equivalent to exp1=exp1*exp2

 exp1/=exp2   is equivalent to exp1=exp1/exp2

 exp1%=exp2   is equivalent to exp1=exp1%exp2

Binary Operators
Relational Operators

<	less than

>	greater than

<=	less than or equals to

>=	greater than or equals to

Equality operators are closely associated with this group

= =	equal to

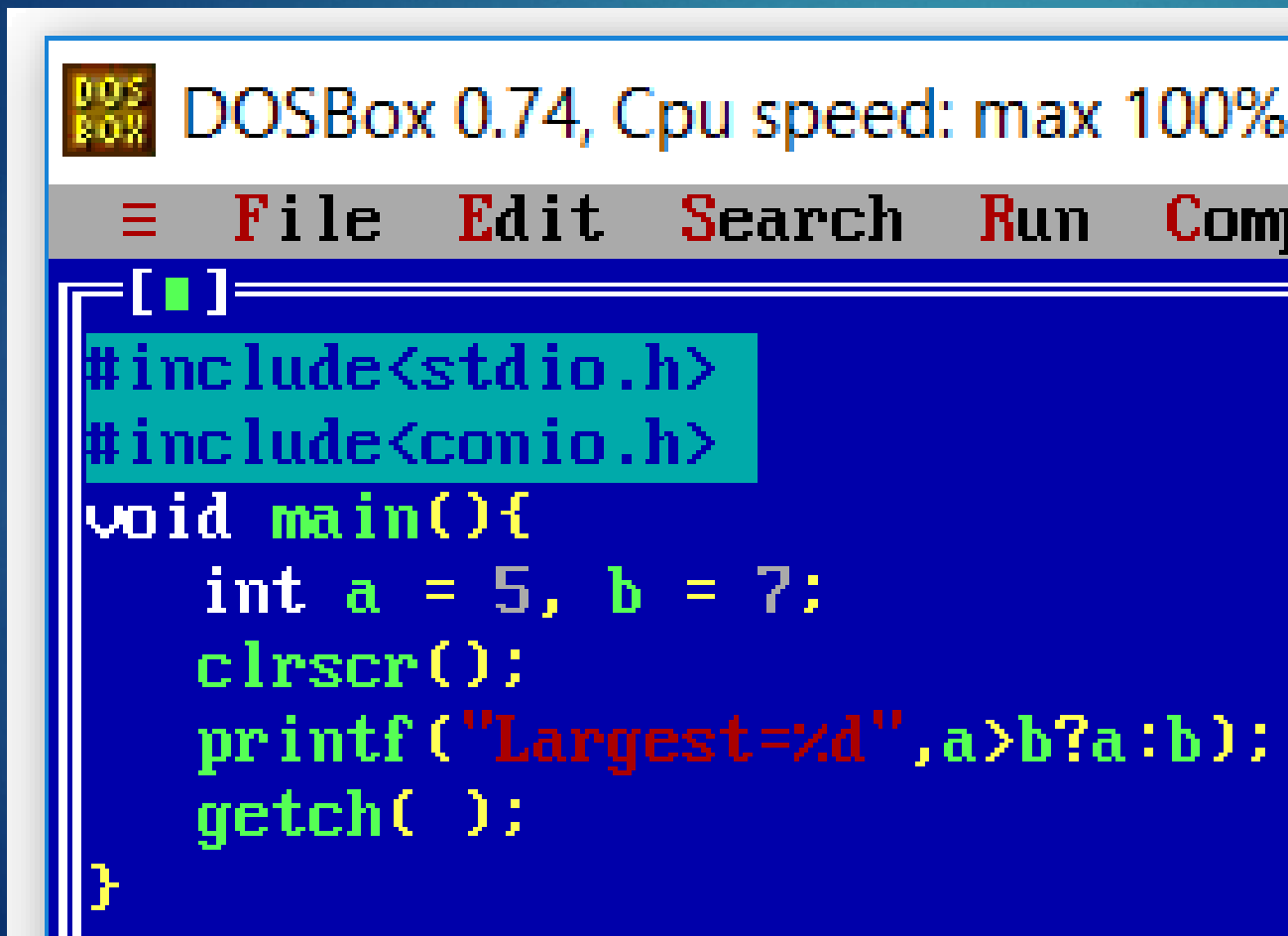!=	not equal to

Binary Operators
Logical Operators

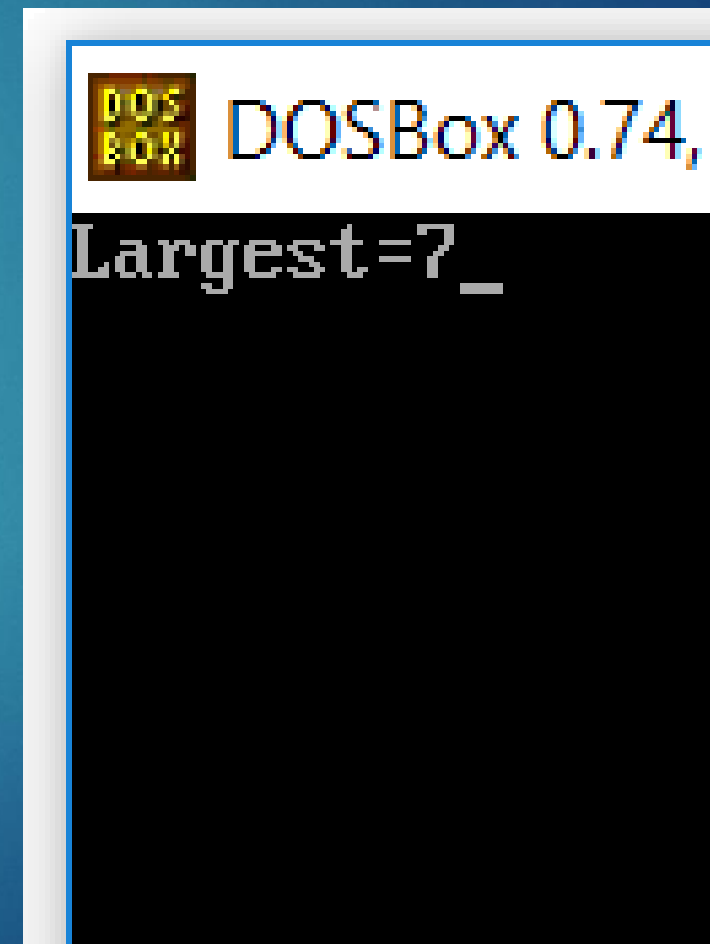&&	logical and (AND)

||	logical or (OR)

Tertiary Operator

▶ *(Test expression) ? true_value : false-value;*

E.g. (j<0)?0:100

```c
#include<stdio.h>
#include<conio.h>
void main(){
    int a = 5, b = 7;
    clrscr();
    printf("Largest=%d",a>b?a:b);
    getch( );
}
```

```
Largest=7_
```

Bitwise Operators

Bitwise Operator

- Bitwise Logical Operators
- Bitwise Shift Operators
- One's Complement Operator

a = 15 = 0000 1111
b =   7 = 0000 0111

a = 15 = 0000 1111
b =   7 = 0000 0111
AND = 0000 0111

a = 15 = 0000 1111
b =   7 = 0000 0111
OR   = 0000 1111

a = 15 = 0000 1111
b =   7 = 0000 0111
XOR = 0000 1000

Bitwise Logical

Operators

▶ Bitwise AND (&)

▶ Bitwise OR (|)

▶ Bitwise XOR (^)

```c
#include<stdio.h>
#include<conio.h>
void main(){
    int a = 15, b = 7, AND, OR, XOR;
    clrscr();
    AND = a & b;
    OR = a | b;
    XOR = a ^ b;
    printf("AND => %d\n", AND);
    printf("OR  => %d\n", OR);
    printf("XOR => %d\n", XOR);
    getch();
}
```

DOSBox 0.74, Cpu speed: max 100% cycle

```
AND => 7
OR  => 15
XOR => 8
```

Bitwise Operators
Bitwise Shift Operators

▶ Left Shift (<<)

▶ Right Shift (>>)

a	0000 1111
Left Shift by 3	0111 1000
Right Shift by 3	0000 0001

```c
#include<stdio.h>
#include<conio.h>
void main(){
    int a = 15, left, right;
    clrscr();
    left = a << 3;
    right = a >> 3;
    printf("Left Shifted by 3 Bits  => %d\n", left);
    printf("Right Shifted by 3 Bits => %d\n", right);
    getch();
}
```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Progra...

Left Shifted by 3 Bits  => 120
Right Shifted by 3 Bits => 1
```

Bitwise Operators

One's Complement Operator

```c
1  #include<stdio.h>
2  #include<conio.h>
3  void main(){
4      int a = 12, ones_c;
5      clrscr();
6      ones_c = ~a;
7      printf("One's Complement of %d is %d", a, ones_c);
8      getch();
9  }
```

-1 = 65,535
-2 = 65,534
…
1 = 1
0 = 0

DOSBox 0.74, Cpu speed: max 100% cycles, Fram
One's Complement of 12 is -13_

a = 12 = 0000 0000 0000 1100

~a   = 1111 1111 1111 0011=65,523

~a   = 1111 1111 1111 0011 = -32,755

Bitwise Operators - Two's Complement

```c
#include<stdio.h>
#include<conio.h>
void main(){
    int a = -13, ones_c, twos_c;
    clrscr();
    ones_c = ~a;
    twos_c = ones_c + 1;
    printf("Two's Complement of %d is %d", a, twos_c);
    getch();
}
```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...

Two's Complement of -13 is 13_

$a = -13 = 1111\ 0011$

$\sim a \qquad = 0000\ 1100$

$+1$

$13 = 0000\ 1101$

Pre-[increment | decrement]

Substitution

Evaluation

Assignment

Post-[increment | decrement]

```c
/* Modify Operators in C */
#include<stdio.h>
#include<conio.h>
void main(){
    int a=1,b=2,c=3; clrscr();
        printf("Initial Value...
    c=++a + ++b;
        printf("\n1.\ta=%d\t...
    c=a++ + ++a;
        printf("\n2.\ta=%d\tb=%d\tc=%d",++a,b++,++c);
        printf("\n3.\ta=%d\tb=%d\tc=%d",a++,b,c);
    a=++a + ++b + ++c;
        printf("\n4.\ta=%d\tb=%d\tc=%d",a,b,c);
    b=a++ + b++ + c++;
        printf("\n5.\ta=%d\tb=%d\tc=%d",a,b,c);
    c=--a + b-- + 3;
        printf("\n6.\ta=%d\tb=%d\tc=%d",a,b,c);
    b=2*--a + b++ - 10;
        printf("\n7.\ta=%d\tb=%d\tc=%d",a,b,c);
    a = b>c? 3 + c++ : 5 - b++;
        printf("\n8.\ta=%d\tb=%d\tc=%d",a,b,c);
    c = 8 + ++b + ++b + b;
        printf("\n9.\ta=%d\tb=%d\tc=%d",a,b,c); getch();
}
```

Terminal output:

```
Initial Values: a=1        b=2        c=3
1.        a=2        b=3        c=5
2.        a=5        b=3        c=7
3.        a=5        b=4        c=7
4.        a=20       b=5        c=8
5.        a=21       b=34       c=9
6.        a=20       b=33       c=57
7.        a=19       b=62       c=57
8.        a=60       b=62       c=58
9.        a=60       b=64       c=200
```

	a	b	c
Initial	1	2	3
	2	3	
			5=(2+3)
Step 1	2	3	5
	3		
			6=(3+3)
	4		
Step 2	5	3	7
		4	
Step 3	5	4	7
	6		
	7	5	8
20=(7+5+8)			
Step 4	20	5	8
	33=(20+5+8)		
	21	34	9
Step 5	21	34	9
	20		
			57=(20+34+3)
		33	
Step 6	20	33	57
	19		
	61=(2*19+33-10)		
		62	
Step 7	19	62	57
60=(3+57)			
			58
Step 8	60	62	58
		63	
		64	
			200=(8+64+64+64)
Step 9	60	64	200

Task
Write output of following program:

```c
#include<stdio.h>
#include<conio.h>
void main(){
    int a=7,b=8,c=9;
    clrscr();
    a=a++ + a++ + a++ + ++a;
    b=++b + b++ + ++b + b;
    c=--c + --b + --a;
    a=a + b + c--;
    b=a-- + b++ + --b;
    c=a + --b + --c;
    a=8 + b + c + --a;
    b=a-- - b-- - --c;
    printf("\n8.\ta=%d\tb=%d\tc=%d",a,b,c);
    getch();
}
```

DOSBox 0.74, Cpu speed: max 100%

```
8.      a=864     b=162     c=468_
```

Hierarchy of Operations

Precedence - operator is to be executed by PMDAS

Associativity is the preference of the operators (of same precedence level), when there is tie between them. e.g. **z=a+b*c**

Then, operation occurs as z=a+(b*c) as * higher precedence than + But when the case is like Z=a*b+c/d, operation occurs as

Z=(a*b)+(c/d) i.e. multiplication is done first. Then (c/d) is evaluated. Then only the two values are added. It happens so because * and / have higher precedence than +. However, as * and / have same precedence level, * is evaluated first, as the associativity of the arithmetic operators is from left to right, hence whichever operator occurs first, that is evaluated.

Arithmetic Expression: z=a+2*b/c*d+7+a*a; a = 1, b = 2, c = 3, d = 4; z =?=13

Operator Category	Operators	Associativity
Arithmetic Multiply, Divide, & Remainder	*, /, %	L->R
Arithmetic Add and Subtract	+, -	L->R
Relational Operators	<, >, <=, >=	L->R
Equality Operators	==, !=	L->R
Logical AND	&&	L->R
Logical OR	||	L->R
Conditional Operator	?:	R->L
Assignment Operator	=, +=, -=, *=, /=, %=	R->L
Unary Operators	-, ++, --, sizeof, (type)	R->L

Thank You!

Er. Shiva K. Shrestha

computer.khwopa@gmail.com