CSE 220 – C Programming

Expressions

Expressions

- Expressions: Formulas to compute a value
  - Variables
  - Constants
  - (a + b) * c
- Operators: tools to build expressions
  - Arithmetic: +, -, *, /
  - Relational for comparisons: >, <, >=, <=
  - Logical, assignment, …

Arithmetic Operators

- Unary: involves one operand
  - `i = +1; j = -1;`
- Binary: requires two operands
  - +: addition
  - -: subtraction
  - *: multiplication
  - /: division
  - %: remainder: 11 % 3 evaluates to 2

What do you think the following code outputs?

```
int a = -4;
int b = +a;
int c = -a;
printf("%d %d %d", a, b, c);
```

```
-4 4 -4
```

```
-4 +4 -4
```

```
-4 4 4
```

```
-4 -4 4
```

Arithmetic Operators

- +,-,*,/:
  - allow int and float operands
  - If both of same type: evaluates as given type
  - If mixed: evaluates as float
  - 1.0 / 2
  - 1 / 2
- %: both operands must be integers
- Cannot use 0 as right hand side of / and %

Operator Precedence

- Precedence rules:
  - 1: unary +, unary –
  - 2: *, /, %
  - 3: binary +, binary –

```
- a + b * c ⇔ (- a) + (b * c)
```

```
  i + -  j / y * x ⇔
i + ( (  (-j)  /  y )  *  x )
```

Assignment Operators

- Simple assignment: =

```
area = 5.5f;
j = 23 + i;
x = x + sqrt(a + b*pow(c, 3));
   /*pow defined in math.h */
```

*Evaluate sqrt(a + b*pow(c, 3)) => 16*
Add to 5 (the value of x) => 21
Store the result in x

memory

		a	b	c	i		area			j		x
		6	2	5	10		5.5			33		21

What is the value of f after the following statement?

```
float f = 5 /2;
```

Assignment Operators

- Compound assignment: uses old value of variable to compute its new value:

```
+=, -=, *=, /=, %=
```

```
height = height * 2;                        height *= 2;
weight = weight / 2;                        weight /= 2;
```

equivalent →

- Lvalue: an object stored in memory
- Assignment operators: modify left operand and require an lvalue as left operand

```
2 = 4; //Error. Can’t store 4 in 2. 2 is not an lvalue
```

Assignment Operators

- Increment/Decrement operators ++ and --:

```
j++;  similar to  j = j + 1;
c--;  similar to  c = c – 1;
```

- Postfix version:

```
int i = 0; printf(“%d”, i++);
//Print then increment
```

- Prefix version:

```
int i = 0; printf(“%d”, ++i);
//Increment then print
```

- Postfix operator have higher precedence
- Is this a valid statement? `++(++x);`

Expression Evaluation

- Expressions can be used as statements

```
i++;//Increments i

i+5;//Evaluates i + 5 and discards the result
```

- Expressions are evaluated according to precedence order of operators

```
a=b+=c++-d+--e/-f

a = b+=   (c++) – d + (--e) /  (-f)

a = b+=   (c++) – d + ((--e) /  (-f))

a = b+= ( (c++) – d + ((--e) /  (-f)))

a = (b+= ( (c++) – d + ((--e) /  (-f))))
```

Expression Evaluation

- C does not specify order of evaluation of subexpressions

```
(a - b)*(c + d) //Evaluate a-b or c+d first?
```

- Avoid expressions that use the value of a variable and modify it in the same expression:

```
 a = (b+= ( (c++) – d + ((--b) /  (-f))))
```

- Use multiple assignment statements instead:

```
x = (--b) /  (-f)
b += (c++) – d + x
a = b;
```

Expression Evaluation

- What is the value of c?

```
a = 5;
c = (b = a + 2) - (a = 1)
```

If (b = a + 2) is evaluated first:

b becomes: 5 + 2 = 7

a becomes: 1

c becomes: 7 – 1 = 6

If (a = 1) is evaluated first:

a becomes: 1

b becomes: 1 + 2 = 3

c becomes: 3 – 1 = 2

What do you think the following code outputs?

```
int i = 2;
int j = i * i++;
print("%d %d", i, j);
```

3 4

3 6

3 8

undefined

Equality Operators

- Equal to: ==
- Not equal to: !=
- Produce 0 or 1

```
int x = 5, y = 5, z;
z = x == y;    //z has value 1
z = x != y;    //z has value 0
y = 2;
Z = x != y     //z has value 1
```

Relational Operators

- <   <=   >=   >
- Produce 0 or 1
- 4 >= 4 has value 1
- 51 < 50 has value 0
- Warning!:

```
5 < 70 < 10
⬄ (5 < 70 ) < 10
⬄ 1 < 10
⬄ 1
```

Logical Operators

- Produce 0 or 1
- Negation: ! (unary):
  - `!expr` has value 1 if expr has value 0
- Logical and: &&
  - `expr1 && expr2`: 1 if both are non zero
  - `x > 1 && x < 10`
- Logical or: ||
  - `expr1 || expr2`: 1 if either is non zero

x	y	x && y	x || y
0	0	0	0
2	0	0	1
0	5	0	1
7	9	1	1

Bitwise Operators

- For bit manipulation:
  - Bitwise AND: &
  - Bitwise inclusive OR: |
  - Bitwise exclusive OR: ^
  - Bitwise complement: ~
  - Left shift: <<
  - Right shift: >>
- We'll be talking about this much more later in the course.
  - The rest of the lecture is a sneak peak at that material.

Bitwise Operators

- Binary representation:

```
int i = 22;          /*     10110 */
int j = 91;          /* 1011011 */
```

- Division by 2

  $22 = 2*11 = 2 * (2 * 5 + 1)$

  $= 2 * (2 * (2*2*1 + 1) + 1) = 2^4 + 2^2 + 2^1$

- Comparison by powers of 2 (1, 2, 4, 8, 16, 32, 64, …)

  $22 = 16 + 4 + 2 = 2^4 + 2^2 + 2^1$

  $91 = 64 + 16 + 8 + 2 + 1 = 2^6 + 2^4 + 2^3 + 2^1 + 2^0$

- In binary: $1111 = 10000 – 1 = 2^4 - 1$

Bitwise Operators

- Bitwise &:

  result = i & j;
  0000000000010110
  $\underline{0000000001011011}$
  0000000000010010 ⇔ $2^1 + 2^4 = 18$

- Bitwise ~ (complement):

  result = ~i;
  $\underline{0000000000010110}$
  1111111111101001 ⇔
  65,513

In this example:

- variables i and j are of type int
- are represented by 16 bits (2 bytes)

Bitwise Operators

- Bitwise exclusive or ^: *1 if bits are different, 0 if the same*

```
result = i ^ j;
0000000000010110
0000000001011011
0000000001001101 ⇔ 77
```

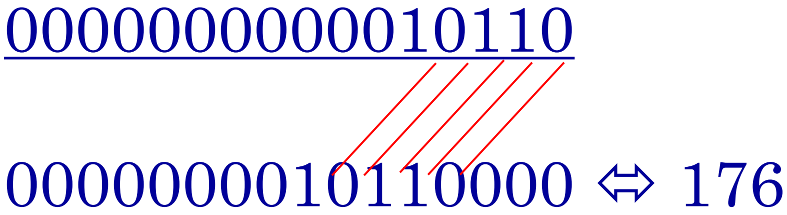
- Bitwise inclusive or |: *1 if at least one of the bits is 1*

```
result = i | j;
0000000000010110
0000000001011011
0000000001011111 ⇔ 95
```

Bitwise Operators

- Left shift:

```
result = i << 3;
```

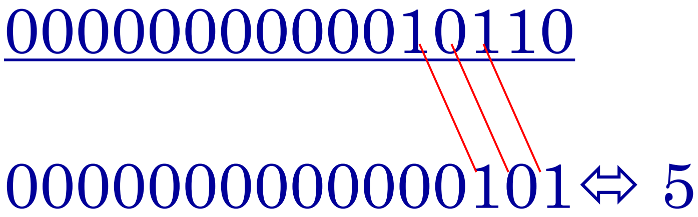


0000000000010110

0000000010110000 ⇔ 176

- Right shift:

```
result = i >> 2;
```



0000000000010110

0000000000000101⇔ 5

Summary

- Expressions
- Operators
- Operator precedence and expression evaluation
- Basic Types and type conversion