Lecture 05

Operators and Expressions

CSE115: Computing Concepts

Expressions

- An **expression** is any valid set of literals, variables, operators, operands and expressions that evaluates to a single value.
- This value can be a number, a string or a logical value.
- For instance a = b + c; denotes an expression in which there are 3 operands a, b, c and two operator + and =.
- The number of operands of an operator is called its **arity**.
- Based on arity, operators are classified as **unary** (1 operand), **binary** (2 operands), **ternary** (3 operands).

Arithmetic Operators

Operation	Operator	Example (a=5,b=3)	Value of expression
Addition	+	`a+6`	11
Subtraction	-	`a-6`	-1
Multiplication	*	`a*b`	15
Division	/	`a/b`	1
Modulus	%	`a%b`	2

Arithmetic Operators

- If both of the operands are `int` types, then the result of the expression is an `int` too. If you want the result to be any other type, you have to use typecasting.

```
int a = 3, b = 5;
double c;
c = 1/5; //value of c is 0
c = 1.00/5; //value of c is 0.2
c = a/b; //value of c is 0
c = (double)a/b; //value of c is 0.6
```

- You can use modulus (%) operation **only on** `int` variables.

Unary Operators

C Operation	Operator	Example
Positive	+	`a = +3;`
Negative	-	`b = -a;`

- The first assigns positive `3` to `a`.
- The second assigns the negative value of `a` to `b`.

Operator Precedence

- When an expression contains more than one operator, the meaning of the expression may not be immediately clear:

  Does `i + j * k` mean `(i + j) * k` or
  `i + (j * k)` ?

- In C, this potential ambiguity is resolved by operator precedence.

- Precedence of the arithmetic operators:

Highest: `+ -` (unary)
`* / %`
Lowest: `+ -` (binary)

Operator Precedence

- Examples:

`i + j * k` means `i + (j * k)`
`-i * -j` means `(-i) * (-j)`
`+i + j / k` means `(+i) + (j / k)`

Associativity

- Operator precedence rules alone aren't enough when an expression contains two or more operators at the same level of precedence. The ***associativity*** of the operators now comes into play.
- The binary arithmetic operators are all left associative (they group from left to right); the unary operators are right associative.
- Examples of associativity:

```
i - j - k       means (i - j) - k
i * j / k       means (i * j) / k
i - j * i + k   means (i - (j * i)) + k
```

Assignment Operators

- Assignment operators are used to combine the '=' operator with one of the binary arithmetic operators
- In the following table, assume that c = 9

Operator	Example	Equivalent Statement	Value of c is
+=	`c += 7`	`c = c + 7`	16
-=	`c -= 8`	`c = c - 8`	1
*=	`c *= 10`	`c = c * 10`	90
/=	`c /= 5`	`c = c / 5`	1
%=	`c %= 5`	`c = c % 5`	4

Increment and Decrement Operators

- The ++ and -- operators increment and decrement variables.
- Both operators have the same precedence as negation.
- Either operator can be prefix or postfix:

```
++i (same as i = i + 1)
i++ (same as i = i + 1)
--i (same as i = i - 1)
i-- (same as i = i - 1)
```

- When used as a prefix operator, ++ increments the variable before its value is fetched:

```
i = 1;
printf("i is %d\n", ++i); /* prints "i is 2" */
```

Increment and Decrement Operators

- When used as a postfix operator, ++ increments the variable after its value is fetched:

```
i = 1;
printf("i is %d\n", i++); /* prints "i is 1" */
printf("i is %d\n", i); /* prints "i is 2" */
```

- The -- operator has similar properties:

```
 i = 1;
printf("i is %d\n", --i); /* prints "i is 0" */

i = 1;
printf("i is %d\n", i--); /* prints "i is 1" */
printf("i is %d\n", i); /* prints "i is 0" */
```

Increment and Decrement Operators

```
int R = 10,
count=10;
```

Statement	Equivalent Statements	R	Count
R = count++;	R = count; count = count + 1	10	11
R = ++count;	count = count + 1; R = count;	11	11
R = count --;	R = count; count = count – 1;	10	9
R = --count;	Count = count – 1; R = count;	9	9

Partial List of C Operators

Precedence	Operator	Associativity
3	* / %	Left
4	+ -	Left
5	= *= /= %= += -=	right

```
#include <stdio.h>
int main()
{
    int x = 3;
    printf("%d\n", 3 -2 / 4);//3
    printf("%f\n", 3 -2.0 / 4);//2.5
    printf("%d\n", -27 / -5 + 4 / 3);//6
    printf("%d\n", 16 % -5 + 7 * 6);//43
 printf("%d\n", -12*3%2*-23/+6-5*2);//-10

    printf("%d\n", x-- * 2 + 5);//11
    printf("%d\n", x);//2
    printf("%d\n", --x * 2 + 5);//7
    printf("%d\n",x);//1
    printf("%d\n", 3 % 5 / (5 % 3));//1
    return 0;
}
```

Bitwise Operators

&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	1's compliment
<<	Shift left
>>	Shift right

All these operators (except ~) can be suffixed with =
For instance a &= b; is the same as a = a & b;

Bitwise Operators

- Truth table

~a	a^b	a|b	a&b	b	a
1	0	0	0	0	0
1	1	1	0	1	0
0	1	1	0	0	1
0	0	1	1	1	1

Bitwise Operators

- Examples

```
    11010011
           &
    10001100
-------------
    10000000
```

```
    11010011
           |
    10001100
-------------
    11011111
```

```
    11010011
           ^
    10001100
-------------
    01011111
```

```
   ~11010011
-------------
    00101100
```

Bitwise Operators

- Examples: int a = 33333, b = -77777;

Expression	Representation	Value
a	**00000000 00000000 10000010 00110101**	**33333**
b	**11111111 11111110 11010000 00101111**	**-77777**
a & b	00000000 00000000 10000000 00100101	32805
a ^ b	11111111 11111110 01010010 00011010	-110054
a | b	11111111 11111110 11010010 00111111	-77249
~ (a | b)	00000000 00000001 00101101 11000000	77248
~ a & ~ b	00000000 00000001 00101101 11000000	77248

Bitwise Operators

- Examples

Right shift

```
 11010011>>3
------------
    00011010
```

Left shift

```
 11010011<<3
------------
    10011000
```

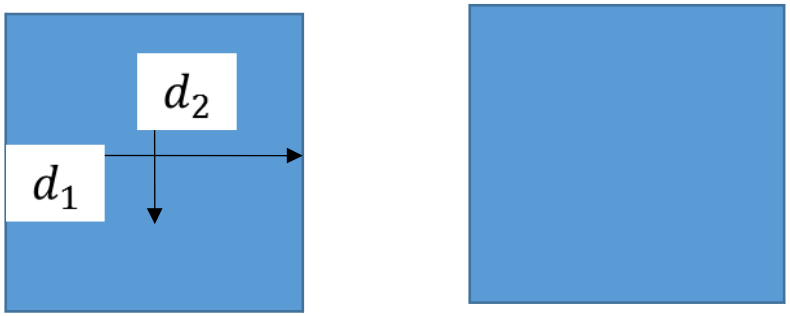
Bitwise Operators

```
char     c = 130;
int      a = 1 << 31;   /* shift 1 to the MSB */
unsigned int b = 1 << 31;
```

Expression	Representation	Action
c	10111100	unshifted
c << 4	1100**0000**	left shifted 4
c >> 4	0000**1011**	right shifted 4
a	10000000 00000000 00000000 00000000	unshifted
a >> 3	**111**10000 00000000 00000000 00000000	right shifted 3
b	10000000 00000000 00000000 00000000	unshifted
b >> 3	**000**10000 00000000 00000000 00000000	right shifted 3

Home-works

- Write a program that calculates the area and the perimeter of a rectangle where the length and the width of the rectangle are provided by the user as inputs.
- Write a program that calculates the area of a triangle where base and height of the triangle are provided by the user as inputs.
- Write a program that calculates area and perimeter of a circle where the radius of the circle is provided by the user as input.
- Write a program that calculates the volume and surface area of a flat washer where the outer diameter () and the inner diameter () are provided by the user as input.



- Write a program that accepts a character (small letter) as input from the user and (a) converts it into uppercase letter and (b) shows its binary equivalent. For example if the user input is ‘b’ then your output is ‘B’ and 01100010 (ASCII code of ‘b’ is 98 and binary equivalent of 98 is 01100010)