Homework #5

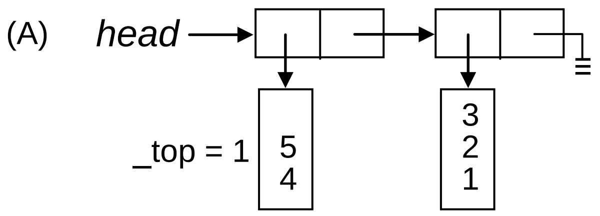
Demo date: 5/10

Stack and backtracking

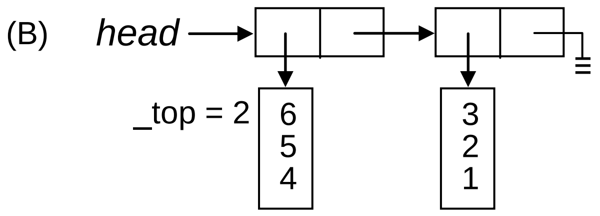
Part A – Stack implementation

A stack may be implemented as a hybrid of arrays and linked lists. For the purpose of this homework, you are asked to implement a stack as a singly linked list in which each node contains an array of n objects of type T for some integer n.

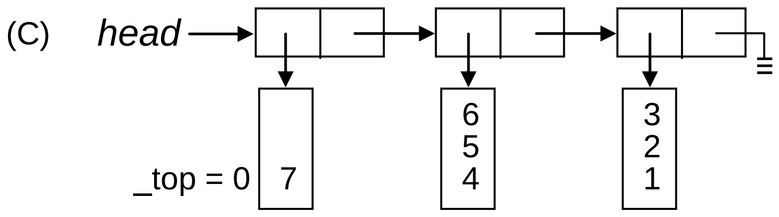
For illustration purpose, let's assume that $n = 3$. After pushing integers 1 to 5, in that order, onto an integer stack. The singly linked list representing the stack shall look like



Pushing 6 onto (A) yields



Pushing 7 onto (B) yields



Popping 7 off (C) restores (B), and popping 6 off (B) restores (A).

Here is the required class template:

```
template<typename T>
class stack {
public:
// types
   typedef size_t size_type;
   typedef T value_type;
// ctors and dtor
   stack(int=2);   // array size; default is 2 for testing purpose
   stack(const stack<T>&);   // copy ctor
   ~stack();
// stack operations
   void push(const value_type&);
   void pop();
   value_type& top();
   const value_type& top() const;
   size_type size() const;
   bool empty() const;
private:
   struct node;
   node* head;             // pointer to the header node
   int _top;               // array index for stack top
   const size_type sz;     // array size
   size_type _size;        // # of elements in the stack
};

template<typename T>
struct stack<T>::node {
   node(T*,node*);         // ctor
   T* stk;
   node* succ;
};
```

Requirements

1 Define all member functions outside the class body.

2 Hints on storage management

Form (B) to (C)

Since there is no available array space in the stack, the **`push`** operation shall allocate a new header node that contains a pointer to a newly allocated array.

Form (C) to (B)

After popping off the stack top, the array contained in the header node becomes empty. Thus, the **`pop`** operation shall deallocate the header node as well as the array contained in it.

3 Important note (See Lecture on Class and ADT, pp30~31)

– For array management
DO NOT use **`new`** and **`delete`** operators.
Instead, use **`operator new[]`** and **`operator delete[]`**.

– For **`push`**
DO NOT use assignment. Instead, use placement new.

– For **`pop`**
Call **`T`**'s dtor to destroy the element being popped off.

How to define the following copy ctor?

```
template<typename T>
stack<T>::stack(const stack<T>& rhs);
```

Method A – Required

Write a loop to make a deep copy of the source **`stack`** object **`rhs`**, as we did in Lecture on Class and ADT, p49.

Everyone has to write this version. The resulting program may be tested on VC++, as usual.

Method B – Optional for 50% bonus

This part employs two new features of C++11, namely,

- delegating ctors, and
- default values of non-static data members

Since VC++ doesn't yet support them, you shall test this part on GNU C++ installed on our department's FreeBSD workstations, e.g.

```
bsd2> g++47 -std=c++11 hw5.cpp
bad2> ./a.out
```

So, you have to learn the basics of FreeBSD workstations – this is why this part is worth 50 bonus points.

Step 1: Initialize **`*this`** as an empty stack

The delegating ctor provides a convenient way to do this, as follows.

```
template<typename T>
stack<T>::stack(const stack<T>& rhs)
: stack(rhs.sz)        // initialize *this
{
     // call the function in step 2
}
```

Step 2: Recursively push the elements of **`rhs`** onto **`*this`**.

To this end, you need the following private function to push the elements of **`rhs`** pointed to by **`p`** onto **`*this`** recursively

```
template<typename T>
void stack<T>::copy(node* p);
```

Although not really necessary for Method B, you are nonetheless asked to give the data member **`head`** a default value **`nullptr`** and use it in the ctor.

```
template<typename T>
class stack {
     // other members omitted
     node* head=nullptr;      // default value of head
}
```

Part B – Stack application: backtracking

In this part, you are asked to solve the set partition problem (Problem 2008-19 of Collegiate Programming Exam) described below by backtracking. (See Lecture on Class and ADT, p12).

Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers, count the number of subsets $A' \subset A$ such that

$$\sum_{a \in A'} a = \sum_{a \in A - A'} a$$

Also, generate all such subsets.

You shall write the following function that displays all the subsets and returns the # of subsets as the function value.

```
int subset(int* a,int n);
```

Algorithm

Let $b = \sum_{a_i \in A} a_i$

If b is odd, then no such subset exists.
If b is even, this problem may be solved as follows.
For $1 \le i \le n,\ j \le b/2$, define

$t[i,j] =$ the number of subsets of $\{a_1, a_2, \ldots, a_i\}$ for each of which the sum of the elements is exactly j

as

$$\begin{aligned} t[1,j] &= 1, && \text{if } \ j = 0 && (\text{i.e. the subset is } \emptyset) \\ & && \text{or } \ j = a_1 && (\text{i.e. the subset is } \{a_1\}) \\ &= 0, && \text{otherwise} && (\text{i.e. } j < 0 \text{ or } j > 0 \text{ but } j \ne a_1) \end{aligned}$$

$$t[i,j] = t[i-1, j-a_i] + t[i-1, j], \qquad \text{if } i > 1$$

$$(\text{i.e. either } a_i \in \text{the subset or } a_i \notin \text{the subset})$$

The value we want is $t[n, b/2]$.

To solve it by backtracking, two stacks are necessary. For testing purpose, one of them should not use the default array size:

```
stack<tuple<int,int> > b;      // for backtrack points
stack<int> s(3);               // for subsets
                               // use 3-element array, say
```

How do backtracking and subset enumeration work?

1 On solving $t[i,j],\ i>1$

Repeat the following steps until $i=1$

1) push the backtrack point **`tuple<int,int>`**$(i-1,j)$ onto stack *b*
2) push the element a_i onto stack *s*
3) decrease i and j

2 On solving $t[1,j]$

1) if $j=0$, output the contents of stack *s*
   else if $j=a_1$, output a_1 and the contents of stack *s*
2) backtrack

3 On backtracking, pop one element off stack *b* and one element off stack *s*

How to output the contents of stack *s*?

The **`stack`** class doesn't provide any public method for displaying the contents of a stack. For the purpose of this homework, you are asked to use the following slow method:

1 **`stack<int> t(s);`** // copy stack *s* to stack *t*
2 Output the contents of stack *t* by popping off all of its elements

File organization requirement

Your program shall be organized in two files.

1 Implementation file (say, stack5.h)
This file contains the implementation of the **`stack`** class.

2 Application file (say, hw5.cpp)
This file includes stack5.h and contains the stack application.

Remark

C++ provides two template compilation models:
1 Inclusion compilation model
2 Separation compilation model

Most C++ compilers only support the inclusion compilation model: the definitions of inline and non-inline member functions and static data members are all placed in the header file.

Under the inclusion compilation model, it is the duty of the compiler to guarantee that the one-definition rule be satisfied.

For simplicity, let's illustrate the two models using function template.

Inclusion compilation model

File A.cpp
```
#include "C.h"
void p();
int main()
{
   f<2>();
}
```

File B.cpp
```
#include "C.h"
void p()
{
   f<2>();
   f<3>();
}
```

File C.h
```
template<int n>
int f()
{
   return n;
}
```

Separation compilation model

File A.cpp
```
#include "C.h"
void p();
int main()
{
   f<2>();
}
```

File B.cpp
```
#include "C.h"
void p()
{
   f<2>();
   f<3>();
}
```

File C.h
```
template<int n>
int f();
```

File C.cpp
```
export template<int n> int f() { return n; }
```

Sample test

Suffice it to run the sample test given in file hw5.cpp, together with the implementation file stack5.h of the **`stack`** class template.

Sample output

```
Test 1...
1 6 7
2 5 7
3 4 7
1 2 4 7
3 5 6
1 2 5 6
1 3 4 6
2 3 4 5
8 subset(s) in total

Test 2...
3 7 8
1 2 7 8
4 6 8
1 3 6 8
1 4 5 8
2 3 5 8
1 2 3 4 8
5 6 7
1 4 6 7
2 3 6 7
2 4 5 7
1 2 3 5 7
3 4 5 6
1 2 4 5 6
14 subset(s) in total

Test 3...
0 subset(s) in total
```