Module Code: CSE401

Session 3: C Data types ,Operators & Expression

Session Speaker :

Prema Monish
premamonish.tll@msruas.ac.in

Session Objectives

- To know about C tokens
- To know about data types
- To know about operators and expressions
- To learn associatively of operators
- To learn evaluation of expression

Session Topics

- C tokens
- Data types
- Types of operators
- Arithmetic operators
- Logical operators
- Relational operators
- Increment & decrement operators
- Conditional operator
- Bitwise operator
- Special operator
- Precedence & associability of operators

Question

- We all know one or more human languages. What is a human language made up of?
  - Alphabets
  - Words
  - Sentences
  - Paragraphs
- A computer program
  - Character set
  - Tokens
  - Statements
  - Functions

C Tokens

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program
- C tokens : individual units.
  - Identifier – main( ) , amount, sum, …
  - Keyword – float, while, for, int,….
  - Constants – -13.5, 500, …
  - Strings – “ABC”, “MCA”, …
  - Operators – + - * % …
  - Special Symbols – [ ] { } ( ) …

Identifier

- It is sequence of one or more Letters or digits along with '_' special character (underscore)
- Refers to names of variables, functions,…
- User defined names.
- Uppercase and lowercase variables
- Variable name used to store data value.
- May take different values at different times during execution.
- Chosen by the programmer.

E.g.

Identifier	Valid/Invalid	Reason for invalid
Employee_id	Valid	-
a3	valid	-
Sum of digits	invalid	No spaces allowed
for	invalid	keyword
For1	Valid	-
3_factorial	invalid	Shouldn't start with digit

Keywords

- Reserved for doing some specific activities
- Fixed meaning, cannot be changed
- They have pre-defined meaning
- Should be written in lower case letters

Some Keywords

Keywords			
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

Constant

- It is fixed value which will not change during execution of program.
- Does not changes during execution of program.
  1. Numeric constant – Integer (decimal, octal, hexadecimal) and Real
  2. Character constant :
     - Single character constant : ‘A’
     - String constant: ”ABCD”

Constant

- Decimal constant –combination of digits from ‘0’ to ‘9’
  - E.x. 10,140,-20
- Octal constant-combination of digits from ‘0’ to ‘7’ with prefix 0
  - E.x. 085,023
- Hexadecimal constants-combination of digits from ‘0’ to ‘9’ and with letters ‘A’ to ‘F’ or ‘a’ to ‘f’ prefix with 0x or 0X
  - E.x. 0x205,0Xa20
- Real constant -constant with fractional part
  1. Decimal /floating point notation –has an integer part followed by decimal point and factional part E.x. 4.20,-9.63
  2. Scientific Notation- it has three parts i.e mantissa ,e or E and exponent

E.x. 6.63e2 means $6.63*10^2$

Variable

- Strings and numeric is stored in memory .
- memory location are named – variable
- it is sequence of one or more Letters or digits along with '_' special character (underscore)
- All variables must be declared before using them in a program and Variable declarations can also be used to initialize the variable. (not good practice)
- We can declare variables in a single statement with the list of variables separated by commas. Eg. int lower, upper, step;

Identifier	Valid/Invalid	Reason for invalid
Employee_name	Valid	-
2number	invalid	Should start with letter
Sum of digits	invalid	No spaces allowed
for	invalid	keyword
Ab_c	Valid	-
Factor!	invalid	Only'_' is allowed

Scope of Variables

- Scope is how far a variable is ***accessible*** and ***life*** is how much time does a variable is existed in the memory (life of variable).
- The scope and life of a variable depends on the location where a variable is declared
- According to their declaration, variables are classified into 3 categories
  1. Block variables
  2. Internal or Local variables
  3. External or Global variables.

Block Variables

- Variables declared within a pair of braces { } are called block variables.
- A block { } may be an independent or in association with any control structure but, not with a function.
- ***Scope of block variables:*** Block variables can be accessed within the block in which they are declared, can also be accesses into the inner block which is within the current block but, can't be accessed outside the block.
- ***Life of block variables:*** These variables appear as the control enters into the block and disappears as the control go out of the block. Hence these variables can't be accessed outside the block.

Block Variables

```
/* scope of block variables */
#include<stdio.h>
int main()
{
 {                          /* outer block */
   int x=10;
   {                           /* inner block */
       printf("x=%d",x);
   }
   printf("\nx=%d",x);
 }
 return 0;
}
```

Output:
x=10
x=10

Local Variables

- Variables declared within a function are called ***local variables***
- ***Scope of local variables:*** The visibility of these variables is limited to the home function in which they are declared.
- ***Life of local variables***: Local variables in a function come into existence only when a function is called, disappears on the completion of function execution.
- Local variables are stored in a place called **Stack Segment** in the computer memory and local to the home function.

Local Variables

- As the memory allocation of local variables allocate (born) on entering the control into the function and de-allocate (dead) on completion of function execution automatically, these variables are called ***automatic variables.***

```
/* scope of automatic variable */
#include<stdio.h>
int main()
{
 int x=10;
 display();
 return 0;
}
void display()
{
 printf("x=%d",x);
}
```

Output:
Error: undefined symbol “x” in function display()

Global Variables

- Variables declared outside the functions are called ***global variables***.
- These variables are available along down to the program from their declaration.
- ***Scope of external variables:*** Global variables are visible only to the functions, which are down to their definition, can't be accessed from the functions above to their definition.
- ***Life of external or global variables***: External variables come into existence on initiating the execution of program, stay permanently as long as the program executes. These variables disappear only on completion of program execution.
- Unlike local variables, external variables keep their values as long as the program executes.

Global Variables

```
/* Scope of external variables */
#include<stdio.h>
int x=10;       /* external variable definition */
void display();
int main()
{
  printf("x=%d",x);  /* printing external variable */
  display();
  printf("\nx=%d",x);/* printing external variable */
  return 0;
}
void display()
{
  x=x+50;  /* changing external variable */
}
```

Output:
x=10
x=60

Data Types

- Type of data that a variable can store is called ***data type***
  1. Primary data types-integer,charcter,floating point,double
  2. User defined data type-typedef,structure, unions,enums
  3. Derived data type-array, function, pointer

Type	Size (bits)	Range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or signed int	16	-32678 to 32767
unsigned int	16	0 to 65535
short int or signed short int	8	-128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	-2147483648 to 1247483647
unsigned long int	32	0 to 4294967295
float	32	3.4 E –38 to 3.4 E + 38
double	64	1.7 E –308 to 1.7 E + 308

Variable Declarations in C

- All variables must be declared before using them in a program.
- Variable declarations can also be used to initialize the variable.
- We can declare variables in a single statement with the list of variables separated by commas.
  - int lower, upper, step;
  - float fahr, celsius;
  - int i=0; // but bad practice
  - char backslash = '\\'

Characters Representation

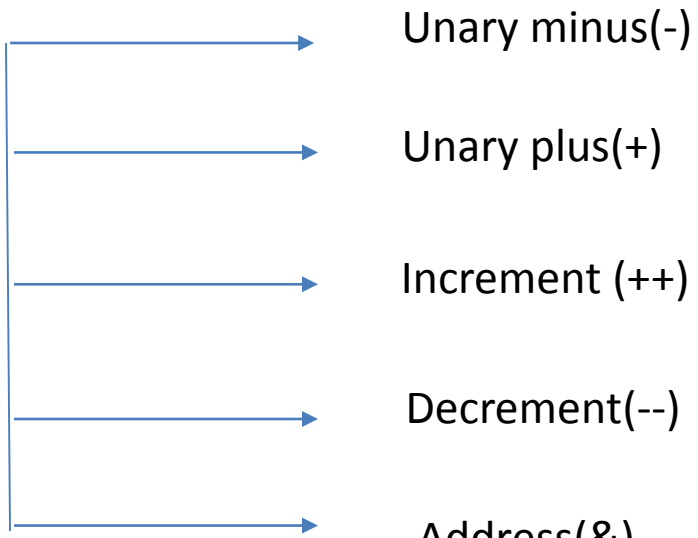
- Characters are represented at the machine level as an integer
- Computers use ASCII (American Standard Code for Information Interchange) code
- The values used as code range from 0 to 255
- A-Z (upper case) are represented by 65-90
- a-z (lower case) are represented by 97-122
- Syntax-printf("ASCII value of character %c: %d\n", i, i);

Operators

- An operator is a symbol that specifies the operation to be performed.
- Operators can be classified into three major categories:-
  - Unary operators
  - Binary operators
  - Ternary operators

Unary operators

- An operator which acts on only one operand to produce the result – unary operand
- Unary operator



Unary operators

- ++, – – are unary operators used to increment, decrement the value of a variable by 1.
- These operators can be used as both prefix and postfix to the operands.

Postfix Operator

- If we use the operator to the right side of an operand then it is called postfix operator (i++, i- -).
- Postfix operator increments or decrements the value of a variable by 1 after performing other operations like assigning, comparing or manipulating

```
#include<stdio.h>
int main()
{
    int x,y;
    x=45;
    y=x++;      /*  Increments after assignment */
    printf("x=%d\ny=%d",x,y);
    return 0;
}
```

Output:

x=46

y=45

Prefix operator

- If we use the operator to the left side of an operand then it is called prefix operator (++i, --i).
- Prefix operator Increments or decrements the value of a variable by 1 before performing other operations.

```
#include<stdio.h>
int main()
{
int x,y;
x=45;
y=++x;      /*  Increments before assignment */
printf("x=%d\ny=%d",x,y);
return 0;
}
```

Output:
46
46

Binary Operators

- An operator performs operation on two operands to produce result

Arithmetic operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- % Modulus (remainder)

Relational Operators

- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to
- == Identically equal to
- != Not identically equal

Binary Operators

Logical operators

- && Logical AND
- || Logical OR
- ! NOT

Assignment operators

- += Addition
- -= Subtraction
- *= Multiplication
- /= Division
- %=Modulus (remainder)

Binary Operators

Bitwise Logical operators

- & Bitwise AND
- | Bitwise OR
- ^ Bitwise XOR
- ~ Bitwise NOT
- >> Right shift
- << Left shift

Assignment operators

- It is a binary operator used in between any two operands.
- It is an important operator used to assign a value of a constant, variable or an expression to a variable.
- The part to the right side of assigning operator is called r-value. This may be a variable, constant or an expression
- The part to the left side of assigning operator is called l-value. This must be a variable.

```
l-value = r-value
```

Assignment operators

Operator	Example	Meaning
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

```
int x=10;
x+=80;                /* equals to x=x+80 */
printf("%d",x);
x-=80;                  /* equals to x=x-80  */
printf("\t%d",x);
x*=20;                 /* equals to x=x*20  */
printf("\t%d",x);
x/=20;                  /* equals to x=x/20  */
printf("\t%d",x);
x%=4;                  /* equals to x=x%4  */
printf("\t%d",x);
```

Output:
90 10 200 10 2

Logical Operators

- NOT reverses the truth value of its operand:

Expression	Return Value
!1	0
!0	1

- AND returns 1 if both operands return non zero values Expression Return Value

1 && 1	1
1 && 0	0
0 && 1	0
0 && 0	0

- OR only returns 0 if both operands return zero:

Expression	Return Value
1 || 1	1
1 || 0	1
0 || 1	1
0 || 0	0

NOT has higher precedence than AND, which has higher precedence than OR.

Bitwise Logical Operator

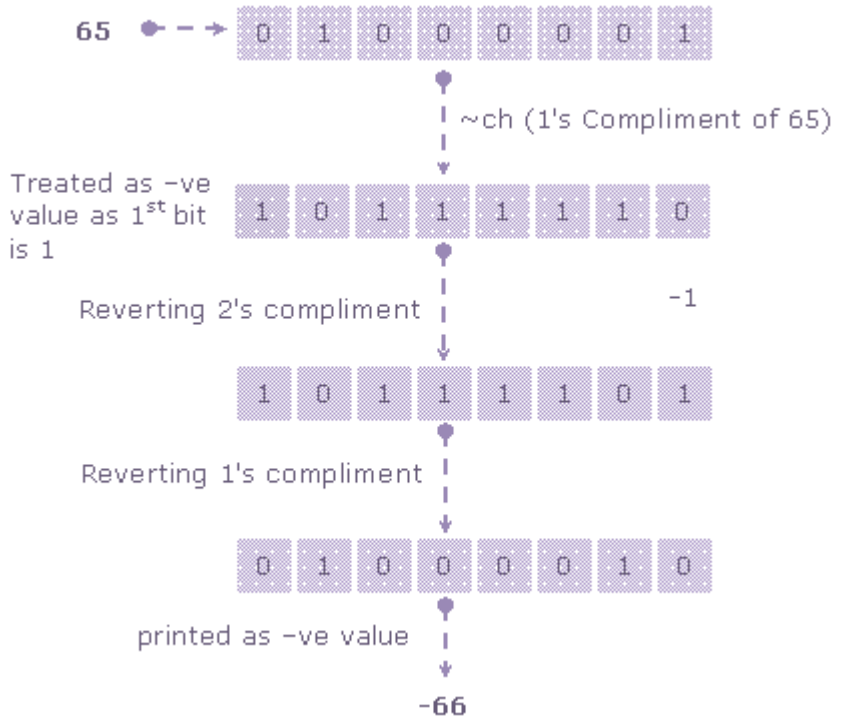
- & bit wise *AND:* 9 & 23 =01001 & 10111=00001
- | bit wise OR : 9 & 23 = 01001 & 10111=11111
- ^ bit wise exclusive *OR* 9 ^ 23=01001^ 10111=11110
- << left shift - takes away bit on the left, add zeros to the right bit, a << b
- >> right shift - takes away bit on the right, add zeros to the left bit ,
  a >> b


```
char ch=13;
printf("%d",ch<<3);
```

Bitwise Logical Operator

int c=65;

printf("after negation %d",~c);// it performs 1's complement and 2's complement



Ternary Operators

- **The Conditional Operator** - It's possible to say: "If this condition is true, do this.... , otherwise, do this .... " all in one line. This can be achieves using the **CONDITIONAL OPERATOR**, which is represented by is represented by **? :**
  - It may look a bit odd at first, but it takes 3 expressions as operands, like this: a ? b : c; a should be a condition you want to test for. b is an expression that is evaluated if a returns a non zero value (in other words, if a is true), else expression c is evaluated.
  - For example: x<1 ? printf("x<1") : printf("x>=1");
  - Note the 3 separate operands are expressions - the whole line is a statement. It's a common mistake to put a semi colon in the middle expression like this:
  - x<1 ? printf("x<1"); : printf("x>=1"); - this will generate an error when you compile!

sizeof() Operator

- sizeof() operators returns the size (number of bytes) of the operand occupies
- Must precede its operand
- Operand may be a constant, a variable or a data type
- Syntax

```
sizeof(operand);
```

- Example

```
x=sizeof(int);
y=sizeof(x);
```

Rules of Associativity

- All the operators available in C language with their precedence and associativity are summarized in the following table.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Evaluation of Expression

Operation 1:

$$5 + 6 - 11 + \underbrace{8 * 5}_{40} + 4 = ?$$

Operation 2:

$$\underbrace{5 + 6}_{11} - 11 + 40 + 4 = ?$$

Operation 3:

$$\underbrace{11 - 11}_{0} + 40 + 4 = ?$$

Operation 4:

$$\underbrace{0 + 40}_{40} + 4 = ?$$

Operation 5:

$$\underbrace{40 + 4}_{44} = 44$$

Summary

- Data types specify the size, representation and organisation of data in the computer's memory
- Primitive Data types are the basic data types that are provided by the programming language
- Three primitive data types were discussed in this session along with their representation, size, format specifiers, related operators and constants
  - Integer
  - Float
  - Character
- User defined Data types are used to define an identifier that would represent an existing data type

Summary

- Expressions consist of operators, the symbols that represent an operation, and operands, the data items on which the operation is applied
- There are many types of operators (arithmetic, comparison, logical and bitwise to name a few)
- Expressions are evaluated based on precedence and associativity of operators

ANY QUERIES