Massey University
COLLEGE OF SCIENCES

159201

Week 3

2014

Te Kunenga
ki Pürehuroa

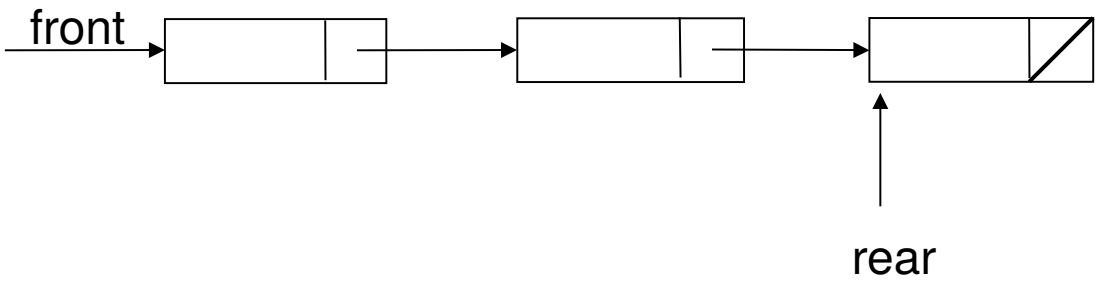
Massey University
COLLEGE OF SCIENCES

L09

Te Kunenga
ki Pūrehuroa

Queues based on Linked-List

```
struct Node {
   float data;
   Node *next;
};
```



Queues based on LL: Node

The same Node structure can be used:

```
struct Node {
   float data;
   Node *next;
};
```


Queues based on LL: class

```
class Queue {
private:
   Node *front, *rear;

public:
   Queue();
   ~Queue();
   void Join(float newthing);
   void Leave();
   float Front();
   bool isEmpty();
};
```


Constructor and destructor

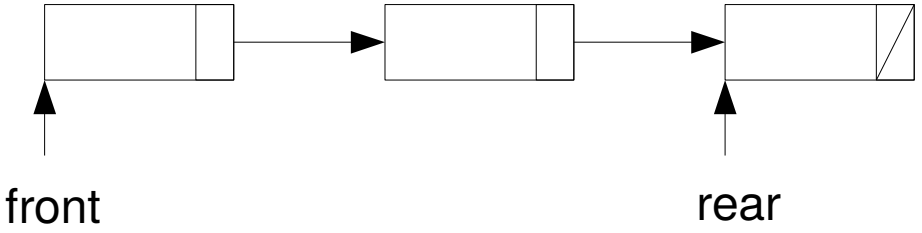
```
Queue::Queue() {
// constructor
   front = NULL;   rear = NULL;
}

Queue::~Queue() {
// destructor
}
```

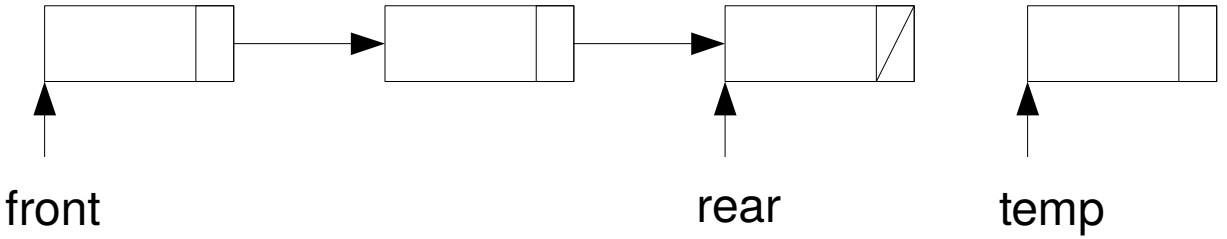

Queues with LL: Join()

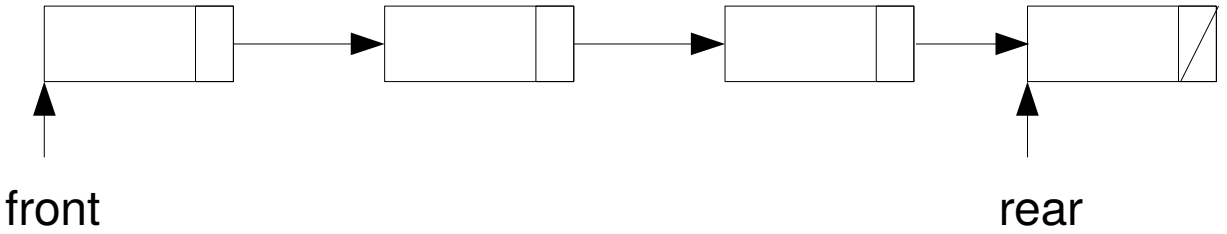
```
void Queue::Join(float newthing) {
// place the new thing at the rear of the queue
  Node *temp;
  temp = new Node;
  temp->data = newthing;
  temp->next = NULL;
  if (rear != NULL) { rear->next = temp; }
  rear = temp;
  if (front == NULL) { front = temp; }
}
//do we cover all special cases?
```


Queues with LL: Join()

Case 1: queue with some elements

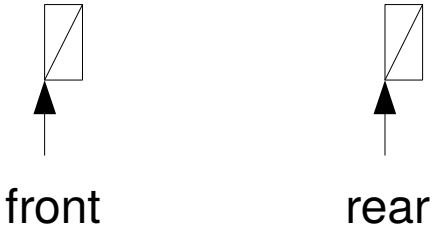


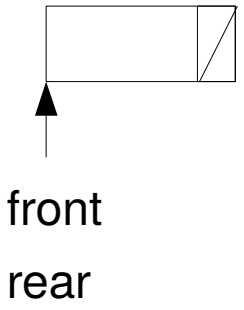




Queues with LL: Join()

Case 2: empty queue





Queues with LL: Leave()

```
void Queue::Leave() {
// remove the front item from the queue
Node * temp;
   if (front == NULL) { return; }
   temp = front;
   front = front->next;
   if (front == NULL) { rear = NULL; }
   delete temp;
}
//did we cover all cases? When queue is empty?
```


Queues with LL: Front() and isEmpty()

```
float Queue::Front() {
// return the value of the front item
   return front->data;
}
//did we cover the case where the queue is
empty?
bool Queue::isEmpty() {
// return true if the queue is empty
   if (front == NULL) { return true; }
   return false;
}
```


Notes about isEmpty()

This function applies to any data structure (not just stacks and queues).

If the isEmpty() function is used immediately after the constructor, it must always return **true**.

Queues with LL: main() sample

```
Queue G, H;
int main() {
  G.Join(7.1);
  printf("%f is at the front of queue G\n",
G.Front());
  H.Join(7.1);
  if (H.isEmpty()) {
     printf("Queue H is empty\n");
  } else {
     printf("%f is at the front of queue H\n",
H.Front());
  }
}
```


Notes: Queues based on LL

- Avoids the creeping problem
- The queues are not of a fixed size
- Random access is not required → FIFO
- **Conclusion**: it is always a good idea to base a queue on a linked-list

Massey University
COLLEGE OF SCIENCES

L 10

Te Kunenga
ki Pūrehuroa

Massey University
COLLEGE OF SCIENCES

Vectors, Lists, Templates

Let's take a closer look at standard data types

Vectors: arrays that can extend

Lists: linked-lists with operations

Templates: open data types (only specify when the object is declared)

STL → **Standard Template Library**

Te Kunenga
ki Pūrehuroa

Vectors

A vector is an abstract data type (ADT)

A vector is an array that can be extended

(the size increases during the runtime)

There are many operations for a vector, e.g.:

```
insert_at_front()

insert_at_rear()

insert_at_index(i)

Various forms of delete_*() (as per insert)

Lenght, etc
```


Vectors

All elements of a vector have an index, which is used in some operations, e.g.:

```
v[i]=32;
```

Implementation of a vector cannot use C/C++ arrays (it would be nice if we could...)

Vectors should be **extendable**.

Vectors Implementation

a) Make a **pointer** and allocate memory in blocks. When memory is full, allocate a **larger new block** and change the pointer, deleting the **old block**.

One can declare vectors with an ***initial size***.

b) Implement the vector as a linked-list of memory blocks. Each block holds a number of elements. Create new blocks when required. If using this option, vectors can be declared with no initial size.

Note: option a) is fast but may have a long wait if new memory is required.

Option b) is slower than an array most of the time.

Lists

A list is an abstract data type (ADT).

A list can be though of as a linked-list with **operations**, e.g.:

```
insert_after(item x);

insert_at_front();

delete_after(item x);

look_at_first_item(); ...etc
```

The elements of a list do not have an ***absolute*** index like a vector.

Lists

The elements of a list do have ***relative*** positions to each other.

The relative positions are kept by pointers (the usual `next`, `previous` etc).

Usually `front` and `rear` pointers are available, these are absolute positions.

Lists implementation

A list is usually implemented as a **linked-list**.

Remember, linked-lists are good for sequential access.

Doubly-linked lists may be useful if access other than sequential access is needed.

Templates

Templates can be applied to any data structure.

A template is a data structure where the data type of the elements is not previously specified.

The data type is specified when the object is **declared**.

Many templates are available in the STL (Standard Template Library)

Templates: Example 1

```
#include <stack>
using namespace std; // this MUST appear in
                     //ANY C++ program
stack<char> S;       // note the data type
S.push('A');         // note that 'push' is
                     //lower case
printf("Size is %d \n", S.size());
```

Running this code prints:

```
Size is 1
```


Note on namespace

```
using namespace std;
```

This means that you don't need to put std:: all over your C++ code.

If you use standard C++ library, you would have to write either:

```
sdt::cout << “print something\n”;
```

```
using namespace std;
...
cout << “print something\n”;
```


Templates: Example 2

```
#include <vector>
using namespace std;
vector<int> v;
int main() {
  int num;
  printf("What size vector do you want? ");
  scanf("%i", &num);
  v.resize(num);                // you MUST set a size before using
  printf("Max size of this vector is %i\n", v.size());
  for (int i = 0; i < 4; i++) {
    printf("Enter a number ");
    scanf("%i", &num);
    v[i] = num;
  }
  for (int i = 0; i < 4; i++) {
    printf("%i\n", v[i]);
  }
}
```


Templates: Example 3

```
#include <vector>
using namespace std;
vector<int> x;

int main() {
  int num;
// here we DO NOT set a size
  for (int i = 0; i < 4; i++) {
    printf("Enter a number ");
    scanf("%i", &num);
    x.push_back(num);
  }
  printf("Max size of this vector is %i", x.size());      // size will be 4

  // now print the numbers as above
}
```


STL

Want to learn more about STL?

Many links, e.g.

http://www.cplusplus.com/reference/stl/

We will explore some of the containers and functions, but not all of them.

You learn as you use it.

NOTES:

many different implementations of STL

Not all are good, there are notorious bugs in some parts of some STL implementations

STL stacks

Not surprisingly, stacks in STL offers the following member functions (slightly different names):

```
empty()

size()

top()

push()

pop()
```


Massey University
COLLEGE OF SCIENCES

STL vectors

STL has the following extra operations:

```
begin, end, rbegin, rend

size, max_size, resize, capacity, empty,
reserve

operator[], at , front, back

assign, push_back, pop_back, insert, erase,
swap, clear, get_allocator


etc
```

Te Kunenga
ki Pūrehuroa

Getting multiple values back

A method can return one value or address back

How can we get more than one value back?

A strange way to do it is to pass an address as argument and change the content where the address points to...

The values are then available to external function via that public method.

Lets see an example:

Stack: getting two values back

Suppose you need to get two values from the top from a stack. The method Top() as we know it can only return one value:

```
float Stack::Top() {//return value of the top item in the stack
        return data[index]; //Warning: what if stack is empty?
}
```

OR

```
float Stack::Top() {//return value of the top item in the stack
        return listpointer->data;//WARNING: what if listpointer is NULL?
}
```


Getting two values back 1

Solution 1: pass an address of an array

```
bool Stack::TopTwo(float *array) {
      if(listpointer==NULL || listpointer->next==NULL) {
            array[0]=-1;
            array[1]=-1;
            return false;//not two items
      }
      else{
            array[0]=listpointer->data;
            array[1]=(listpointer->next)->data;
            return true; //success
      }
}
```


Getting two values back 1

Solution 1: at main

```
float temparray[2];
A.TopTwo(temparray);
//A.TopTwo(&temparray[0]) is the same address
cout << "Returned values " <<
      temparray[0] << << " " << temparray[1] << endl;
```


Getting two values back 2

Solution 2: pass two addresses of floats ('C style')

```
bool Stack::TopTwo(float *a, float *b) {
      if(listpointer==NULL || listpointer->next==NULL) {
            return false;//not two items
      }
      else{
            *a = listpointer->data;
            *b = (listpointer->next)->data;
            return true; //success
      }
}
```


Getting two values back 2

Solution 2: at main, two possibilities:

```
float a,b;
A.PopTwoandReturn(&a,&b);
cout << "Returned values " << a << << " " << b << endl;
```

```
float *a,*b;
a=new float; b=new float;
A.PopTwoandReturn(a,b);
cout << "Returned values " << *a << << " " << *b <<
endl;
```


Getting two values back 3

Solution 3: pass two addresses of floats ('C++ style')

```
bool Stack::TopTwo(float &a, float &b) {
      if(listpointer==NULL || listpointer->next==NULL) {
            return false;//not two items
      }
      else{
            a = listpointer->data;
            b = (listpointer->next)->data;
            return true; //success
      }
}
```


Getting two values back 3

Solution 3: at main

```
float a,b;
A.PopTwoandReturn(a,b);
cout << "Returned values " << a << << " " << b << endl;
```


Challenges

1) Using STL, create a queue *and a* stack *in the same C++ program, and use the basic operations on main() to play with elements for both ADTs.*

Massey University
COLLEGE OF SCIENCES

L 11

Te Kunenga
ki Pūrehuroa

Template list in C++

Templates are not lists, lists are not templates.

Templates can in theory use **any** data structure.

Templates are compiled with the data type inserted into the template.

In certain functions we may have problems.

E.g., in a function `FirstItem()` a line such as:

```
item = front->data;
```

Would not work if T is a *stack* or other complex data type.

Templates: a List

```
template <class T>
class List {
private:
  struct Node {
    T data;
    Node *next;
  };
  Node *front, *current;

public:
  List();
  ~List();
  void AddtoFront(T newthing);
  bool FirstItem(T & item);
  bool NextItem(T & item);
};
```


Templates: a List (part 2)

```
//constructor and destructor

template <class T>
List<T>::List() {
  front = NULL;  current = NULL;
}


template <class T>
List<T>::~List() {

}
```


Templates: a List (part 3)

```
//functions AddtoFront and FirstItem
template <class T>
void List<T>::AddtoFront(T newthing) {
Node *temp;
  temp = new Node;
  temp->data = newthing;
  temp->next = front;
  front = temp;
}

template <class T>
bool List<T>::FirstItem(T & item) {
  if (front == NULL) { return false; }
  item = front->data;
  current = front;
  return true;
}
```


Templates: a List (part 4)

```
//function NextItem

template <class T>
bool List<T>::NextItem(T & item) {
  current = current->next;
  if (current == NULL) { return false; }
  item = current->data;
  return true;
}
```


Templates: a List (part 5)

```
//main()
List<int> L; //create a list of integers

int main() {

  L.AddtoFront(28);
  L.AddtoFront(54);            // which number is now first in the list?
  L.AddtoFront(53);
  L.AddtoFront(52);

  bool ok;  int x;
  ok = L.FirstItem(x);
  while (ok) {
    printf("%i \n", x);
    ok = L.NextItem(x);
  }
}
```


Templates: a List (comments)

```
List<int> L; //create a list of integers

int main() {

  L.AddtoFront(28);
  L.AddtoFront(54);
  L.AddtoFront(53);
  L.AddtoFront(52);
  bool ok;  int x;
  ok = L.FirstItem(x);   the value is 'returning' to x through its pointer
  while (ok) {
    printf("%i \n", x);
    ok = L.NextItem(x);  the pointer to next is private!
  }
}
```


Massey University
COLLEGE OF SCIENCES

L 12

Te Kunenga
ki Pūrehuroa

A book list

Lets see an example with lists.

We want to create a simple book list with Title and Call_number.

We are going to use our previous class List.

We either copy and paste or include that file as part of our code. (Section A, from previous lect.)

We then create a class Book, with specific member functions. (Section B)

Finally, main() and a simple display function is created (Section C)

A book list example: class Book

```
//-----------------------------------------
// Section B : the Book class
class Book {
private:
  char title[80];
  float callnumber;
public:
  Book() {callnumber = 0.0;}          // an inline function
  ~Book() { }
  void GetData();
  void Display();
};
```


A book list example: functions

```
void Book::GetData() {
  cout << "Enter the title " << endl;
  //fgets(title, sizeof(title),stdin);
  cin >> title;
  cout << "Enter the call number " << endl;
  //fgets(callnumber, sizeof(callnumber),stdin);//WRONG!! callnumber is a
float
  cin >> callnumber;//same story: what happens if you enter a char?
}


void Book::Display() {
cout << callnumber << " " << title << endl;
}
```


A book list example: main()

```
//-----------------------------------------------
// Section C : functions, global variables and main
void DisplayAllBooks();
List<Book> booklist;  //global var

int main() {
  Book temp;  char ch; //ch a single char, not such a good idea...
  // load some books into the booklist
  cout << "Enter several books into the list" << endl;
  while (true) {
    temp.GetData();
    booklist.AddtoFront(temp);   // will this have copy problems?  No!
    cout << "Enter another book? (Y/N) " << endl;
    cin >> ch;
    if (ch == 'N') { break; }
  }
  cout << "Here is the list in REVERSE order" << endl;
  DisplayAllBooks();
}
```


A book list example: Display

```
void DisplayAllBooks(){
        //display all books from the booklist
        bool ok;
        Book temp;//why do we need this here?
        ok = booklist.FirstItem(temp);
        while (ok == true) {
                temp.Display();
                ok = booklist.NextItem(temp);
        }
}
```


Words on I/Os in C and C++

Many former standard *.h files are deprecated

This is a source of confusion to the modern compilers.

Example:

```
#include <iostream.h> //old style, deprecated


#include <iostream> //new style

using namespace std; //essential to avoid std::
```


I/Os in C

Never use `gets()`

This is a deprecated function that is prone to buffer overflow attacks.

Use `fgets()` instead. It uses standard input (keyboard).

Example:

```
char title[80];

...

gets (title);//NO

fgets(title, sizeof(title),stdin);//YES
```


I/Os in C++

use `istream::get` or `istream::getline`

Example (from standard input):

```
char title[80];

...

cin.get(title,80);//no overflow,as long as 80 chars

//or

cin.getline(title,80);
```


Massey University
COLLEGE OF SCIENCES

Challenges

1) Re-implement (or copy) the List class discussed above

Modify the program to read data from a text file and fill up the List with lots of data.

Print the List to make sure that all data from the text file is copied to the List.

Te Kunenga
ki Pürehuroa