**Flow Trader - C++ Training**

*2017Dec*

About string

1. Given a string containing multiple brackets, check if brackets are completed.

```
bool is_bracket_completed(const std::string& str)
{
	std::stack<char> stack;
	for(auto iter=str.begin(); iter!=str.end(); ++iter)
	{
		if (*iter=='(' || *iter=='[' || *iter=='{')
		{
			stack.push(*iter);
		}
		else if (*iter==')' || *iter==']' || *iter=='}')
		{
			if	(*iter==')')	{ if (stack.top()=='(') stack.pop(); else return false; }
			else if (*iter==']')	{ if (stack.top()=='[') stack.pop(); else return false; }
			else			{ if (stack.top()=='{') stack.pop(); else return false; }
		}
	}
	return stack.empty();
}
```

2. Two strings are anagram if they are consisted of the same set of characters in different orders.

```
bool is_anagram(const std::string& str0, const std::string& str1)
{
	if (str0.size() != str1.size()) return false;

	std::string sorted_str0 = str0;
	std::string sorted_str1 = str1;
	std::sort(sorted_str0.begin(), sorted_str0.end());
	std::sort(sorted_str1.begin(), sorted_str1.end());

	auto iter0=sorted_str0.begin();
	auto iter1=sorted_str1.begin();
	for(; iter0!=sorted_str0.end(); ++iter0,++iter1)
	{
		if (*iter0!=*iter1) return false;
	}
	return true;
}
```

3. Find all repeated words in a string.

```
void repeated_words(const std::string& str, std::vector<std::string>& words)
{
	std::unordered_map<std::string, int> histogram;
	size_t pos = 0;
	while(pos < str.size())
	{
		size_t found_pos = str.find(" ", pos);
		std::string found_word;

		if (found_pos != std::string::npos)
		{
			found_word = str.substr(pos, found_pos-pos);
			pos = found_pos + 1;
		}
		else
		{
			found_word = str.substr(pos);
			pos = str.size();
		}

		for(auto& x : found_word) x = tolower(x);
		auto iter = histogram.find(found_word);
		if (iter != histogram.end()) ++(iter->second);
		else histogram[found_word] = 1;
	}

	words.clear();
	std::cout << "\nhistogram";
	for(auto i=histogram.begin(); i!=histogram.end(); ++i)
	{
		std::cout << "\n" << i->first << " = " << i->second;
		if (i->second > 1) words.push_back(i->first);
	}
}
```

## About integer array

1. Generate a new integer by reversing bits. *[Modified question from Facebook : count the number of 1-bits]*

```
int bit_reverse(int x)
{
	int y = x;
	int z = 0;
	while(y > 0)
	{
		if (y%2 == 0) z = z*2;
		else          z = z*2+1;
		y >>= 1;      // >> means right shift, >>= means right shift and assign.
	}
	return z;
}
```

```
int count_1bit(int x)
{
	int count = 0;
	while(x>0)
	{	count += x%2;
		x = x/2;
	}
	return count;
}
```

```
int count_1bit(int x) // 2 bits per iteration, less iterations
{	int LUT[] = {0,1,1,2};
	int count = 0;
	while(x>0)
	{	count += LUT[x%4];
		x = x/4;
	}
	return count;
}
```

2. Find integer pairs differ by *k* in *O(N)*. Don't implement $O(N^2)$ exhaustive search nor *O(NlogN)* inplace sorting.

```
void kdifference(const std::vector<int>& input, int k, std::vector<std::pair<int,int> >& output)
{
	std::unordered_set<int> hash(input.begin(), input.end());
	for(auto iter=hash.begin(); iter!=hash.end(); ++iter)
	{
		auto ans_iter = hash.find((*iter)+k);
		if (ans_iter != hash.end())
		{
			output.push_back(std::make_pair(*iter, *ans_iter));
		}
	}
} // This implementation returns (x,y) and (y,x) in output. For better implementation, see alg3.doc.
```

3. Implement a custom sorting which places even / odd numbers of LHS / RHS respectively.

```
void inplace_even_odd_sorting(std::vector<int>& input)
{
	auto iter0 = input.begin();
	auto iter1 = input.end();
	--iter1;
	while(iter0!=iter1)
	{
		if ((*iter0)%2 == 1)
		{ //  std::swap(iter0, iter1); // BUG : not desired swapping effect
			std::swap(*iter0, *iter1);
			--iter1;
		}
		else ++iter0;
	}
}	// This is similar to CTI question : "Remove space from string", remember to use 2 pointers.
```

## Dynamic programing

1. Find all subvectors recursively.

```
void subvector(const std::vector<int>::const_iterator& begin,
		const std::vector<int>::const_iterator& end,
			std::unordered_set<std::vector<int>>& output)
{
	if (begin == end) return;

	// *** Recursive implementation : find all subvectors starting with ... *** //
	subvector(begin+1, end, output);
	auto iter0 = begin;
	auto iter1 = end;
	while(iter0!=iter1)
	{
		output.insert(std::vector<int>(iter0,iter1));
		--iter1;
	}
}
```