COMP-SCI 201L (FS17) - **Section 2**

Lab 11 - 11/28/17

POWERBALL - Claim Prizes using Bit Manipulation

Bit manipulation is the act of algorithmically manipulating bits. Computer programming tasks that require bit manipulation include low-level device control, error detection and correction, data compression, encryption, and so on. Admittedly, modern programming languages allow the programmer to work directly with abstractions instead of bits that represent those abstractions. Nevertheless, the skill to well use bitwise operators is still crucial for every programmer, since bitwise operation has many great advantages. For example, bitwise operation is very fast, since all the bits are processed in parallel, which can dramatically drop the time complexity for certain jobs. Also, if we pack different information in different bits of one data, then when communication is needed, we can just send out one data with all information included, instead of multiple data, which significantly accelerates the communication amongst different machines. Programming questions that include bit manipulation still have high frequency in today's industrial job interview.

Bitwise Operators

Today, we are going to learn 5 common bitwise operators in C++, bitwise and (**&**), bitwise or (**|**), bitwise exclusive or (**^**), bitwise left shift (**<<**), and bitwise right shift (**>>**). Table 1 gives examples of how those operators work, and Table 2 shows the precedence of C++ common operators in compound expression.

Table 1. Five common C++ bitwise operators.

Name	In C++	Example	Application/Trick

Bitwise **and**	**&**	`5 & 3 = 1`
		`0101`
		`& 0011`
		`--------`
		`0001`
			• Determine odd/even.
			• Check if n is power of 2.
			• Extract certain bit(s).
			• Turn off* n's rightmost on* bit: $n \,\&\, (n-1)$.

Bitwise **or**	**|**	`5 | 3 = 7`
		`0101`
		`| 0011`
		`--------`
		`0111`
			• Set certain bit(s) on.
			• Determine odd/even.
			• Set binary value(s) (0 or 1) to certain bit(s) of a number.
			• Equality: $n \mid 0 = n$

Bitwise **exclusive or**	**^**	`5 ^ 3 = 6`
		`0101`
		`^ 0011`
		`--------`
		`0110`
			• Important equalities:
			$0 \wedge n = n$
			$a \wedge b \wedge b = a$
			$a \wedge b \wedge c \wedge b = a \wedge c$
			• In-place swap 2 numbers.

Bitwise **left shift**	**<<**	`5 << 1 = 10` `0101 → 1010`	• Multiply by 2. • Left truncation.

Bitwise **right shift**	**>>**	`5 >> 2 = 1`
		`0101 → 0010`
		`→ 0001`
			• Divide by 2.
			• Average of 2 integers.
			• Right truncation.

*If a bit is 1, we say the bit is **on**; if a bit is 0, we say the bit is **off**.

Table 2. Operator precedence table.

Operator	Precedence
! (logical not)	3
***** (multiplication) **/** (division) **%** (modulus)	5
+ (addition) **-** (subtraction)	6
<< (bitwise left shift) **>>** (bitwise right shift)	7
< (less than) or **<=** **>** (greater than) or **>=**	8
== (logical equal) or **!=**	9
& (bitwise and)	10
^ (bitwise exclusive or)	11
| (bitwise or)	12
&& (logical and)	13
|| (logical or)	14
? : (conditional)	15
= (assignment) **+= -= *= /= %=** **&= |= ^=** **<<= >>=**	16

How All Powerball Numbers are Stored in One 64-Bit Unsigned Integer?

In each game, players select 5 numbers from a set of 69 white balls (numbered 1 to 69) and one number from 26 red balls (numbered 1 to 26). The red ball number can be the same as one of the white balls. **The order of selection does not matter**. The numbers in each ticket is **not** guaranteed to be sorted. Therefore, you may need to sort the numbers.

In this Lab, we are using a C++ 64-bit unsigned integer (the **`uint64_t`** type) to store all the information of user selected numbers, which is illustrated in Figure 3 (left). We use 8 bits to represent a selected ball. The first bit represents the color of the ball (white or red) and the rest 7 bits represent the number of the ball (1 to 69 for white ball and 1 to 26 for red ball). Finally, the rest 16 bits are used to represent the user ID. The prize rule is illustrated in Figure 3 (right). O means **a** white ball match and ● means **the** red ball match.

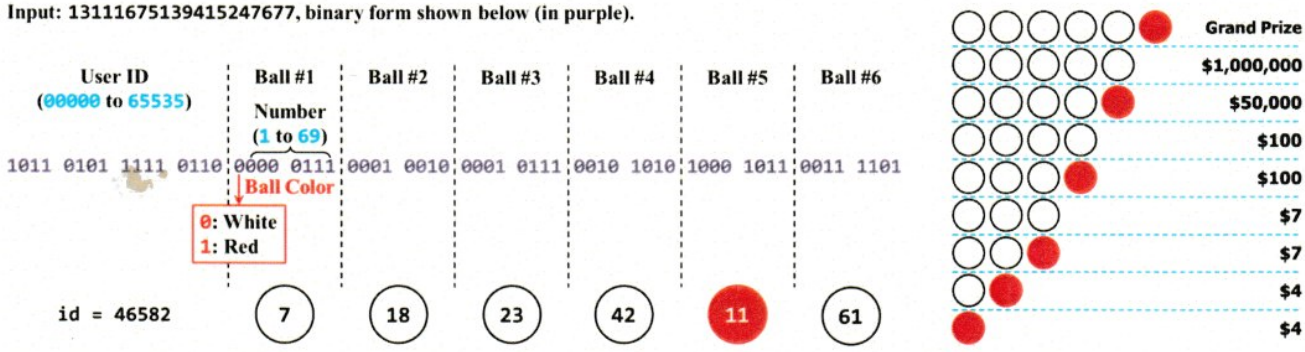

Figure 3. (Left) Explanation of how different values are packed in one 64-bit unsigned integer. (Right) Prize rule. O means **a** white ball match and ● means **the** red ball match.

Lab Instruction

1. Start a new, empty visual C++ project in MS Visual Studio.
2. Download "`Ticket.h`", "`Ticket.cpp`" and "`Lab_11.cpp`" from Blackboard. Add these files to your project.
3. In `Ticket.h`, write a **`class`** called **`Ticket`**. It contains 3 `private` variables: an unsigned integer named **`id`** that stores the user id, a `vector` (of size 5) of unsigned integers named **`whiteballs`** that stores the user selected white ball numbers, and an unsigned integer named **`redball`** that stores the user selected red ball number.
4. Write a constructor of `Ticket` that passes a 64-bit unsigned integer (`uint64_t` type). The constructor extracts the user id, the five white ball numbers and the red ball number from the 64-bit unsigned integer passed in, using bitwise operations, then set the `private` variables to those extracted values.
5. Write a **`const`** function called **`Prize()`** which passes 3 variables: **`white`** (`const vector<unsigned int>&` type, the winning numbers of white balls), **`red`** (`const unsigned int&` type, the winning number of red ball), and **`Grand_Prize`** (`const unsigned int&` type, the current grand prize). The function calculates the prize of the user (in USD), based on the comparison between the user selected numbers and the winning numbers. The prize rule is illustrated in Figure 3 (right).
6. Overload the "`>>`" (input stream) and "`<<`" (output stream) operators for `class Ticket`. For each `Ticket`, output the user id **only**. Because the `Prize()` function relies on the current winning numbers, we are going to output the prize for the user in our `main()` program. **Please note that the user id is 5 characters long, so if the value is less than `10000`, you need to add leading zeros in front of it in the output.** Sample input and output are provided for your reference.
7. In `Lab_11.cpp`, the winning numbers and grand prize are provided as global constants. In your `main()` program, you need to read several 64-bit unsigned integers from the input file, parse the packed information into `Ticket`(s), and output prize for each `Ticket`.

Submission and Grading

1. Submit your work **no later than Thursday, Nov 30th, 2017, by 11:59 PM**. Late submission will **never** be accepted. Since it is the last Lab assignment and we are finishing up the semester. **No extension request will be granted.** Thank you for your understanding.
2. Submit 3 files: "`Ticket.h`", "`Ticket.cpp`" and "`Lab_11.cpp`" (**unzipped**).
3. Your code will be graded using another input file (stilled named `input.txt`) to see if it can generate the expected correct output file. Also, the lab grader will look at your code manually to see if the code quality meets our other requirements.
4. **All your Lab grades will be posted on Blackboard by Monday, Dec 4th, 2017. A sample exam 2 (with solution) will be available on Blackboard by Thursday Nov 30th, 2017.**