Computer Programming Lab 10

2023/12/12 Tim Chen

Contents

Bitwise Operators

Bitwise Operators

Operator	Description
&=	Bitwise AND assignment operator
|=	Bitwise inclusive OR assignment operator
^=	Bitwise exclusive OR assignment operator
<<=	Left-shift assignment operator
>>=	Right-shift assignment operator

Bitwise Operators (cont.)

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Example of bitwise operator

```c
// a = 5(00000101), b = 9(00001001)
int a = 5, b = 9;
printf("a & b = %d\n", a & b); // output = 1
```

Bitwise Operators (cont.)

Process of `a & b`

```
      0 0 0 0 0 1 0 1 <- a
and)  0 0 0 0 1 0 0 1 <- b
----------------------
      0 0 0 0 0 0 0 1
```

Bitwise Operators (cont.)

Example of left shift

```c
int three = 3;
printf("three = %d\n", three); // 3
three <<= 1; // Left shift one bit
printf("After left shift three = %d\n", three); // 6
```

```
            0 0 1 1 <- 3
left shift) 0 1 1 0 <- 6
```

Bitwise Operators (cont.)

Bitwise NOT (one's complement) has no assignment operator

```c
int main(void) {
  unsigned char cc = 0X0F;
  printf("NOT cc = %c\n", ~cc); // -16
}
```

1. `cc` = `0X0F` = `00001111`
2. `~cc` = `0XF0` = `11110000` $\longleftarrow -16$

ASCII code

ASCII code

DEC	HEX	BIN	SYMBOL
48	30	00110000	0
65	41	01000001	A
97	61	01100001	a
58	3A	00111010	:
55	37	00110111	7
50	32	00110010	2

ASCII code (cont.)

```
unsigned char colon = ':';
unsigned char seven = '7';
printf("':' & '7' = %c\n", colon & seven);
```

Output:

```
':' & '7' = 2
```

ASCII code (cont.)

Process of `colon & seven`

```
      0 0 1 1 1 0 1 0  <- colon
and)  0 0 1 1 0 1 1 1  <- seven
----------------------
      0 0 1 1 0 0 1 0  <- 2
```

Quick Sort

Quick Sort

```c
void qsort(
  void *base, size_t nitems, size_t size,
        int (*compar)(const void *, const void*)
)
```

Quick Sort (cont.)

```c
int compar(const void * a, const void * b) {
  int l = *(const char *)a; // a is `const char *` now
  int r = *(const char *)b; // r is *b
  /* function return type
   * negative, if first < second
   * 0,        if first equal second
   * positive, if first > second
   */
}
```

Quick Sort (cont.)

```c
int compar(const void * a, const void * b) {
  // char - char
  return *(const char *)a - *(const char *)b;
}
```

Quick Sort (cont.)

```c
int compar(const void * a, const void * b);

int main(void) {
  char array[] = { 'a', 'c', 'b', 'd' };
  qsort(array, 4, sizeof(char), compar);
  /*
   * Before sort:  a c b d
   * After sort:   a b c d
   */
}
```

Homework 10: Number of Islands

Homework 10: Number of Islands

I/O format:

Input:

- An `m * n` 2D binary grid.
- `m` is the length of the `grid`
- `n` is the length of `grid[i]`
- `grid[i][j]` is either `0` or `1`

Output:

- Print out the number of islands

Homework 10: Number of Islands (cont.)

Input sample:

```
1 1 1 1 0
1 1 0 1 0
1 1 0 0 0
0 0 0 0 0
```

Output sample:

```
1
```

Exercise 10: Syntax Tree

Exercise 10: Syntax Tree

Description:

- Convert a expression to a abstract syntax tree(AST).
- The expression only has `+`, `-`, `*`, `/`, `(`, `)` and variables (a to z and A to Z).
- Ensure that each operator will be enclosed in parentheses regardless of the four fundamental operations of arithmetic.

Exercise 10: Syntax Tree (cont.)

I/O Format:

Input:

- Input has one line.
- First line has a expression of length $L\left(1 \leq L \leq 2 \cdot 10^5\right)$.

Output:

- Output the preorder of the AST.
- There is no `\n` at the end of each character.

Exercise 10: Syntax Tree (cont.)

Input sample:

```
((a+b)*(c+d))
```

Output sample:

```
*+ab+cd
```

Exercise 10: Syntax Tree (cont.)

Input sample:

```
(((a+b)*(c+d))+e)
```

Output sample:

```
+*+ab+cde
```

Appendix A

Tips 1

`diff` Your Standard Output

```
./a.out < 1.in > my.out
vimdiff my.out 1.out # or `diff my.out 1.out`
```

```
# The hyphen tells `diff` to use std input
./a.out < 1.in | diff 1.out -
```

From `man diff`:

> `If either file1 or file2 is ‘-’, the standard input is used in its place.`

Tips 2

Using Shell Script

```
gcc main.c                   # Compile
./a.out < 1.in > my.out      # Execute
diff 1.out my.out            # Diff
```

```
gcc main.c                         # Compile
./a.out < 1.in | diff 1.out -      # Execute then Diff
```

```
gcc main.c && ./a.out < 1.in | diff 1.out - # All in one
```

Tips 2 (cont.)

Using Shell Script (cont.)

First, put the command into `compile.sh`

```
echo "(g++ ./syntax_tree.cc) && ./a.out < 1.in | diff
1.out -" > compile.sh
```

and check

```
$ cat compile.sh
(g++ ./syntax_tree.cc) && ./a.out < 1.in | diff 1.out -
```

Tips 2 (cont.)

Using Shell Script (cont.)

Last, run it via `sh compile.sh`

```
sh compile.sh
```

Appendix B

Some Links

- Neovim - https://neovim.io
- LeetCode - https://leetcode.com
- My dotfiles - https://github.com/gnitoahc/.dotfiles
- My email - chaotingchen10@gmail.com
- Simple Syntax Tree - https://gnitoahc.github.io/simple-syntax-tree

Thank you