Bloomberg Lugano

Interview 1	*2015 May 20*	*22:15 – 23:15*	*using HackerRank*
Interview 2	*2015*		
Interview 3	*2015*		

Question 1

```
int main(int argc, char *argv[])
{
    char    abc[27];
    char    *ptr = abc;
    strcpy(abc, "abcdefgxyz");
    /* What are the types and values of expressions:
     *
     * 1. abc                   type = char*           value = ???        &abc[0]
     * 2. *abc                  type = char            value = 'a'
     * 3. abc[2]                type = char            value = 'c'
     * 4. &abc[3]               type = char*           value = ???
     * 5. abc+4                 type = char*           value = 'e'
     * 6. *(abc+5) + 2          type = char            value = 'f'+2 = 'h'
     * 7. abc[10]               type = char            value = '\0'
     * 8. abc[12]               type = char            value = undefined
     * 9. &ptr,                 type = char**          value = ???
     */
    return 0;
}
```

Question : Explain the difference between malloc and new. Demostrate.
Answer : Both use heap memory, constructor is called in new.

```
int *p0 = malloc(sizeof(int) * 10);
int *p1 = new int[number_of_object];
```

Question : Is there another way to allocate an object?
Answer : By declaring stack variable.

```
int i; // stack memory
```

Question : Explain the difference between stack variable and heap variable.
Answer : Stack variable can be accessed faster. Its lifespan is the same as the function scope. Heap memory allows dynamic allocation (i.e. runtime allocation).

Question : What happen if new is called many time (assume each call allocates a little only)?
Answer : Memory fragmentation. Result in slow allocation.

Question 2

```
#include <vector>
void foo()
{
    std::vector<int> v;
    v.reserve(10);
    for(int i=0; i<1000000; ++i) v.push_back(i);
}
```

Question : Please calculate the number of reallocation.
Answer : Allocation happens when the size of v reaches : 2^4, 2^5, ..., hence the answer is N-3, where

$$\begin{aligned}
2^N &= 1{,}000{,}000 \\
N &= \text{floor}(\log_2(1{,}000{,}000)) \\
\text{ans} &= \text{floor}(\log_2(1{,}000{,}000)) - 3 \\
&\sim \text{floor}(\log_2(1024 \times 1024)) - 3 \\
&= \text{floor}(\log_2(2^{10} \times 2^{10})) - 3 \\
&= \text{floor}(\log_2(2^{20})) - 3 \\
&= 20 - 3 \\
&= 17
\end{aligned}$$

Question : What are 2^{16} and 2^{32}? Please calculate by hand and give approximation only.
Answer : We should make use of 2^{10} = 1024 = 1K.

2^16 = 256^2 = 65536
2^32 = 65536^2 = 1000,000,000 * 4=   4G

Question 3
Write a function that multiplies an integer by 31 without using *.

```
int mult31(int n)
{
   return (n << 5 - n); // shifting to LHS by 5 bits
}
```

Question : How does mutex lock work?
Answer : It is just an **atomic bistate variable**, with 2 states : unlocked and locked. Racing condition happens when multithreads share common resources. Common resouces should be protected by mutex in order to ensure only one thread is accessing / modifying at one time.

Question : What is deadlock?
Answer : It happens **(1) when we introduce lock in the resources and (2) when concurrency involves locking two or more resources**, threads may block each other forever when they keep waiting for resources being locked by the others **(it forms a cycle in graph, with vertex representing thread, edge representing common resource between two threads)**. A classical example is the dining philosophers problem, each one sitting on the round table requires two forks to eat spaghetti, they may get stuck when some or all of them holds a fork. Another example is multi process making transaction among a pool of accounts, each transaction involves two accounts (FROM and TO), which should be locked before transaction is made, there is occasion that :

Process 1	transact from account A to account B	now locked A and trying to lock B
Process 2	transact from account B to account C	now locked B and trying to lock C
Process 3	transact from account C to account A	now locked C and trying to lock A

Question : How to avoid deadlock?
Answer (1) : Lock ordering - resources are assigned with an ID, when a thread need to acquire more than one resources, it should lock the resource in predefined order (i.e. either ascending or descending), and optionally, the thread should release locked resources if it cannot acquire all the lock it needs to start its job.

Answer (2) : Lock timeout
Answer (3) : Lock detection (need to build a graph and perform cycle detection)
Answer (4) : Please avoid self deadlock with recursive mutex
Answer (5) : Please google Banker's algorithm.

Answer (6) : Implement an arbitrator who centralizes all resources management. There is a mutex in the arbitrator, i.e. only one thread is served by arbitrator at a time. The thread needs to tell the arbitrator what resources it needs, the request is either accepted (if arbitrator finds all resources it needs are available) or rejected (if arbitrator finds that one or more resources it needs is not available).

Question : What is instruction cache?
Answer : CPU cache is a special block of memory for fast and frequent access, it is a copy of frequently used memory in the system memory. Cache is faster than the system memory. CPU cache includes instruction cache (for instruction fetch) and data cache (for data fetch). In C++, there is no way to force the CPU to put certain variable into cache (does keyword `register` help?).

Question : What is the computational load of sorting in STL containers?
Answer : nlog(n)

Question : What is the most efficient sorting algorithm if we want to sort std::vector<char>?
Answer : Pigeon hole sort.

Question 4

```
class A
{
public:        A()              { cout << "a";        }
               A(int x):X(x)    { cout << "b";        }
              ~A()              { cout << "c" << X;   }
private:      int X;                                  };

class B : public A
{
public:        B()              { cout << "d";        }
               B(int x):X(x)    { cout << "e" << x;   }
              ~B()              { cout << "f" << X;   }
private:      int X;                                  };

class Base
{
public:        Base():a(5)      { cout << "g";        }
              ~Base()           { cout << "h";        }
private:      A a;                                    };

class Derived : public Base
{
public:        Derived():b(99)  { cout << "i";        }
              ~Derived()        { cout << "j";        }
private:      B b;                                    };

int main(int argc, char* argv[])
{
     { Derived d; }
     cout << "#";
     return 0;
}
```

Question : What will be printed?

Answer : b g a e i j f c h c #   correct

~~b g a e i # j f c h c   incorrect~~

Extra questions from web : Interview with Bloomberg

- How to reverse an integer?
- How to detect two linked list intersect with each other?
- When to use binary search tree? When to use hash table?

For operations : `insert`, `erase` and `find`, it is always O(log(N)) for BST, O(1) for hash without collision and O(N) for hash with collision. BST allows bidirectional iterator (hence given an iterator, we can easily find its neighbours), hash allows forward iterator only, which does not iterate in an ordered way. Besides, BST allows efficient method to find all keys lying with a range.

Extra questions from Goldman Sachs

Q1 : How does virtual function table work? — For class having virtual function

Q2 : How does virtual table pointer work? — For object pointing to virtual fct table

Q3 : How do we handle big number problem (for floating point)?

Q4 : What is the validity of STL's iterator?