C-Refresher: Session 05
Operators in C

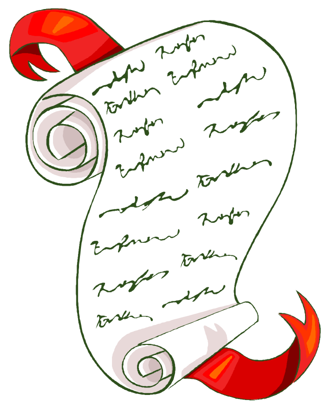
Arif Butt
Summer 2017

I am Thankful to my student Muhammad Zubair bcsf14m029@pucit.edu.pk for preparation of these slides in accordance with my video lectures at

http://www.arifbutt.me/category/c-behind-the-curtain/

Today’s Agenda

- Review of Data types
- Type Conversion
- Operators in C
- Properties of Operators
- Bitwise Operators
- Understanding and Applications of Bitwise Operators



Review of Data Types

- Variables that we declare in C have some specific datatype specified
- Along with some datatype, variables also have a storage class associated with them

❑Storage Classes:

- These are the classes which provide information about the location and visibility of the variable
- There are different storage classes available in C which are used for different purposes

Review of Data Types(cont…)

1.auto

- This is the default class
- All variables that you declare are local to a block or function. These variables are discarded when you exit from the block or function
- You can use `register` keyword with automatic variable
- The variables that are automatic are stored in registers, if possible, for quick access
- If `auto` is not mentioned then it is considered implicit

Review of Data Types(cont…)

2. static

- The `static` variables can be made
  - Within a function
  - Outside a function
- **Within a function**
  - If a variable is declared `static` within a function, it will retain its value between various calls to the function, i.e. when the user exits the function the value of the variable is not discarded rather it is retained and when a new call is made to the function that retained value can be used

Review of Data Types(cont…)

- **Outside the function**
  - The variable that are declared `static` outside the function are global variables, and they can be accessed at any time
- Now declaring a variable outside a function can have two cases
  - With `static` keyword--gives internal linkage, means access within that `.o` or `.c` file in which variable is declared not for other object files
  - Without `static` keyword--gives external linkage, means access in the entire program i.e. in all the `.o` files

Review of Data Types(cont…)

3. extern

- Declaring a variable with `extern` keyword gives it an external linkage

❑Type Qualifier

- Defines some special properties of the variable being declared
- There are two types of type qualifiers
  - `const`
  - `volatile`

Review of Data Types(cont…)

- **const**
  - Places a variable in read only memory and increases the opportunities for optimization
  - e.g. `const double GRAVITY=9.8;`
- **volatile**
  - It tells the compiler that the variable value may be changed at anytime outside the program
  - e.g. `volatile int date;` => it indicates that the value of variable `date` might have changed by another program

Review of Data Types(cont…)

- `volatile const int location=725780;` => it indicates that the value of the variable `location` may be modified by another program but it cannot be changed inside the program

Type Conversion

- There are two types of conversion from one datatype to the other
  - implicit
  - explicit
- **Implicit conversion:**
  - When an expression involves variables of different datatypes then they are needed to be converted to a common datatype, this is done implicitly
  - A lower datatype is converted to a higher datatype before an arithmetic operation proceeds, i.e. a `char` to `short`, `short` to `int`, `int` to `long`, `long` to `long long`, `long long` to `float`, `float` to `double`, `double` to `long double` and so on

Type Conversion(cont…)

- In an assignment operator expression, the value is converted according to the datatype of the variable on the left of the assignment operator, this may sometimes cause loss of data
- e.g.
- `long l, int i, float f, double d`
- `int x = l/i + i*f – d;`
- The final result of the arithmetic computation will be a `double`, which will be converted to an `int`

Type Conversion(cont…)

- **Explicit Conversion/Casting:**
- Casting is a way of converting from one datatype to another datatype, maybe forcefully
- **Syntax**
  - `(type-name)expression` or
  - `type-name(expression)`
- **e.g.** `int x; long l; float f;`
- `x = (int)7.5;`   /*`7.5` will be converted to an `int`(`.5` will be truncated)*/

Type Conversion(cont…)

- `x = (int)21.3/int(4.5);` //at first `21.3` and `4.5` will both be truncated to an `int` and then the division operation will be carried out
- `x = (int)(l+f);` //`l` and `f` will be added first and the result will be converted to an `int`
- `x = (int)ch1 + ch2;` //`ch1` will be converted to an `int` first and then added to `ch2`

Operators in C

- C language is very rich in built-in operators, which can be divided as

1. Arithmetic Operators
   - `-, +, *, /, %, ++, --`
2. Relational Operators
   - `<, <=, >, >=, ==, !=`
3. Logical Operators
   - `!, &&, ||`
4. Bitwise Operators
   - `&, |, ^, ~, <<, >>`

Operators in C(cont…)

5. Assignment Operators
   - `=, +=, -=, *=, /=, %=, <<=, >>=`
6. Misc Operators
   - `sizeof(), &, *, ?:`

Properties of Operators

1. **Arity of Operator**
   - It is the number of operands an operator can take
   - e.g.
     - `a++` => arity = 1
     - `a+b` => arity = 2
2. **Precedence of Operator**
   - When there are more than one operators involved in an expression then it is the precedence of the operators which decides that which operator should be evaluated first

Properties of Operators(cont…)

- e.g. `1 > 2 + 3 && 4` /*there are 3 operators involved in the expression `>`, `+` and `&&` */
- The order of evaluation of this expression will be

```
1.+  => 2+3=5
2.>  => 1>5=0
3.&& => 0&&4=0
```

- It's better to write this expression like
  - `((1>(2+3))&&4)`

- **Note:** In C, zero means false, and anything else means true

Properties of Operators(cont…)

3. **Associativity of Operator**
   - This property is used when two or more operators in an expression have the same precedence
   - It can be
     - left associative => expression on the left should be evaluated first, e.g. + operator is left associative
     - right associative => expression on the right should be evaluated first, e.g. ^ operator is right associative

Bitwise Operators

- Bitwise operators are very important in low level programming
- Applications of Bitwise operators include
    1. Checking file permissions
    2. Low level device control
    3. Error detection & Correction
    4. Data Compression Algorithm
    5. Encryption Algorithm

Understanding Bitwise Operators

1. **NOT(~)**
   - It simply switches the bits from `0->1` and `1->0`
   - e.g.
   - `unsigned char ch=5;` //`ch=0000 0101`
   - `ch = ~ch;` //`ch=1111 1010`
   - `printf(“%c”,ch);` //will print `250`
2. **AND(&)**
   - It applies `AND` operation on the bits of the two numbers on which it is applied

Understanding Bitwise Operators(cont…)

- e.g.
- `unsigned char ch1=5;   //ch1=0000 0101 = 5`
- `unsigned char ch2=4;   //ch2=0000 0100 = 4`
- `unsigned char ch3=ch1&ch2; //ch3=0000 0100=4`

3. **OR(|)**
   - It applies `OR` operation on the bits of the two numbers on which it is applied
   - e.g.
   - `unsigned char ch1 = 5; //ch1=0000 0101 = 5`
   - `unsigned char ch2 = 4; //ch2=0000 0100 = 4`
   - `unsigned char ch3=ch1 | ch2;//ch3=0000 0101=5`

Understanding Bitwise Operators(cont…)

4. XOR(^)

- It performs `XOR` operation on the two numbers on which it is applied
- In `XOR` operation, the result is `1` if odd number of bits are `1`, otherwise the result is `0`
- e.g.
- `unsigned char ch1 = 5; //ch1=0000 0101 = 5`
- `unsigned char ch2 = 4; //ch2=0000 0100 = 4`
- `unsigned char ch3=ch1 ^ ch2;//ch3=0000 0001=1`

Understanding Bitwise Operators(cont…)

5. Left Shift(<<)

- It adds `n` no. of `0-bit(s)` on the right side of the bits of the number
- Left shift by `n-bits` is like multiplying the number by 2^n
- e.g.
- `5<<2` //it says that left shift `5` by `2-bits` as `5 = 0000 0101`
- `5 << 2` gives `0001 0100` and `0001 0100 = 20` also $5*2^2=20$

Understanding Bitwise Operators(cont…)

6. **Right Shift(>>)**
   - It pumps `n` no. of MSB(Most Significant Bit) on the left side of the bits of the number
   - i.e.
   - For Signed number it pumps `1's` (called Arithmetic Shift)
   - For Unsigned numbers it pumps `0's` (called Logical Shift)
   - Right shift by `n-bits` is like dividing the number by 2^n
   - e.g.
   - `5>>2 //`it says that right shift `5` by `2-bits` as `5 = 0000 0101` so `5 >> 2` gives `0000 0001 = 1` and also $5/2^2=1$

Some Important Concepts of Bitwise Operators

AND (&)	OR (|)	XOR (^)
0&x =0	0|x =x	0^x =x
-1&x =x	-1|x =-1	-1^x =x
x&x =x	x|x =x	x^x =0
x&x =0	x|x =1	x^x =-1

Applications of Bitwise Operators

1. **Swapping**
   - For now, we have done swapping of two numbers using

   i. A third variable like
      - `temp=x;`
      - `x=y;`
      - `y=temp;`
      - This technique requires an extra variable

   ii. Plus/Minus Operators like
      - `x=x+y;;`
      - `y=x-y;`
      - `x=x-y;`
      - This technique may cause overflow errors

Applications of Bitwise Operators(cont…)

iii. Multiplication/Division Operators like

- `x=x*y;;`
- `y=x/y;`
- `x=x/y;`
- This technique may not give good results when `y` is `zero`

- Here we are going to discuss a fourth way, i.e. using **XOR** operator
  - Let `x=5;     //x=0101`
  - `y=10; //1010`
  - `x=x^y;;    //x=1111`
  - `y=x^y;           //y=0101 = 5`
  - `x=x^y;           //x=1010 = 10`
  - You see a swap has occurred

Applications of Bitwise Operators(cont…)

2. Check Ranges of various Datatypes

- Bitwise operators can be used to check the ranges of different datatypes
- e.g.
- `unsigned char (8-bits)`
- `Min value = 0`
- `Max value =` 2^8-1
- **so as** 2^8 `= 1<<8`
- `int unsigned_MaxValue = (1<<8)-1;` /*`255` which is the Max value */

Applications of Bitwise Operators(cont…)

3. Checking Status of a bit

- For checking the status of a bit, we have to do two simple steps
- Step-1: Create a `mask`
- Step-2: `result = variable & mask`
- Mask contains `1` at the location of that particular bit and contains zeroes everywhere else
- e.g.
- `unsigned char ch=11;    //ch=0000 1011`
- `unsigned char mask=1<<2; /*mask=0000 0100` we are to check `bit# 2`*/
- `unsigned char result=ch & mask;       /*result=0` => bit is not set*/

Applications of Bitwise Operators(cont…)

4. Setting a bit

- Bitwise operators can also be used to set a particular bit by using a two step procedure
- Step-1: Create a `mask`
- Step-2: `result=variable | mask`
- e.g.
- `unsigned char ch=11; //ch=0000 1011`
- `unsigned char mask=1<<2;` /* `mask=0000 0100` we are to set `bit# 2` */
- `unsigned char result=ch | mask;` //`result=0000 1111` => bit has been set

Applications of Bitwise Operators(cont…)

- **Note:** You can also use the same logic to set a range of bits or to check if a range of bits is set or not

<u>SUMMARY</u>