Rajarata University of Sri Lanka
Department of Physical Sciences

COM1407
Computer Programming

LECTURE 04

C OPERATORS

T.C IRUGALBANDARA

Objectives

- At the end of this lecture students should be able to;
  - Define the terms operators, operands, operator precedence and associativity.
  - Describe operators in C programming language.
  - Practice the effect of different operators in C programming language.
  - Justify evaluation of expressions in programming.
  - Apply taught concepts for writing programs.

Operators and Operands

- Operands
  - Operands are numerical, text and Boolean values that a program can manipulate and use as logical guidelines for making decisions.
  - Like with math, computer programs can use constant values and variables as operands.
  - If you have the variable "dozens" with the value of “2" and the constant "12" in the equation "dozens*12 = 24."
  - Both "dozens" and "12" are operands in the equation.

Operators and Operands (Cont…)

- Operators
  - Operators are used to manipulate and check operand values.
  - In C programming Operators are symbols which take one or more operands or expressions and perform arithmetic or logical computations.
  - If the operator manipulates one operand it is known as unary operator; if the operator manipulates two operands, it is known as binary operator and if the operator manipulates three operands, it is known as ternary operator.

Operators and Operands (Cont…)

- C has following operators
  - Arithmetic operators → all are binary operators
  - Logical operators → all are binary operators
  - Relational operators → all are binary operators
  - Bitwise operators → all are binary operators except ! and ~
  - Increment, decrement operators → unary operators
  - Assignment operators → binary operators
  - Conditional operator → ternary operator
  - Type cast operator →unary operator
  - sizeof operator → unary operator
  - Comma operator → binary operator

Arithmetic Operators

Operator	Description
A+B	adds A with B;
A-B	subtracts B from A;
A*B	multiplies A by B;
A/B	divides A by B;
I%J	gives the remainder of I divided by J. Here I and J are expressions of any integer data type;

Here A and B are expressions of any basic data type

Arithmetic Operators (Cont…)

```
#include <stdio.h>

int main (void)

{

      int a = 100;

      int b = 2;

      int c = 25;

      int d = 4;

      int result;

      result = a - b;

      printf ("a - b = %i\n", result);

      result = b * c;

      printf ("b * c = %i\n", result);

      result = a / c;

      printf ("a / c = %i\n", result);

      result = a + b * c;

      printf ("a + b * c = %i\n", result);

      printf ("a * b + c * d = %i\n", a * b + c * d);

      return 0;

}
```

Arithmetic Operators (Cont...)

```
#include <stdio.h>
int main (void)
{
   int a = 25;
   int b = 2;
   float c = 25.0;
   float d = 2.0;
   printf ("6 + a / 5 * b = %i\n", 6 + a / 5 * b);
   printf ("a / b * b = %i\n", a / b * b);
   printf ("c / d * d = %f\n", c / d * d);
   printf ("-a = %i\n", -a);
   return 0;
}
```

Arithmetic Operators (Cont...)

```
#include <stdio.h>
int main (void)
{
   int a = 25, b = 5, c = 10, d = 7;
   printf ("a %% b = %i\n", a % b);
   printf ("a %% c = %i\n", a % c);
   printf ("a %% d = %i\n", a % d);
   printf ("a / d * d + a %% d = %i\n",
   a / d * d + a % d);
   return 0;
}
```

Logical Operators

Operator	Description
A && B	has the value 1 if both A and B are nonzero, and 0 otherwise (and B is evaluated only if A is nonzero);
A || B	has the value 1 if either A or B is nonzero, and 0 otherwise (and B is evaluated only if A is zero);
!A	has the value 1 if a is zero, and 0 otherwise.

A and B are expressions of any basic data type except void, or are both pointers;

Logical Operators (Cont…)

```
#include <stdio.h>

int main()
{
   int a = 5;
   int b = 20;
   int c = 1;

   printf( "( a && b ) = %d \n", a && b );
   printf( "( a || b ) = %d \n", a || b );
   printf( " !a  = %d \n", !a  );
   return 0;
}
```

Relational Operators

Operator	Description
A < B	has the value 1 if A is less than B, and 0 otherwise;
A <= B	has the value 1 if A is less than or equal to B, and 0 otherwise;
A > B	has the value 1 if A is greater than B, and 0 otherwise;
A >= B	has the value 1 if A is greater than or equal to B, and 0 otherwise;
A == B	has the value 1 if A is equal to B, and 0 otherwise;
A != B	has the value 1 if A is not equal to B, and 0 otherwise;

A and B are expressions of any basic data type except void, or are both pointers;

Relational Operators (Cont…)

- The usual arithmetic conversions are performed on A and B.
- The first four relational tests are only meaningful for pointers if they both point into the same array or to members of the same structure or union.
- The type of the result in each case is int.

Relational Operators (Cont…)

```
#include <stdio.h>


int main()
{
   int a = 5;
   int b = 20;
   int c = 5;


   printf( "( a < 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( "( c == a ) = %d \n", c == a );
   printf( "( a != b ) = %d \n", a != b );
   return 0;
}
```

Bitwise Operators

Operator	Description
A & B	performs a bitwise AND of A and B;
A | B	performs a bitwise OR of A and B;
A ^ B	performs a bitwise XOR of A and B;
~A	takes the ones complement of A;
A << n	shifts A to the left n bits;
A >> n	shifts A to the right n bits;

A, B, n are expressions of any integer data type;

Bitwise Operators (Cont…)

- The usual arithmetic conversions are performed on the operands, except with << and >>, in which case just integral promotion is performed on each operand.
- If the shift count is negative or is greater than or equal to the number of bits contained in the object being shifted, the result of the shift is undefined.

Bitwise Operators (Cont....)

- Bitwise operator works on bits and perform bit-by-bit operation.
- The truth tables for &, |, and ^ is as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Bitwise Operators (Cont…)

- Assume A = 60 and B = 13 in binary format, they will be as follows:

  A = 0011 1100

  B = 0000 1101

  A&B = 0000 1100

  A | B = 0011 1101

  A^B = 0011 0001

  ~A = 1100 0011

  A << 2 = 240 i.e., 1111 0000

  A >> 2 = 15 i.e., 0000 1111

Bitwise Operators (Cont…)

```
#include <stdio.h>
int main()
{
   int a = 60;
   int b = 13;
   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( "( a << 2 ) = %d \n", a << 2);
   printf( "( a >> 2 ) = %d \n", a >> 2);
   return 0;

}
```

Increment and Decrement Operators

Operator	Description
++A	increments A and then uses its value as the value of the expression;
A++	uses A as the value of the expression and then increments A;
--A	decrements A and then uses its value as the value of the expression;
A--	uses A as the value of the expression and then decrements A;

A is a modifiable value expression, whose type is not qualified as const.

Increment and Decrement Operators (Cont…)

```
#include <stdio.h>
int main() {
   int a = 21;
   int c =10;;

   c = a++;
   printf( "Line 1 - Value of a and c are : %d - %d\n", a, c);
   c = ++a;
   printf( "Line 2 - Value of a and c are : %d - %d\n", a, c);
   c = a--;
   printf( "Line 3 - Value of a and c are : %d - %d\n", a, c);
   c = --a;
   printf( "Line 4 - Value of a and c are : %d - %d\n", a, c);
   return 0;
}
```

Assignment Operators

- A = B → stores the value of B into A;
- A *op*= B →applies *op* to A and B, storing the result in A.

A is a modifiable value expression, whose type is not qualified as const;
op is any operator that can be used as an assignment operator ; B is an expression;

Assignment Operators (Cont…)

- In the first expression, if B is one of the basic data types (except void), it is converted to match the type of A.
- If A is a pointer, B must be a pointer to the same type as A, a void pointer, or the *null* pointer.
- If A is a void pointer, B can be of any pointer type.
- The second expression is treated as follows;

  A = A *op* (B)

Assignment Operators (Cont…)

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

Assignment Operators (Cont…)

Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Assignment Operators (Cont…)

```
#include <stdio.h>


int main()

{


   int a = 21;

   int c ;


   c =  a;

   printf("Line 1 - =  Operator Example, Value of c =
%d\n", c );

   c +=  a;

   printf("Line 2 - += Operator Example, Value of c =
%d\n", c );
```

Assignment Operators (Cont…)

```
   c /=  a;

   printf("Line 5 - /= Operator Example, Value of c
= %d\n", c );

   c  = 200;

   c %=  a;

   printf("Line 6 - %= Operator Example, Value of c
= %d\n", c );

   c <<=  2;

   printf("Line 7 - <<= Operator Example, Value of c
= %d\n", c );

   c >>=  2;

   printf("Line 8 - >>= Operator Example, Value of c
= %d\n", c );
```

Assignment Operators (Cont…)

```
c ^=  2;

   printf("Line 10 - ^= Operator Example, Value of c =
%d\n", c );


   c |=  2;

   printf("Line 11 - |= Operator Example, Value of c =
%d\n", c );


   return 0;


}
```

Conditional Operators

- Given that A, B, C are expressions; then the expression

  A ? B : C

has as its value B if A is nonzero, and C otherwise; only expression B or C is evaluated.

In order to generate 1 or 0 for A, it should be a relational or logical expression.

Conditional Operators (Cont…)

- Expressions B and C must be of the same data type.
- If they are not, but are both arithmetic data types, the usual arithmetic conversions are applied to make their types the same.
- If one is a pointer and the other is zero, the latter is taken as a null pointer of the same type as the former.
- If one is a pointer to void and the other is a pointer to another type, the latter is converted to a pointer to void, and that is the resulting type.

Conditional Operators (Cont…)

```
#include<stdio.h>


int main()
{
      int num;
      int flag;


      printf("Enter the Number : ");
      scanf("%d",&num);


      flag = ((num%2==0)?1:0);


      printf ("Flag value : %d \n", flag);
      return 0;
}
```

Type Cast Operator

- You can convert the values from one type to another explicitly using the **cast operator** as follows:

  (type_name) expression

- Refere Lecture 04

sizeof Operator

- Given that *type* is as the name of a basic data type, an enumerated data type (preceded by the keyword enum), a typedef-defined type, or is a derived data type; A is an expression; then the expression

  sizeof (*type*) ➔has as its value the number of

  bytes needed to contain a value of the specified

  type;

  sizeof A ➔ has as its value the number of bytes

  required to hold the result of the evaluation of A.

sizeof Operator (Cont…)

```
#include<stdio.h>

int main() {

   int ivar = 100;
   char cvar = 'a';
   float fvar = 10.10;
   double dvar = 18977670.10;

   printf("%d\n", sizeof(ivar));
   printf("%d\n", sizeof(cvar));
   printf("%d\n", sizeof(fvar));
   printf("%d\n", sizeof(dvar));
   return 0;
}
```

Comma Operator

- The comma operator can be used to link the related expressions together.
- A comma linked expression is evaluated from left to right and the value of the right most expression is the value of the combined expression.

  x = (a = 2, b = 4, a+b)

- In this example, the expression is evaluated from left to right.
- So at first, variable ‘a’ is assigned value 2, then variable ‘b’ is assigned value 4 and then value 6 is assigned to the variable ‘x’.
- Comma operators are commonly used in for loops, while loops, while exchanging values.

Comma Operator (Cont…)

- Comma operator returns the value of the rightmost operand when multiple comma operators are used inside an expression.
- Comma Operator Can acts as :
  - **Operator** : In the Expression
  - **Separator** : Declaring Variable , In Function Call Parameter List

Comma Operator (Cont...)

```
#include<stdio.h>
int main()
{
   int num1 = 1, num2 = 2;
   int res, x, a, b;

   x = (a = 2, b = 4, a+b);

   res = (num1, num2);
   printf("%d\n", res);

   x = (a = 2, b = 4, a+b);
   printf("%d\n", x);
   return 0;
}
```

C Arithmetic Expression

- Arithmetic expression in C is a combination of variables, constants and operators written in a proper syntax.
- C can easily handle any complex mathematical expressions but these mathematical expressions have to be written in a proper syntax.

Operator Precedence & Associativity

- Operator Precedence
  - When an expression contains multiple operators, the precedence of the operators controls the order in which the individual operators are evaluated.
  - For example, the expression x + y * z is evaluated as x + (y * z) because the * operator has higher precedence than the binary + operator.
  - The precedence of an operator is established by the definition of its associated grammar production.

Operator Precedence & Associativity (Cont…)

- Operator Associativity
  - A higher precedence operator is evaluated before a lower precedence operator.
  - If the precedence levels of operators are the same, then the order of evaluation depends on their associativity (or, grouping).

Operator Precedence & Associativity (Cont…)

- Example

(1 > 2 + 3 && 4)

This expression is equivalent to:

((1 > (2 + 3)) && 4)

i.e, (2 + 3) executes first resulting into 5 then, first part of the expression (1 > 5) executes resulting into 0 (false) then, (0 && 4) executes resulting into 0 (false)

Operator Precedence & Associativity (Cont…)

(1 > 2 + 3 && 4) can be written as follows;

=((1 > (2 + 3)) && 4)

=((1>5) && 4)

=(0 && 4)

=0

Summary of C Operators

- These operators are listed in order of decreasing precedence.
- Operators grouped together have the same precedence.

Summary of C Operators (Cont…)

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

Questions

- Try this out:
  - 250*2/4+1
  - 5%2*5/5
  - 2+300-200/2
  - 1 && 1 || 0
  - 1 || 1 +3-4 && 1

Objective Re-cap

- Now you should be able to:
  - Define the terms operators, operands, operator precedence and associativity.
  - Describe operators in C programming language.
  - Practice the effect of different operators in C programming language.
  - Justify evaluation of expressions in programming.
  - Apply taught concepts for writing programs.

References

- Chapter 04, Appendix A, section 5 - Programming in C, 3rd Edition, Stephen G. Kochan

Q & A

NEXT: PROGRAM CONTROL STRUCTURES – DECISION MAKING & BRANCHING