

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

Implementation Issues

g++ compiler directive -D

defining our own exception

Implementing Stacks on our own

our own vector

our own single linked list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

MCS 360 Lecture 14
Introduction to Data Structures
Jan Vershelde, 24 September 2010

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack
with the STL vector

implementing a stack
with the STL list

Implementation Issues

g++ compiler
directive -D

defining our own
exception

Implementing Stacks on our own

our own vector

our own single linked
list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

In the STL, a stack is an *adapter*:
either a vector or a list implements a stack.

A dictionary between STL stack, vector, and list,
for any item t of type T :

<code>stack<T> s</code>	<code>vector<T> v</code>	<code>list<T> L</code>
<code>s.push(t)</code>	<code>v.push_back(t)</code>	<code>L.push_back(t)</code>
<code>if(!s.empty())</code>	<code>if(!v.empty())</code>	<code>if(!L.empty())</code>
<code> t = s.top()</code>	<code> t = v.back()</code>	<code> t = L.back()</code>
<code> s.pop()</code>	<code> v.pop_back()</code>	<code> L.pop_back()</code>

An online Standard Template Library Programmer's Guide is
at <http://www.sgi.com/tech/stl/>

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

Implementation Issues

g++ compiler

directive -D

defining our own exception

Implementing Stacks on our own

our own vector

our own single linked list

list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

stack uses STL vector

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#ifndef MCS360_STACK_STL_VECTOR_H
#define MCS360_STACK_STL_VECTOR_H
#include <vector>
namespace mcs360_stack_stl_vector
{
    template <typename T>
    class Stack
    {
    private:
        std::vector<T> data;
    public:
        void push( T item );
        bool empty();
        T top();
        void pop();
    };
}
#include "mcs360_stack_stl_vector.tc"
#endif
```

mcs360_stack_stl_vector.tc

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
ownour own vector
our own single linked
list

```
namespace mcs360_stack_stl_vector
{
    template <typename T>
    void Stack<T>::push( T item ) {
        this->data.push_back(item);
    }
    template <typename T>
    bool Stack<T>::empty() {
        return this->data.empty();
    }
    template <typename T>
    T Stack<T>::top() {
        return this->data.back();
    }
    template <typename T>
    void Stack<T>::pop() {
        this->data.pop_back();
    }
}
```

testing: 3 2 1 blast off!

Encapsulating
Vectors and
ListsSTL vectors and lists
implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
ownour own vector
our own single linked
list

```
#include "mcs360_stack_stl_vector.h"
#include <iostream>
using namespace mcs360_stack_stl_vector;
using namespace std;

int main()
{
    Stack<int> s;

    for(int i=1; i<=3; i++) s.push(i);
    while(!s.empty())
    {
        int item = s.top();
        cout << " " << item;
        s.pop();
    }
    cout << " blast off!" << endl;
    return 0;
}
```

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

Implementation Issues

g++ compiler

directive -D

defining our own exception

Implementing Stacks on our own

our own vector

our own single linked list

list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

stack uses STL list

Encapsulating
Vectors and
Lists

STL vectors and lists
implementing a stack
with the STL vector

implementing a stack
with the STL list

Implementation
Issues

g++ compiler
directive -D

defining our own
exception

Implementing
Stacks on our
own

our own vector
our own single linked
list

```
#ifndef MCS360_STACK_STL_LIST_H
#define MCS360_STACK_STL_LIST_H
#include <list>
namespace mcs360_stack_stl_list
{
    template <typename T>
    class Stack
    {
    private:
        std::list<T> data;
    public:
        void push( T item );
        bool empty();
        T top();
        void pop();
    };
}
#include "mcs360_stack_stl_list.tc"
#endif
```

mcs360_stack_stl_list.tc

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
namespace mcs360_stack_stl_list
{
    template <typename T>
    void Stack<T>::push( T item ) {
        this->data.push_back(item);
    }
    template <typename T>
    bool Stack<T>::empty() {
        return this->data.empty();
    }
    template <typename T>
    T Stack<T>::top() {
        return this->data.back();
    }
    template <typename T>
    void Stack<T>::pop() {
        this->data.pop_back();
    }
}
```

testing: 3 2 1 blast off!

Encapsulating
Vectors and
Lists

STL vectors and lists
implementing a stack
with the STL vector

implementing a stack
with the STL list

Implementation
Issues

g++ compiler
directive -D

defining our own
exception

Implementing
Stacks on our
own

our own vector
our own single linked
list

```
#include "mcs360_stack_stl_list.h"
#include <iostream>
using namespace mcs360_stack_stl_list;
using namespace std;

int main()
{
    Stack<int> s;

    for(int i=1; i<=3; i++) s.push(i);
    while(!s.empty())
    {
        int item = s.top();
        cout << " " << item;
        s.pop();
    }
    cout << " blast off!" << endl;
    return 0;
}
```

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists
implementing a stack
with the STL vector
implementing a stack
with the STL list

Implementation Issues

g++ compiler
directive -D
defining our own
exception

Implementing Stacks on our own

our own vector
our own single linked
list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

uniform interfaces

Encapsulating
Vectors and
Lists

STL vectors and lists
implementing a stack
with the STL vector

implementing a stack
with the STL list

Implementation
Issues

g++ compiler
directive -D

defining our own
exception

Implementing
Stacks on our
own

our own vector
our own single linked
list

Today we test 4 stack implementations:

$$\{ \text{STL, our own} \} \times \{ \text{vector/array, linked list} \}$$

with the *same* test program “blast off!”.

Header of `use_mcs_360_stl_vector.cpp`:

```
#include "mcs360_stack_stl_vector.h"
#include <iostream>
using namespace mcs360_stack_stl_vector;
using namespace std;
```

Header of `use_mcs_360_stl_list.cpp`:

```
#include "mcs360_stack_stl_list.h"
#include <iostream>
using namespace mcs360_stack_stl_list;
using namespace std;
```

one test program with g++ -DEncapsulating
Vectors and
ListsSTL vectors and lists
implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
ownour own vector
our own single linked
list

Except for the headers, the test programs are the same.

Header of `use_mcs360_stack.cpp` starts with

```
#ifndef mcs360_stl_list
#include "mcs360_stack_stl_list.h"
using namespace mcs360_stack_stl_list;
#define kind 2
#else
#include "mcs360_stack_stl_vector.h"
using namespace mcs360_stack_stl_vector;
#define kind 1
#endif
```

Note (g++ -E calls only the preprocessor):

- g++ -Dmcs360_stl_list changes default,
- in main we can use the value of kind.

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

Implementation Issues

g++ compiler directive `-D`

defining our own exception

Implementing Stacks on our own

our own vector

our own single linked list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive `-D`

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

the exception `Empty_Stack`Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive `-D`defining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

Deriving from the exception class

`std::invalid_argument`in the file `mcs360_empty_stack.h`:

```
#ifndef MCS360_EMPTY_STACK_H
#define MCS360_EMPTY_STACK_H
#include <stdexcept>
```

```
class Empty_Stack : public std::invalid_argument
{
public:

    Empty_Stack(std::string s) :
        std::invalid_argument(s) {}
};
#endif
```

testing the exception

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#include "mcs360_empty_stack.h"
#include <iostream>
using namespace std;

int main()
{
    try {
        cout << "trying to pop ..." << endl;
        throw Empty_Stack
            ("throwing empty stack, please catch");
    } catch(Empty_Stack e) {
        cout << "caught exception with message \""
            << e.what() << "\"" << endl;
    }
    cout << "throwing, no try-catch ..." << endl;
    throw Empty_Stack
        ("pop fails because empty stack");
```

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists
implementing a stack with the STL vector
implementing a stack with the STL list

Implementation Issues

g++ compiler directive `-D`
defining our own exception

Implementing Stacks on our own

our own vector
our own single linked list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive `-D`

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

mcs360_stack_our_vector.h

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#ifndef MCS360_STACK_OUR_VECTOR_H
#define MCS360_STACK_OUR_VECTOR_H
namespace mcs360_stack_our_vector {
    template <typename T>
    class Stack {
        private:
            static const int capacity = 64;
            int current;
            T *data;
        public:
            Stack()
            // creates an empty stack
            {
                data = new T[capacity];
                current = -1;
            }
    };
}
```

mcs360_stack_our_vector.tc

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#include "mcs360_empty_stack.h"

namespace mcs360_stack_our_vector
{
    template <typename T>
    void Stack<T>::push( T item )
    {
        this->current = this->current + 1;
        this->data[current] = item;
    }

    template <typename T>
    bool Stack<T>::empty()
    {
        return (this->current == -1);
    }
}
```

top and pop

```
template <typename T>
T Stack<T>::top()
{
    if(current<0)
        throw Empty_Stack("No top!");
    return this->data[current];
}

template <typename T>
void Stack<T>::pop()
{
    if(current<0)
        throw Empty_Stack("No pop!");
    this->current = this->current - 1;
}
}
```

testing: 3 2 1 blast off!

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```

#include "mcs360_stack_our_vector.h"
#include <iostream>
using namespace mcs360_stack_our_vector;
using namespace std;

int main()
{
    Stack<int> s;

    for(int i=1; i<=3; i++) s.push(i);
    while(!s.empty())
    {
        int item = s.top();
        cout << " " << item;
        s.pop();
    }
    cout << " blast off!" << endl;
    return 0;
}

```

Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists
implementing a stack with the STL vector
implementing a stack with the STL list

Implementation Issues

g++ compiler directive -D
defining our own exception

Implementing Stacks on our own

our own vector
our own single linked list

1 Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack with the STL vector

implementing a stack with the STL list

2 Implementation Issues

g++ compiler directive -D

defining our own exception

3 Implementing Stacks on our own

our own vector

our own single linked list

mcs360_stack_our_list.h

Encapsulating
Vectors and
Lists

STL vectors and lists
implementing a stack
with the STL vector
implementing a stack
with the STL list

Implementation
Issues

g++ compiler
directive -D
defining our own
exception

Implementing
Stacks on our
own

our own vector
our own single linked
list

```
#ifndef MCS360_STACK_OUR_LIST_H
#define MCS360_STACK_OUR_LIST_H
#define NULL 0
namespace mcs360_stack_our_list {
    template <typename T>
    class Stack {
        private:
            #include "mcs360_stack_our_node.h"
            Node *L;
        public:
            Stack() { L = NULL; }
            void push( T item );
            bool empty();
            T top();
            void pop();
    };
}
#include "mcs360_stack_our_list.tc"
#endif
```

mcs360_stack_our_node.h

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#ifndef MCS360_STACK_OUR_NODE_H
#define MCS360_STACK_OUR_NODE_H

struct Node
{
    T data; // T is template parameter
    Node *next; // pointer to next node

    Node(const T& item, Node *ptr = NULL) :
        data(item), next(ptr) {}
};
#endif
```

mcs360_stack_our_list.tc

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```
#include "mcs360_empty_stack.h"

namespace mcs360_stack_our_list
{
    template <typename T>
    void Stack<T>::push( T item )
    {
        Node *nd;
        nd = new Node(item,this->L);
        this->L = nd;
    }

    template <typename T>
    bool Stack<T>::empty()
    {
        return (this->L == NULL);
    }
}
```

top and pop

Encapsulating
Vectors and
Lists

STL vectors and lists

implementing a stack
with the STL vectorimplementing a stack
with the STL listImplementation
Issuesg++ compiler
directive -Ddefining our own
exceptionImplementing
Stacks on our
own

our own vector

our own single linked
list

```

template <typename T>
T Stack<T>::top()
{
    if(this->empty())
        throw Empty_Stack("No top!");
    return this->L->data;
}

```

```

template <typename T>
void Stack<T>::pop()
{
    if(this->empty())
        throw Empty_Stack("No pop!");
    Node *nd;
    nd = this->L->next;
    delete this->L;
    this->L = nd;
}
}

```

testing: 3 2 1 blast off!

Encapsulating
Vectors and
Lists

STL vectors and lists
implementing a stack
with the STL vector
implementing a stack
with the STL list

Implementation
Issues

g++ compiler
directive -D
defining our own
exception

Implementing
Stacks on our
own

our own vector
our own single linked
list

```
#include "mcs360_stack_our_list.h"
#include <iostream>
using namespace mcs360_stack_our_list;
using namespace std;

int main()
{
    Stack<int> s;

    for(int i=1; i<=3; i++) s.push(i);
    while(!s.empty())
    {
        int item = s.top();
        cout << " " << item;
        s.pop();
    }
    cout << " blast off!" << endl;
    return 0;
}
```

Comparing Stack Implementations

Encapsulating Vectors and Lists

STL vectors and lists

implementing a stack
with the STL vector

implementing a stack
with the STL list

Implementation Issues

g++ compiler
directive -D

defining our own
exception

Implementing Stacks on our own

our own vector

our own single linked
list

When comparing we consider

- time: $O(1)$ for all operations?
- space: wasting memory?

Compared to vectors, lists seem to be the winner:

- + adding to front of list is $O(1)$,
no $O(n)$ reallocation as with vectors.
- + memory efficient
vectors reserve space, may go unused.

However, consider the following:

- standard allocation occurs in blocks,
- frequent calls to OS block threads.

Summary + Assignments

Discussed 4 implementations of a stack, covered §5.3.

Assignments:

- 1 Make the files `mcs360_stack_stl.h` and `.tc` so that with `g++ -Dmcs360_stl_list` the STL `list` class is used instead of the default STL `vector` class.
- 2 Adjust `mcs360_stack_stl_vector.tc` so that the `top` and `pop` throw `Empty_Stack` when appropriate.
- 3 Define an exception class `Stack_Overflow` for use in `mcs360_stack_our_vector.tc` when a client does a `push` on a full stack.
- 4 Provide `mcs360_stack_our_vector.tc` with a `reserve` function that doubles the capacity of the stack when a client does a `push` on a full stack.