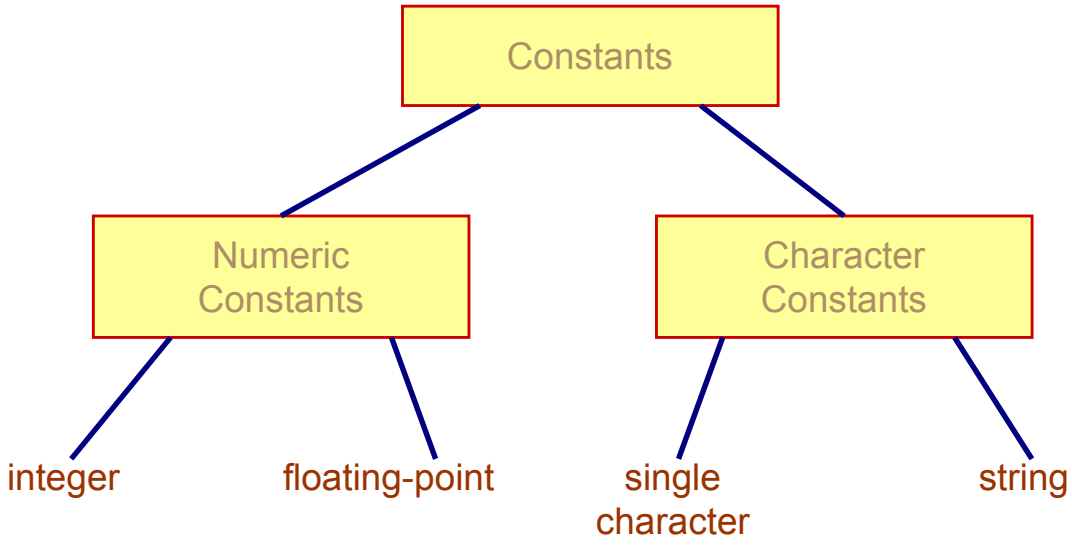
COMPUTER PROGRAMMING - I

The C Character Set

- The C language alphabet:
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

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

Constants



Numeric Constants

- **Rules for constructing Numeric constant**
  1) An integer constant must have at least one digit.
  3) It can either be positive or negative.
  4) No commas or blanks are allowed within an integer constant.
  5) If no sign precedes a numeric constant, it is assumed to be positive.
  6) **Integer constant** must not have decimal point
- Eg. 426, +782,-8000,-7605
- 7) **Real Constant** must have a decimal point
  E.g.: +867.9, -26.9876, 654.0

Real Constant

- **Rules for constructing Real constants (Exponential Form)**
  1) The mantissa part and the exponential part should be separated by the letter ‘e’
  2) The mantissa may have a positive or negative sign(default sign is positive)
  3) The exponent must have at least one digit
  4) The exponent must be a positive or negative integer(default sign is positive)
- 5) Exponent should not be fraction
  E.g.: +3.2e-4, 4.1e8, -0.2e+4, -3.2e-4

Character Constant

- **Character constant:**
- A character constant is an alphabet, a single digit or a single special symbol enclosed within inverted commas.
  E.g.: ’B’, ’I’, ’#’
- **String Constant**:
- Sequence of characters enclosed in double quotes.
  - The characters may be letters, numbers, special characters and blank spaces.
- Examples:

  “nice”, “Good Morning”, “3+6”, “3”, “C”

What Are Variables in C?

- **Variables** in C have the same meaning as variables in algebra. That is, they represent some unknown, or variable, value.

$$x = a + b$$

Naming Variables

- Variable names or identifiers in C
  - May only consist of letters, digits, and underscores
  - May be as long as you like, but mostly the first 31 characters are significant
  - May not begin with a number
  - May not be a C **reserved word (keyword)**

Keywords

- A keyword is a **reserved word**. You cannot use it as a variable name, constant name etc.
- There are 32 keywords in C language as given below:

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

Keywords of C

- Flow control (6) – if, else, return, switch, case, default
- Loops (5) – for, do, while, break, continue
- Common types (5) – int, float, double, char, void
- structures (3) – struct, typedef, union
- Counting and sizing things (2) – enum, sizeof
- Rare but still useful types (7) – extern, signed, unsigned, long, short, static, const
- Evil keywords which we avoid (1) – goto
- More keywords(3) - auto, register, volatile

Naming Conventions

- C programmers generally agree on the following conventions for naming variables.
  - Begin variable names with lowercase letters
  - Use meaningful identifiers
  - Separate “words” within identifiers with underscores or mixed upper and lower case.
  - Examples: surfaceArea surface_Area surface_area
  - Be consistent!

Case Sensitivity

- C is case sensitive
- It matters whether an identifier, such as a variable name, is uppercase or lowercase.
- Example:
- area
- Area
- AREA
- ArEa
- are all seen as different variables by the compiler.

Which Are Legal Identifiers?

AREA	area_under_the_curve
3D	num45
Last-Chance	#values
x_yt3	pi
	%done
lucky***	Float
float	int_type
group one	

Which Are Legal Identifiers?

AREA	area_under_the_curve
3D	num45
Last-Chance	#values
x_yt3	pi
	%done
lucky***	Float
float	int_type
group one	

Declaring Variables

- Before using a variable, you must give the compiler some information about the variable; i.e., you must declare it.
- The general form of a variable declaration:

  data_type variable_name ;

  int value ;

  float area ;
- Variables of the same type can be defined in one declaration:

  data_type var_name_1 , var_name_2 , … , var_name_n;
- Examples of variable declarations:

  int value1, value2 ;

Declaring Variables

- When we declare a variable
  - Space is set aside in memory to hold a value of the specified data type
  - That space is associated with the variable name
  - That space is associated with a unique **address**
- Visualization of the declaration

  int num ;

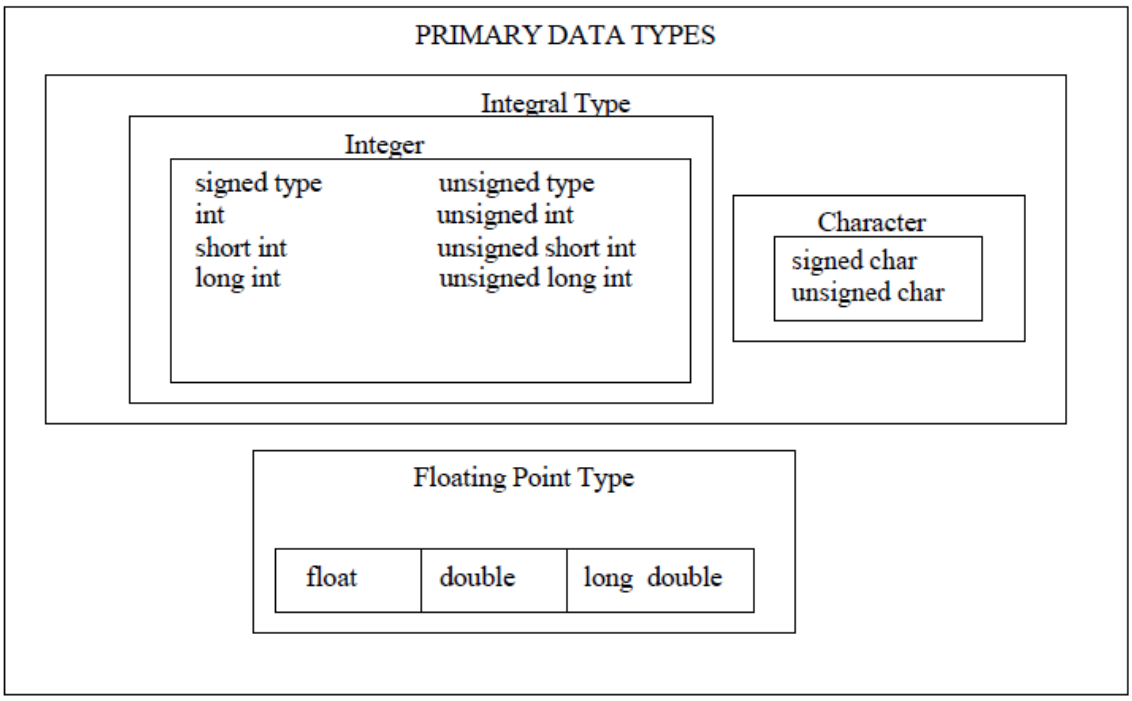
  num =10;

Data type

- Data type means what kind of value it can store and operations allowed on that data

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Primary Data Types



Integer data type

Type	Size (bits)	Range
int or signed int	16	-32,768 to 32767
unsigned int	16	0 to 65535
short int	8	-128 to 127
unsigned short int	8	0 to 255
long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295

Floating data type

Type	Size(bits)	Range
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932to 1.1E+4932

Character data type

Type	Size (bits)	Range
char	8	-128 to 127
unsigned char	8	0 to 255

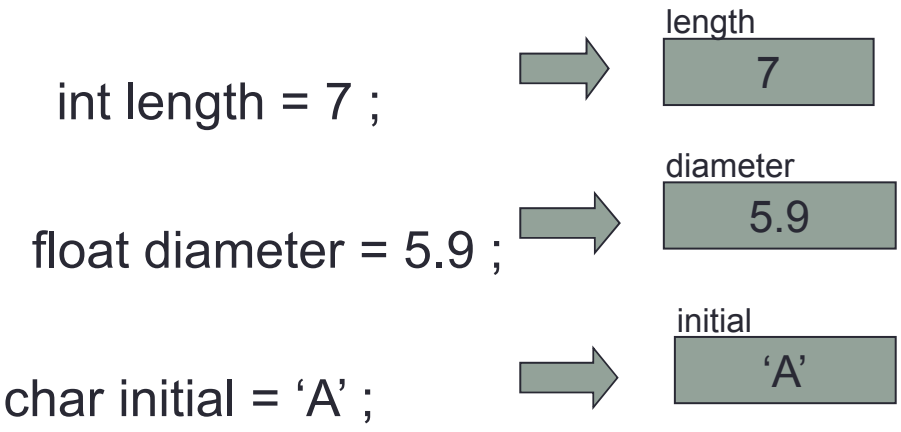
Using Variables: Assignment

- Variables may have values assigned to them through the use of an **assignment statement**.
- Such a statement uses the **assignment operator =**
- It assigns the value of the righthand side of the statement (the **expression**) to the variable on the lefthand side.
- Examples:

```
diameter = 5.9 ;
area = length * width ;
```

Note that only single variables may appear on the lefthand side of the assignment operator.

Using Variables: Initialization



Program

```
#include<stdio.h>
int main()
{
    int x,y;
    int sum;
    x = 10;
    y =20;
    sum = x+ y;
   printf(“ Sum is %d”, sum);
}
```

Reading input from Keyboard

- Another way of giving values to variables is to input data through keyboard using the **scanf** function.

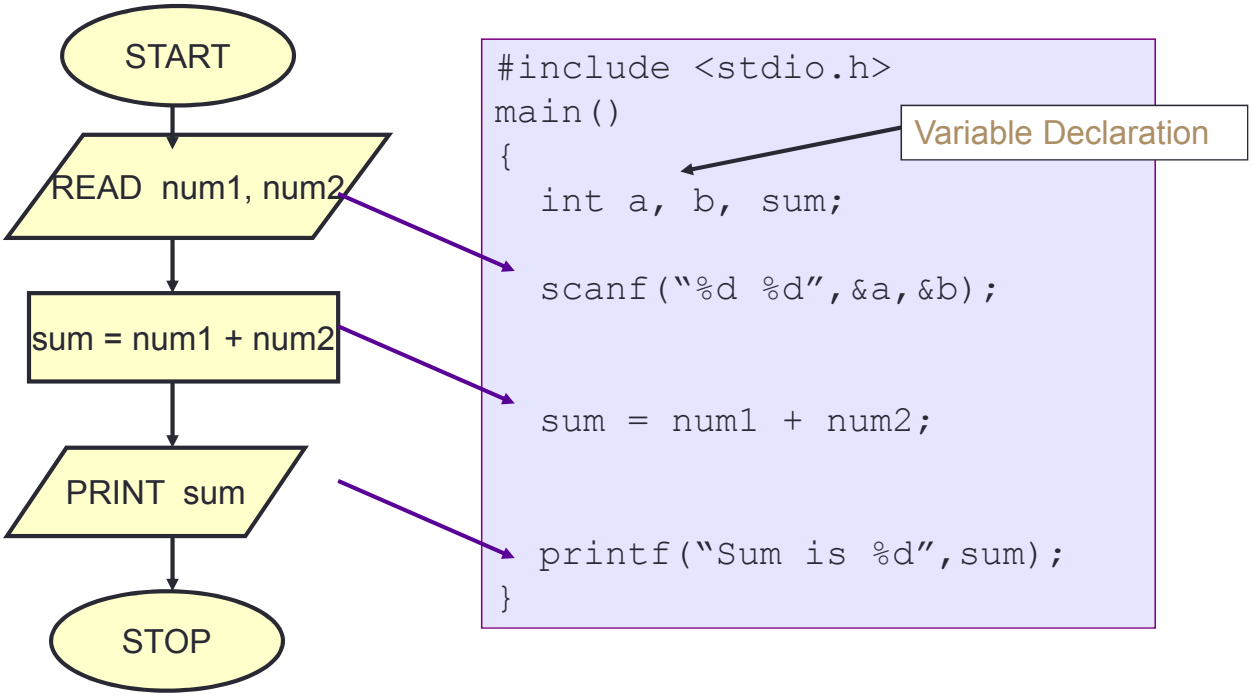
- The general format of **scanf** is as follows.
- **scanf**(“control string”,&variable1,&variable2,….);

- The ampersand symbol **&** before each variable name is an operator that specifies the variable name’s *address.*
- Eg: **scanf(“%d”,&number);**

Control strings

Type	Control String or format specifier
integer	%d
float	%f
char	%c

Adding two numbers



Practice time

- **WAP to accept integer number and display its square.**
- #include<stdio.h>
- int main()
- {
- int num;
- int square;
- printf("Enter the number");
- scanf("%d",&num);
- square = num*num;
- printf("The square of number = %d",square);
- return 0;
- }

Practice time

- **WAP to swap two integers**
- #include<stdio.h>
- int main()
- {
- int num1,num2;
- int temp;
- printf("Enter two numbers");
- scanf("%d %d",&num1,&num2);
- printf("value before swapping number1 = %d ,Number2 = %d \n“, num1,num2)
- temp = num1;
- num1=num2;
- num2 =temp;
- printf("value after swapping number1 = %d , Number2 =%d",num1,num2);
- }

Practice time

- **WAP to accept speed and time and calculate distance.**
- #include<stdio.h>
- int main()
- {
- float speed, time, distance;
- printf("Enter speed and time");
- scanf("%f %f", &speed,&time);
- distance = speed * time;
- printf ("\n The distance traversed is:%f \n",distance);
- return 0;
- }

Operators

- Operators are used in programs to manipulate data and variables.
- The combination of operators and operands is said to be an expression.

a+b

Operators

There are following types of operators to perform different types of operations in C language.

- 1. Arithmetic operators
- 2. Relational operators
- 3. Logical operators
- 4. Assignment operators
- 5. Increment and Decrement operators
- 6. Conditional operators
- 7. Bitwise operators
- 8. Special operators

Arithmetic operators

- The operators are
- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- % (Modulo division)
- Eg: 1) a-b 2) a+b 3) a*b 4) p%q
- The modulo division produces the remainder of an integer division.
- The modulo division operator cannot be used on floating point data.
- **Note: C does not have any operator for *exponentiation*.**

Integer Arithmetic

- When both the operands in a single arithmetic expression are integers, the expression is called an *integer expression ,* and the operation is called *integer arithmetic.*

Practice time

- int x =7 , y=3;
- x +y =
- x - y =
- x * y =
- x / y =
- x % y =

Practice time

- int x =7 , y=3;
- x +y = 10
- x - y = 4
- x * y = 21
- x / y = 2
- x % y = 1

Arithmetic operators

- During modulo division the sign of the result is always the sign of the first operand.
- That is
- -14 % 3 =
- -14 % -3 =
- 14 % -3 =

Arithmetic operators

- During modulo division the sign of the result is always the sign of the first operand.
- That is
- -14 % 3 = -2
- -14 % -3 = -2
- 14 % -3 = 2

Real Arithmetic

- An arithmetic operation involving only real operands is called *real arithmetic.*
- If **x and y** are floats then we will have:
- 1) x = 6.0 / 7.0 = 0.857143
- 2) y = 1.0 / 3.0 = 0.333333

- The operator % cannot be used with real operands.

Mixed-mode Arithmetic

- When one of the operands is real and the other is integer, the expression is called *a mixed-mode arithmetic* expression and its result is always a real number.
- Eg: 1) 15 / 10.0 = 1.5

Practice time

- int x=7, float y=3;
- x + y =
- x - y =
- x * y =
- x / y =
- x % y =
- 25/4.0
- 25 % 4.0
- 125.0 /10
-

Practice time

- int x=7, float y=3;
- x + y = 10.000000
- x - y = 4.000000
- x * y = 21. 000000
- x / y = 2.333333
- x % y = not allowed
- 25/4.0 = 6.25
- 25 % 4.0 = not allowed
- 125.0 /10 =12.5
-

Arithmetic operators

- If more than one arithmetic operator occurs in an expression we need to apply precedence first
- If more than one operator occurs with same precedence use law of associativity which is from left to right for arithmetic operators
- I - * / % - left to right
- II + -
- int a =5, b=7,c =3;
- int d
- d = a * b / c + a * b
- d=46
- Whenever any expression is evaluated complete entire right hand side and then place final value of RHS on LHS

Quiz time

```
int main()
{
   int a = 14;
   int  b =5;
   printf("%d + %d =%d\n",a,b,a+b);
   printf("%d - %d =%d\n",a,b,a -b);
   printf("%d * %d =%d\n",a,b,a*b);
   printf("%d / %d =%d\n",a,b,a/b);
   printf("%d % %d =%d\n",a,b,a%b);
}
```

Output

- 14 + 5 =19
- 14 - 5 =9
- 14 * 5 =70
- 14 / 5 =2
- 14 % 5 =4

Quiz time

```
int main()
{
   float a = 14;
   float  b =5;
   printf("%f + %f =%f\n",a,b,a+b);
   printf("%f - %f=%f\n",a,b,a -b);
   printf("%f * %f =%f\n",a,b,a*b);
   printf("%f / %f =%f\n",a,b,a/b);
   printf("%f  % %f =%f\n",a,b,a%b);
}
```

- 14.000000 + 5.000000 =19.000000
- 14.000000 - 5.000000=9.000000
- 14.000000 * 5.000000 =70.000000
- 14.000000 / 5.000000 =2.800000

- Invalid operands

Practice time

- **WAP to enter 2 digit number and find sum of digits**

```
#include<stdio.h>
int main()
{
  int num,digit1,digit2;
 int sum;
 printf("Enter 2 digit no");
 scanf("%d",&num);
 digit1= num/10;
 digit2=num%10;
 sum = digit1+ digit2;
 printf("Sum of digits is %d",sum);
  return 0;
}
```

Relational operators

- Used to compare quantities

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Relational operators

- The expression containing a relational operator is termed as a *relational expression.*
- The value of a relational expression is either *one* or *zero.*

10 > 20	is false = 0
25 < 35.5	is true = 1
12 > (7 + 5)	is false =0

- In C any non zero value is considered as true, 0 is considered as false
- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

  a + b > c – d is the same as (a+b) > (c+d)

Relational operators

Sample code

```
if  (x > y)
   printf (“%d is larger\n”, x);
else
  printf (“%d is larger\n”, y);
```

Quiz time

```
int main()
{
   int a = 8;
   int  b =-5;

   printf("%d < %d =%d\n",a,b,a<b);
   printf("%d == %d =%d\n",a,b,a==b);
   printf("%d != %d =%d\n",a,b,a!=b);
   printf("%d > %d =%d\n",a,b,a>b);
   printf("%d <= %d =%d\n",a,b,a<=b);
   printf("%d >= %d =%d\n",a,b,a>=b);
}
```

- 8 < -5 =0
- 8 == -5 =0
- 8 != -5 =1
- 8 > -5 =1
- 8 <= -5 =0
- 8 >= -5 =1

Logical operators

- C has the following three *logical operators.*
  - **&&** (logical **AND**)
  - **||** (logical **OR**)
  - **!** (logical **NOT**)

- Eg: 1) if(age>55 && sal<1000)
- 2) if(number<0 || number>100)

Logical Operators

- Logical AND (&&)
  - Result is true if both the operands are true.
- Logical OR (||)
  - Result is true if at least one of the operands are true.
- Logical NOT(!)
  - Negates the result

X	Y	X && Y	X || Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

Logical operators

- If there is a relational operator and logical operator, the relational operator takes higher preference
- AND operator does not check RHS if the LHS is false(0)
- OR operator does not evaluate the RHS if LHS is true(1)

Quiz time

- int i = 3, j = 7;
- float f = 5.5, g = 4.5;
- char ch = ‘T’

Expression	True/False	Value
(i <= 5) && (ch ==‘T’)		
(j < 8) || ch ==‘L’		
(f + g) == 10.0 || I < 2		
f>=6 || (i * j)<15		
! (f > 5.0)		

Quiz time

- int i = 3, j = 7;
- float f = 5.5, g = 4.5;
- char ch = ‘T’

Expression	True/False	Value
(i <= 5) && (ch ==‘T’)	True	1
(j < 8) || ch ==‘L’	True	1
(f + g) == 10.0 || I < 2	True	1
f>=6 || (i * j)<15	False	0
! (f > 5.0)	False	0

Assignment Operator

- The usual assignment operator is '='.
- C has a set of 'shorthand' assignment operators of the form,
- v **op** = exp;
- x += y+1;
- This is same as the statement
- x=x+(y+1);
- a + =1 is a = a + 1
- a - = 1 is a = a – 1
- a *= n + 1 is a = a * (n+1)

Increment and Decrement operator

- C has two very useful operators, *increment* and *decrement* operator: ++ and --
- The operator ++ adds 1 to the operands while - - subtracts 1.
- It takes the following form:
- ++m; or m++
- --m; or m—

Increment and Decrement operator

- When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then variable is incremented (or decremented) by one
- When prefix ++ (or --) is used in an expression, the variable is incremented (or decremented) first and then expression is evaluated using the new value of variable

Quiz time

```
int main()
{
    int p, q;
    p = 10;
    q = 20;
    /* post increment*/
    printf("Value of p        :%d\n",p);
    printf("Value of p++      :%d\n",p++);
    printf("new value of p    :%d\n",p);
```

Quiz time

- printf("Value of q :%d\n",q);
- printf("Value of ++q :%d\n",++q);
- printf("new value of q :%d\n",q);
- }

Quiz time

- Value of p :10
- Value of p++ :10
- new value of p :11
- Value of q :20
- Value of ++q :21
- new value of q :21

Quiz time

```
int main()
{
    int k =3,l=4,m;
    m=++k + --l;
    printf("Value of m %d\n",m);
    m=k++ + --l;
    printf("Value of m %d\n",m);
    printf("Value of k %d\n",k);
}
```

Quiz time

- Value of m 7
- Value of m 6
- Value of k 5

Conditional operator

- A ternary operator pair “?:” is available in C to construct conditional expression of the form:

  - exp1 ? exp2 : exp3;

- Here *exp1* is evaluated first. If it is true then the expression *exp2* is evaluated and becomes the value of the expression.
- If *exp1* is false then *exp3* is evaluated and its value becomes the value of the expression.

Quiz Time

```
int main()
{
    int a = 8,b =10,op1,op2;
    op1 = a < b ? a:b;
    printf("The option 1 = %d\n",op1);
    op2 = a > b ? a:b;
    printf("The option 2 = %d\n",op2);
}
```

Quiz Time

- The option 1 = 8
- The option 2 = 10

Special operators

- **The Size of Operator**
- The sizeof operator when used with an operand, it returns the number of bytes the operand occupies.

- Eg: 1) m = **sizeof**(sum);
- 2) n = **sizeof(long int)**
- 3) k = **sizeof**(235L)

Bitwise operators

Symbol	Operator
&	bitwise AND
|	bitwise inclusive OR
^	bitwise XOR (eXclusive OR)
<<	left shift
>>	right shift
~	bitwise NOT (one's complement)

Bitwise operators

- In C, following 6 operators are bitwise operators (work at bit-level)
- **& (bitwise AND)** Takes two numbers as operand and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
- **| (bitwise OR)** Takes two numbers as operand and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.
- **^ (bitwise XOR)** Takes two numbers as operand and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
- **<< (left shift)** Takes two numbers, left shifts the bits of first operand, the second operand decides the number of places to shift.
- **>> (right shift)** Takes two numbers, right shifts the bits of first operand, the second operand decides the number of places to shift.
- **~ (bitwise NOT)** Takes one number and inverts all bits of it

```
int main()
{
    int a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
   printf("a = %d, b = %d\n", a, b);
   printf("a&b = %d\n", a&b); // The result is 00000001
   printf("a|b = %d\n", a|b);  // The result is 00001101
   printf("a^b = %d\n", a^b); // The result is 00001100
   printf("~a = %d\n", a = ~a);   // The result is 11111010
   printf("b<<1 = %d\n", b<<1);  // The result is 00010010
   printf("b>>1 = %d\n", b>>1);  // The result is 00000100
   return 0;
}
```

Example

- a = 5, b = 9
- a&b = 1
- a|b = 13
- a^b = 12
- ~a = -6
- b<<1 = 18
- b>>1 = 4

Practice time

- **WAP to accept three numbers and find its average**
- #include<stdio.h>
- int main()
- {
- int num1,num2,num3;
- float average;
- printf("Enter three numbers");
- scanf("%d %d %d",&num1,&num2,&num3);
- average = (float)(num1 + num2 +num3)/3;
- printf("\nThe average of number = %f",average);
- return 0;
- }

Type Casting

```
int a=10, b=4, c;
float x, y;

c = a / b;
x = a / b;
y = (float) a / b;
```

The value of c will be 2
The value of x will be 2.0
The value of y will be 2.5

Operator Precedence

Highest	postfix ++, --
	unary +, -, prefix ++, --, !
	*, /, %
	binary +, -
	<, >, <=, >=
	=, !=
	&&
Lowest	||

Evaluation of Expression

- Rules for evaluation of expression
- When parentheses are used, the expressions within parentheses assumes highest priority.
- Evaluation begins with innermost sub expression

Practice Time

- WAP to accept distance in km and convert to meter, centimetre and inches.
- WAP to convert temperature in Fahrenheit to Celsius
- WAP to accept two digit number and display it in reversed form
- WAP to accept two numbers and display the result of their AND,OR,EXOR and NOT operator