Comp2014 Object Oriented Programming

Lecture 11

Class Templates and Linked Data Structures

Topics covered by the lecture

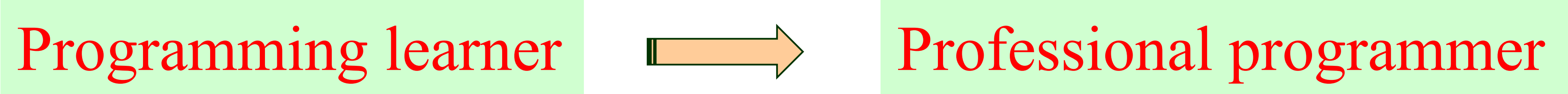
- Function and function overloading (review)
- Operator overloading

Difficulty factor: ****

- References (mind the difference between C++ and Java)
- Friends (function or class)

Topic covered in the lecture

- Template
  - Function template
  - Class template
- General concept of data structures
- Linked list



Programming learner

Professional programmer

functionalities

efficiency

See example arraypi.cpp

Function Templates

- Suppose that we have a set of arrays, each in different data type. We need to write a function to print the members of each array.

```
void printArray(int *array, int count) {
   for( int i = 0; i < count; i++)
            cout << array[ i ] << “ “;
   cout << endl;
}
```

```
void printArray(double *array, int count) {
   for( int i = 0; i < count; i++)
            cout << array[ i ] << “ “;
   cout << endl;
}
```

```
void printArray(char *array, int count) {
   for( int i = 0; i < count; i++)
            cout << array[ i ] << “ “;
   cout << endl;
}
```

Say no to copy-and-paste

Can we merge them into a single function?

Yes, we can!

Function Templates

Templates are used in cases where we want to use a range of similar functions or develop a set of similar classes. In these situations we write a base version which is then adapted to different data types.

```
template <class Datatype>
void printArray( Datatype *array, int count) {
  for( int i = 0; i < count; i++)
        cout << array[ i ] << “ “;
  cout << endl;
}
```

For the above piece of code, when the compiler finds a reference to the printArray function in the code it substitutes the type of the first parameter throughout the function.

See arraypft.cpp

Class Templates

They are called *containers*.

Many data structures, such as *array*, *linked lists*, *stacks*, *queues* etc., can be thought of independently from the type of objects within them. Operationally they are the same irrespective of whether they contain integers, doubles, characters or any user defined data type. Thus if we define a class for a data structure it would be useful if it could contain data in any data type. To do this we must use a CLASS TEMPLATE.

Class Templates: declaration

Demonstrate an implementation of vector

```
template<class T>
class dArrayT {
 private:
    T* arr;
    int size;      // The number of filled numbers
    int capacity; // The capacity of the array
    void resize();
 public:
    dArrayT(int c);
    ~dArrayT();
    bool insert(int pos, T val);
    bool remove(int pos);
    int length() const {return size;}
    T& operator[] (int i); //operator overloading (as a l-value)
    T operator[] (int i) const; //operator overloading
};
```

There is a data type variable T here.

Class Templates: definition

Each method of a template class is a template function

```
template<class T>
bool dArrayT<T>::insert(int pos, T val) {
    if (pos < 0 || pos > size)
       return false;
    if(size + 1 > capacity)
       resize();
    if(pos == size) {
       arr[size++] = val;
    } else {
       // move right
       for(int i=size-1;i>=pos;i--)
           arr[i+1] = arr[i];
       arr[pos] = val;
       size++;
    }
    return true;
}
```

Remember to use the *template* keyword before each member function outside the class declaration and the sharp angle bracket <T> in the class scope.

More methods …

Class Templates: applications

```
int main() {
    dArrayT<int> a1;    // array of integers
    dArrayT<double> a2; // array of doubles
    dArrayT<char> a3;   // array of characters
    dArrayT<FlightTicket> a4;
    dArrayT<HotelVoucher> a5;
    dArrayT<EventTicket> a6;
    dArrayT<Package> a7;

    //….

    return 0;
}
```

See dArrayTUsage.cpp

Vector: a dynamic array template

vector is a dynamic array in the **Standard Template Library (STL)**. Many methods are implemented.

Header file:

```
#include <vector>
```

```
vector<Order> OrderBundle;
```

Simple operations:

- `OrderBundle.push_back(order)`: *appending a new element value at the end.*
- `OrderBundle.pop_back()`: *removing an element form the end.*
- `OrderBundle[index]` : *random access with index*

See vectorApp.cpp

Array vs vector

- Array is relatively more efficient than vector. If you know the size of the data, use array.
- Vector is more flexible, especially if you do not know the size of data.
  - Be careful of the difference:

```
extPersonType list[500];//500 objects
vector<extPersonType> list;//0 objects
```

- A set of built-in functions in the Standard Template Library can be used with vector, such as sort, max, min, …

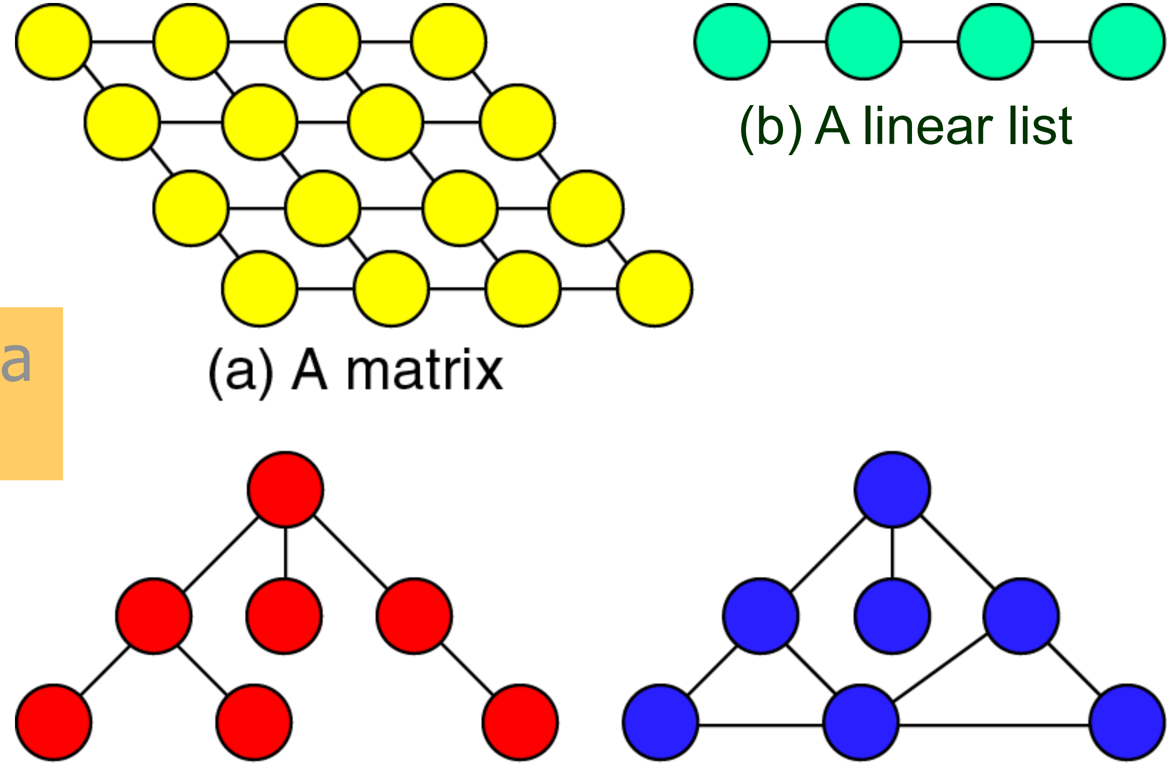
See Practical Task 7.5

Data structure: general concept

- A dynamic array can be more useful and convenient than the normal array but
  - Changing the size of the array requires creating a new array and then copying all data from the array with the old size to the other array with the new size
  - The data in the array are next to each other sequentially in memory, which means that inserting an item inside the array requires shifting some other data in the array.
- We do not have to store data in an array!

Data structure: general concept

Typical data structures



(a) A matrix

(b) A linear list

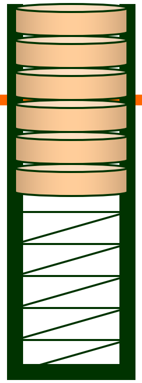
(c) A tree

(d) A network

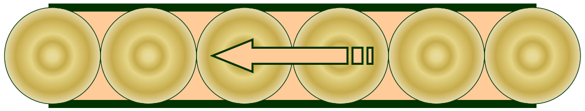
Typical linked data structures

- Linked list:

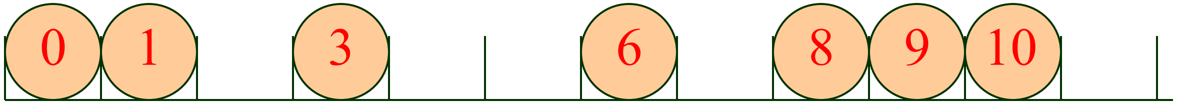
- Stack: first in last out (FILO)



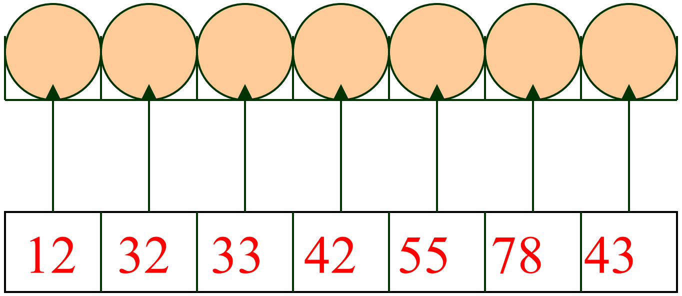
- Queue: first in first out (FIFO)



- Hash table:

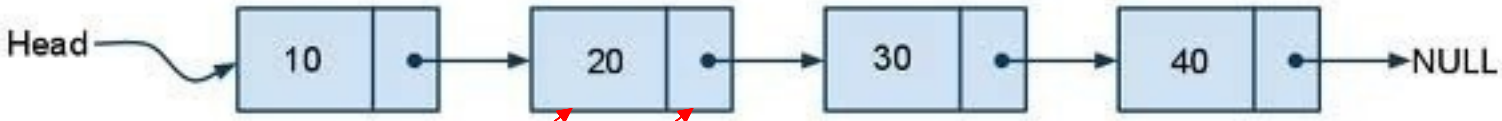


- Map:



What are inside? objects!!!

Linked List Concept

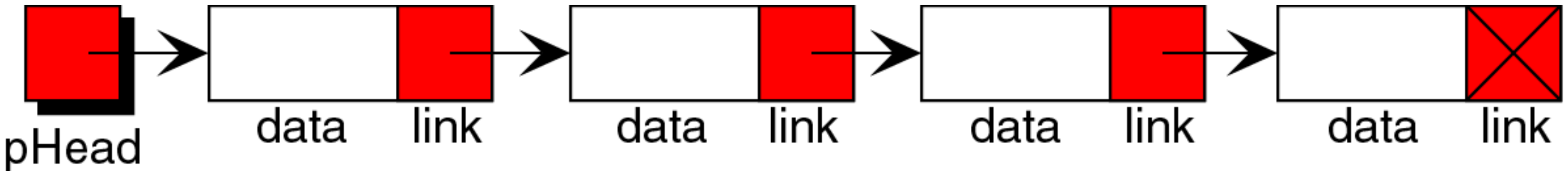


```
Template<class Type>
class Node {
public:
   Type data;
   Node *link;
};
```

Linked list: an ordered collection of data in which each element (node) contains the location of the next element

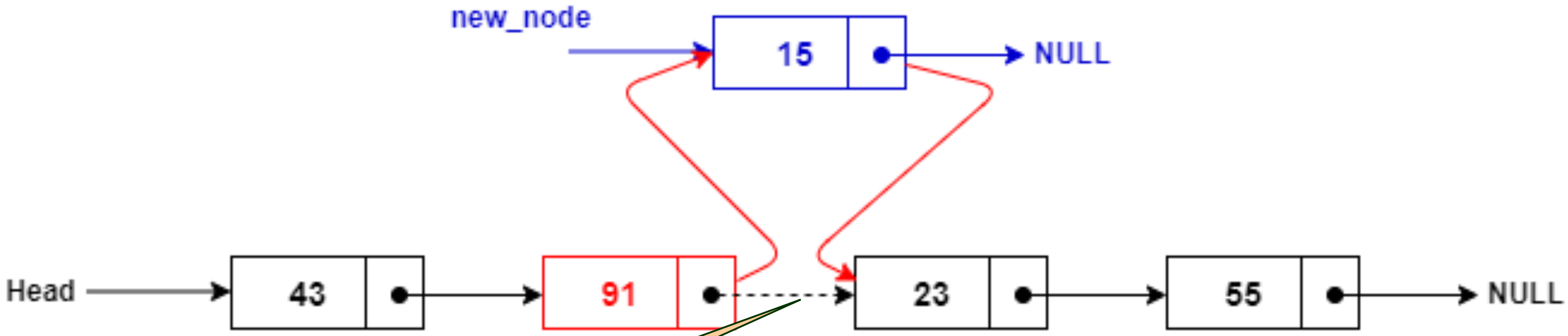
See example SimpleLinkedList.cpp

Typical operations



- Insert a node
- Delete a node
- Modify a node
- Search for a data item in a node
- Traversal over a linked list

Insert a node into a linked list

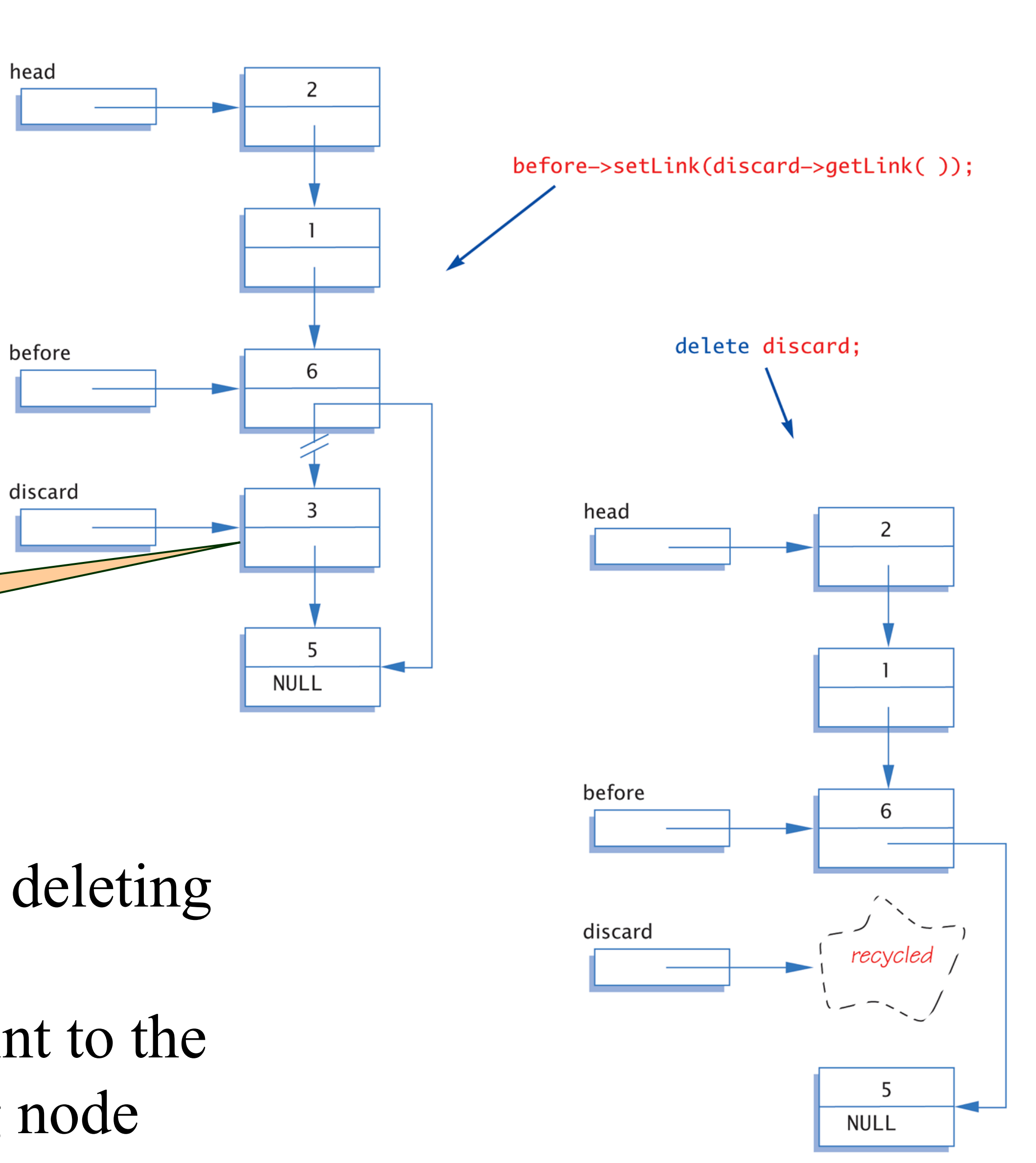


Insert this node

- Find the position to insert
- Create a node
- Put in the data
- Put in the pointer of next node
- Change the pointer of the node before

Deleting a Node

Display 17.7 **Removing a Node**



```
before->setLink(discard->getLink( ));
```

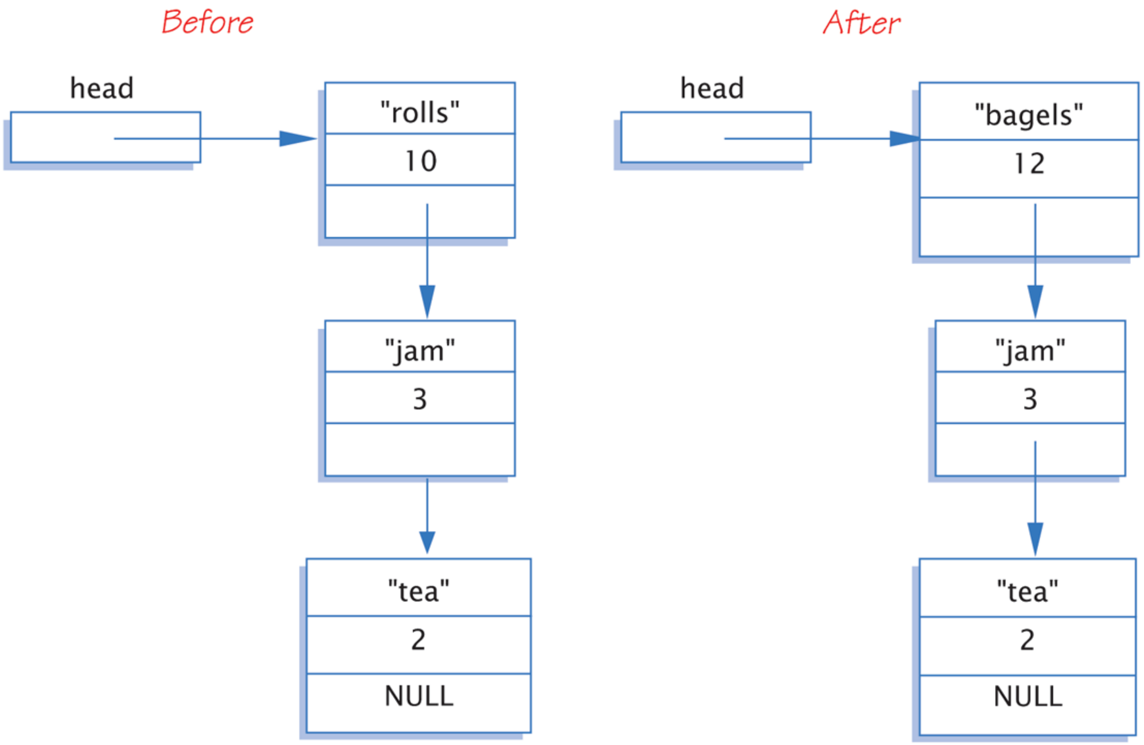
```
delete discard;
```

Delete this node

- Find the node before the deleting node
- Change its pointer to point to the node next to the deleting node
- Recycle the memory.

Modify a node

```
head->count = 12;
head->item = "bagels";
```



Homework

- Read textbook Chapters 16 & 17.
- Work on your assignment 2 if you have not complete. The deadline for assignment 2 is 5pm Friday 14 Oct 2022.
- Demonstration of assignment 2 will be in your practical class next week.