NGÔN NGỮ LẬP TRÌNH

Bài 11: Các cấu trúc dữ liệu liên kết

Giảng viên: Lý Anh Tuấn

Email: tuanla@tlu.edu.vn

Nội dung

1. Các nút và danh sách liên kết
   - Tạo, tìm kiếm
2. Ứng dụng danh sách liên kết
   - Ngăn xếp, hàng đợi
3. Biến lặp (iterator)
   - Con trỏ và biến lặp

Giới thiệu

- Danh sách liên kết
  - Được tạo bằng việc sử dụng các con trỏ
  - Mở rộng và co lại trong thời gian chạy
- Cây cũng sử dụng các con trỏ
- Các con trỏ là xương sống của các cấu trúc như vậy
  - Sử dụng các biến động
- Thư viện khuôn mẫu chuẩn
  - Có phiên bản định nghĩa trước của một số cấu trúc

Tiếp cận

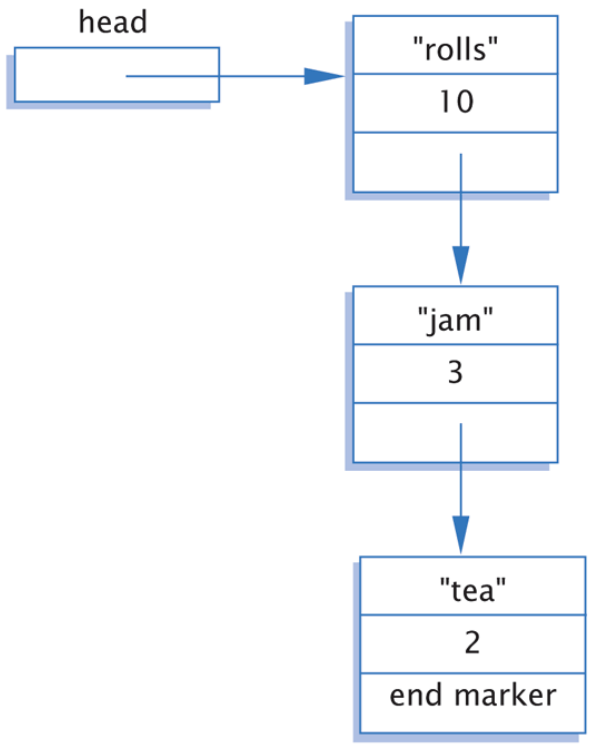
- Ba cách vận hành các cấu trúc dữ liêu như vậy
  1. Các hàm toàn cục và các struct có dữ liệu public
  2. Các lớp với các biến thành viên private và các hàm truy cập và biến đổi
  3. Các lớp bạn
- Danh sách liên kết sử dụng phương pháp 1
- Ngăn xếp, hàng đợi sử dụng phương pháp 2
- Cây sử dụng phương pháp 3

Nút và danh sách liên kết

- Danh sách liên kết
  - Ví dụ đơn giản của "cấu trúc dữ liệu động"
  - Bao gồm các nút
- Mỗi nút là biến kiểu cấu trúc hoặc lớp được tạo động bằng new
  - Các nút cũng bao gồm các con trỏ tới các nút khác
  - Cung cấp các liên kết

Nút và con trỏ

Display 17.1 **Nodes and Pointers**



Định nghĩa nút

- struct ListNode
  {
  string item;
  int count;
  ListNode *link;
  };
  typedef ListNode* ListNodePtr;
- Trật tự ở đây là quan trọng
  - Listnode được định nghĩa trước vì được sử dụng trong typedef
- Cũng lưu ý tới khả năng có chu trình

Con trỏ head

- Hộp được gán nhãn "head" không phải là một nút:
  ListNodePtr head;
  - Một con trỏ trỏ tới một nút
  - Được thiết lập trỏ tới nút đầu tiên của danh sách
- head được sử dụng để "duy trì" vị trí bắt đầu của danh sách
- Cũng được sử dụng làm đối số của hàm

Ví dụ truy cập nút

- (*head).count = 12;
  - Thiết lập thành viên *count* của nút được trỏ bởi *head* bằng 12
- Toán tử thay thế, ->
  - Được gọi là "toán tử mũi tên"
  - Ký hiệu viết tắt kết hợp * và .
  - head->count = 12; //tương tự như trên
- cin >> head->item
  - Gán chuỗi nhập vào cho thành viên *item*

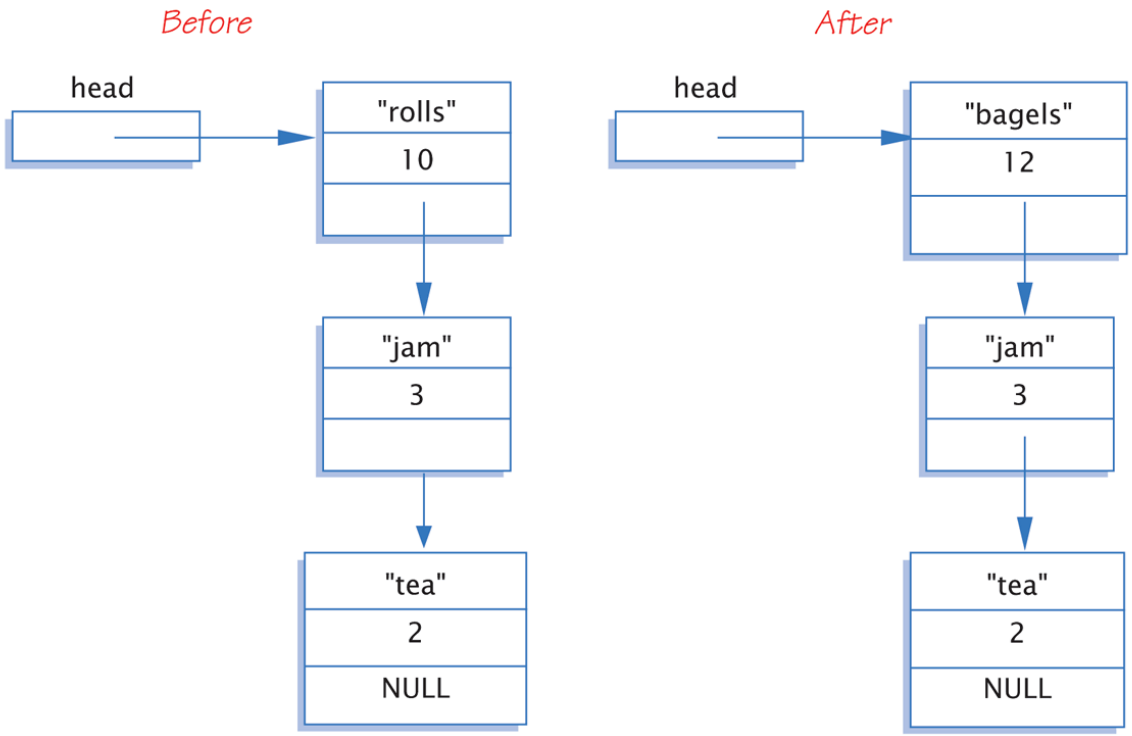
Ký hiệu kết thúc

- Sử dụng NULL cho con trỏ nút
  - Được xem là cờ hiệu cho các nút
  - Chỉ ra rằng không còn liên kết phía sau nút này
- Việc cung cấp ký hiệu kết thúc tương tự như cách chúng ta sử dụng mảng được nhập giá trị một phần

Truy cập dữ liệu nút

Display 17.2 Accessing Node Data

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



Danh sách liên kết

- Danh sách minh họa được gọi là danh sách liên kết
- Nút đầu tiên được gọi là *head*
  - Được trỏ tới bởi con trỏ có tên là *head*
- Nút cuối cũng đặc biệt
  - Biến con trỏ thành viên của nó là NULL
  - Dễ dàng kiểm tra việc kết thúc danh sách liên kết

Định nghĩa lớp danh sách liên kết

```
class IntNode
{
public:
        IntNode() { }
        IntNode(int theData, IntNOde* theLink)
                : data(theData), link(theLink) { }
        IntNode* getLink()          const           {return link;}
        int getData()               const           {return data;}
        void setData(int theData)                   {data = theData;}
        void setLink(IntNode* pointer)              {link=pointer;}
private:
        int data;
        IntNode *link;
};
typedef IntNode* IntNodePtr;
```

Lớp danh sách liên kết

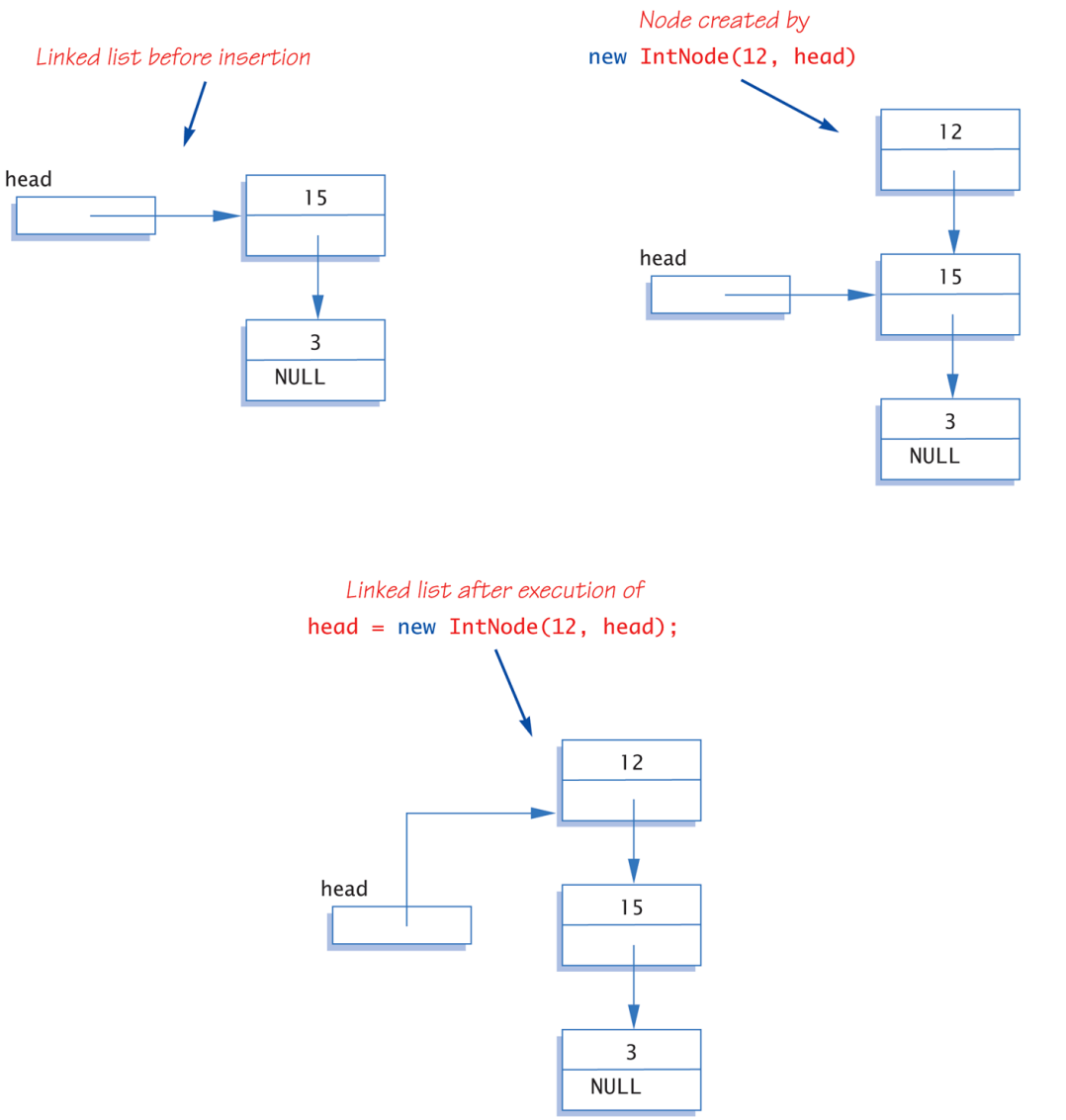
- Lưu ý tất cả các định nghĩa hàm thành viên là nội tuyến
  - Đủ nhỏ và đơn giản
- Lưu ý hàm tạo hai tham số
  - Cho phép tạo các nút với thành viên giá trị dữ liệu và thành viên liên kết
  - Ví dụ:
    IntNodePtr p2 = new IntNode(42, p1);

Tạo nút đầu tiên

- IntNodePtr head;
  - Khai báo biến con trỏ *head*
- head = new IntNode;
  - Cấp phát động nút mới
  - Nút đầu tiên trong danh sách, do vậy được gán cho head
- head->setData(3);
  head->setLink(NULL);
  - Thiết lập dữ liệu cho head
  - Thiết lập liên kết tới NULL vì nó là nút duy nhất

Thêm một nút vào đầu danh sách liên kết

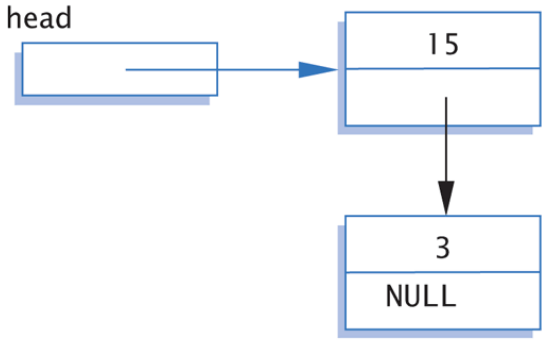
Display 17.3 Adding a Node to the Head of a Linked List



Lỗi thường gặp: Mất nút

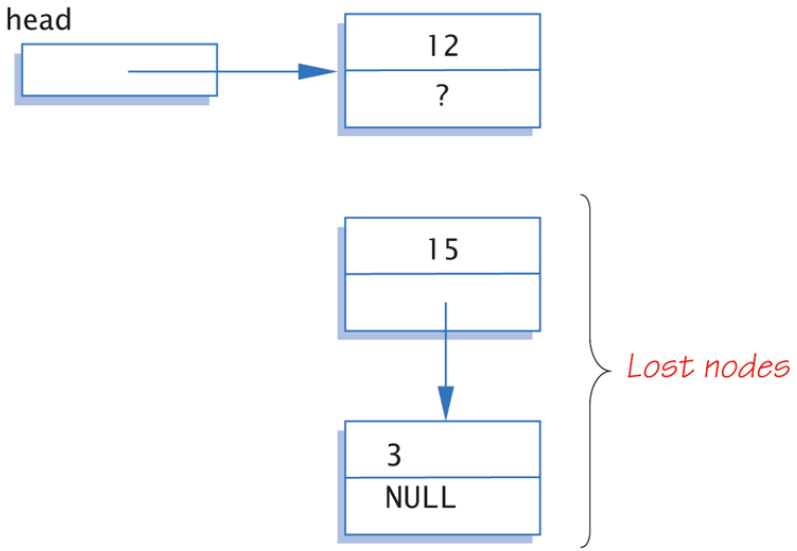
Display 17.5 **Lost Nodes**

Linked list before insertion



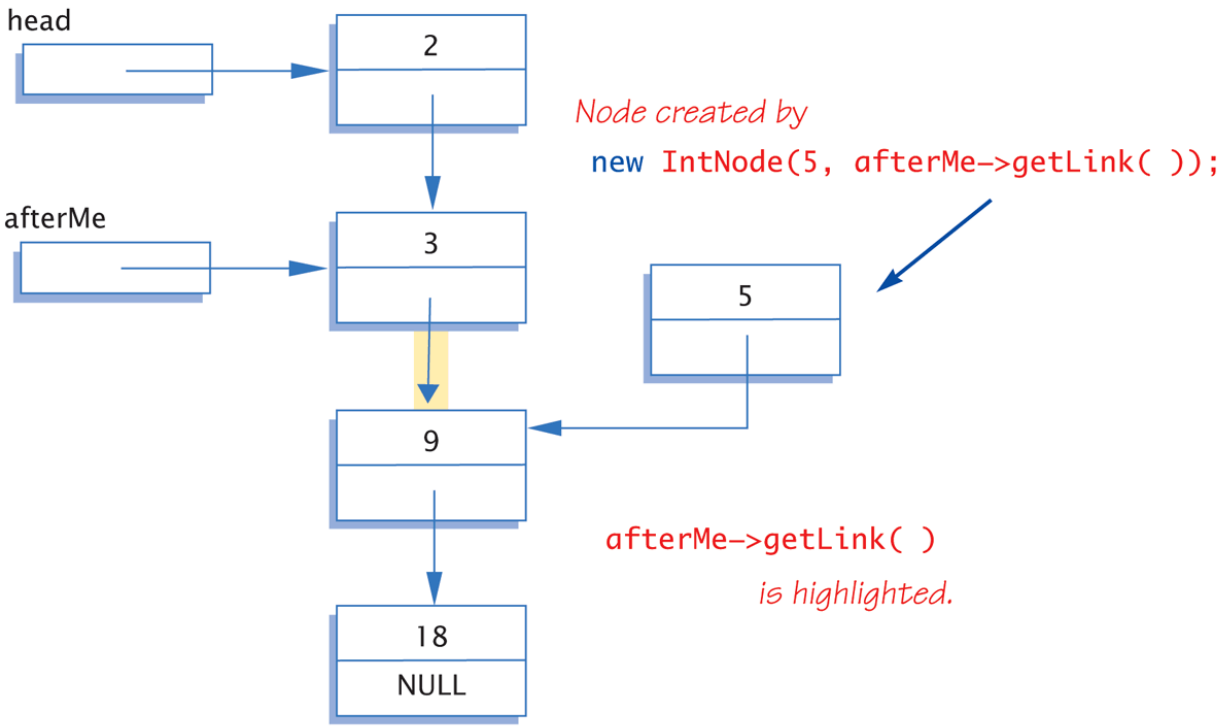
Situation after executing

```
head = new IntNode;
head->setData(theData);
```

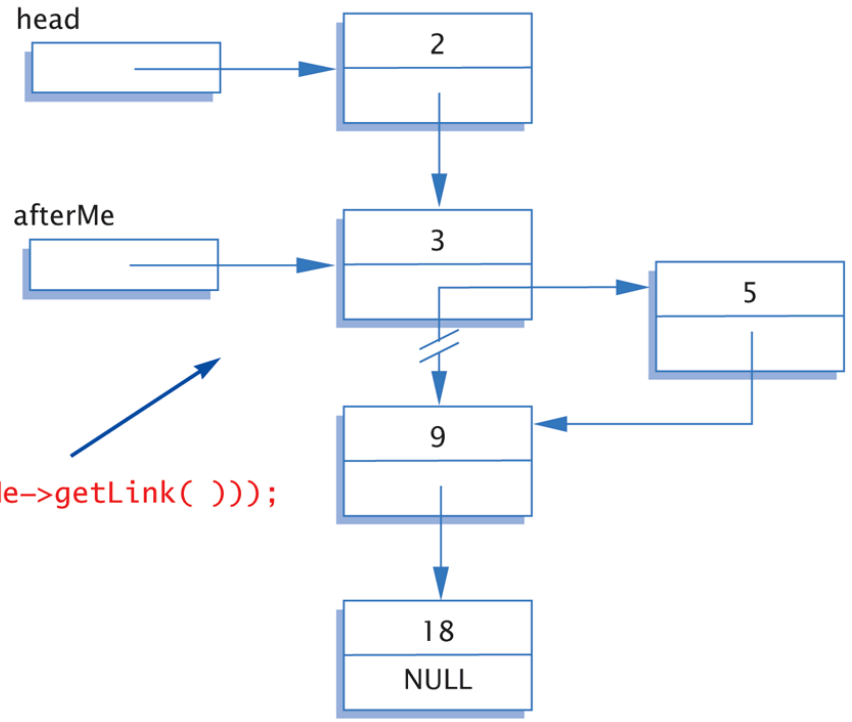


Chèn vào giữa danh sách liên kết

Display 17.6 **Inserting in the Middle of a Linked List**



Chèn vào giữa danh sách liên kết

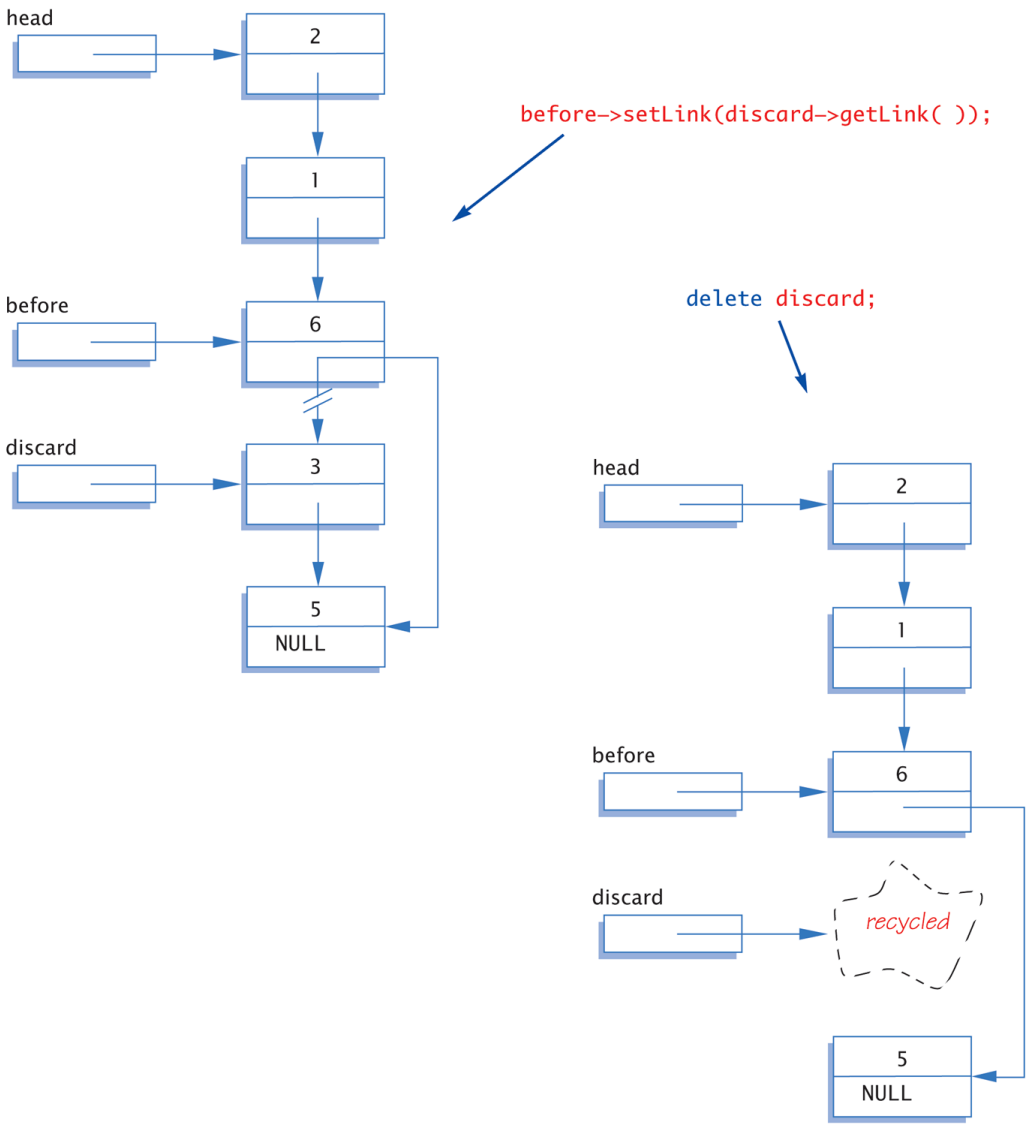


Final result of

```
afterMe->setLink(
            new IntNode(theData, afterMe->getLink( )));
```

Loại bỏ một nút

Display 17.7 **Removing a Node**



Tìm kiếm trên danh sách liên kết

- Hàm với hai đối số:
  IntNodePtr search(IntNodePtr head, int target);
  //Tiền điều kiện: con trỏ head trỏ tới đầu
  //danh sách liên kết. Con trỏ ở nút cuối cùng là NULL.
  //Nếu danh sách là rỗng, head là NULL
  //Trả về con trỏ trỏ tới nút đầu tiên chứa mục tiêu
  //Nếu không thấy, trả về NULL
- Duyệt danh sách
  - Tương tự như duyệt mảng

Mã giả cho hàm search

- while (here không trỏ tới nút mục tiêu hoặc nút cuối cùng)

```
{
        Làm here trỏ tới nút tiếp theo trong danh
        sách liên kết
}
if (nút here trỏ tới mục tiêu)
        return here;
else
        return NULL;
```

Thuật toán cho hàm search

- while (here->getData() != target &&
  here->getLink() != NULL)
  here = here->getLink();

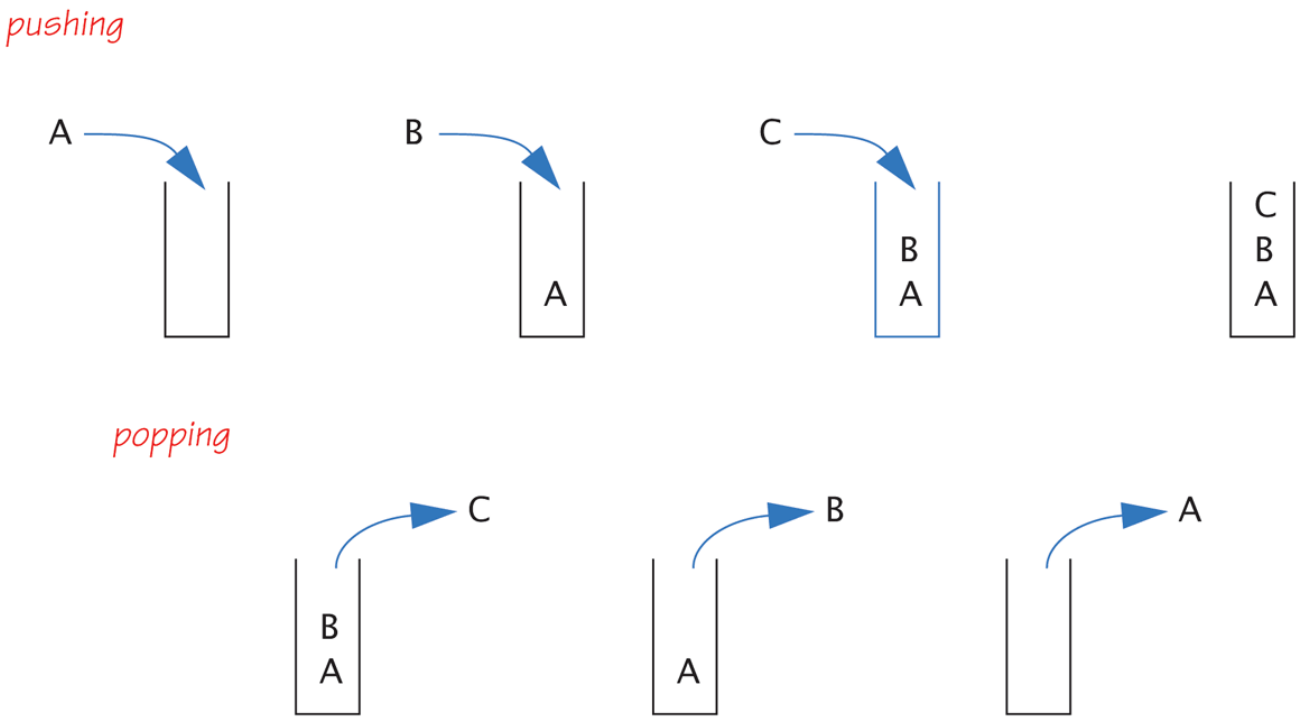
  if (here->getData() == target)
  return here;
  else
  return NULL;
- Phải xét thêm trường hợp đặc biệt khi danh sách rỗng

Ngăn xếp

- Cấu trúc dữ liệu ngăn xếp
  - Nhận dữ liệu theo trật tự ngược lại của cách lưu trữ
  - LIFO – vào sau/ra trước
  - Hình dung giống như cái hố trên mặt đất
- Ngăn xếp được sử dụng cho nhiều công việc
  - Theo dõi lời gọi hàm C++
  - Quản lý bộ nhớ
- Chúng ta sử dụng danh sách liên kết để thi hành ngăn xếp

Ngăn xếp

Display 17.12 **A Stack**



Giao diện cho lớp khuôn mẫu ngăn xếp

Display 17.13 **Interface File for a Stack Template Class**

```
1  //This is the header file stack.h. This is the interface for the class
2  //Stack, which is a template class for a stack of items of type T.
3  #ifndef STACK_H
4  #define STACK_H

5  namespace StackSavitch
6  {
7      template<class T>
8      class Node
9      {
10     public:
11         Node(T theData, Node<T>* theLink) : data(theData), link(theLink){}
12         Node<T>* getLink( ) const { return link; }
13         const T getData( ) const { return data; }
14         void setData(const T& theData) { data = theData; }
15         void setLink(Node<T>* pointer) { link = pointer; }
16     private:
17         T data;
18         Node<T> *link;
19     };
```

You might prefer to replace the parameter type T *with* `const T&`.

Giao diện cho lớp khuôn mẫu ngăn xếp

Display 17.13 **Interface File for a Stack Template Class**

```
20          template<class T>
21      class Stack
22      {
23    public:
24          Stack( );
25          //Initializes the object to an empty stack.

26          Stack(const Stack<T>& aStack);

27          Stack<T>& operator =(const Stack<T>& rightSide);

28          virtual ~Stack( );

29          void push(T stackFrame);
30          //Postcondition: stackFrame has been added to the stack.

31          T pop( );
32          //Precondition: The stack is not empty.
33          //Returns the top stack frame and removes that top
34          //stack frame from the stack.

35          bool isEmpty( ) const;
36          //Returns true if the stack is empty. Returns false otherwise.
37      private:
38          Node<T> *top;
39      };

40  }//StackSavitch
41  #endif //STACK_H
```

Copy constructor

The destructor destroys the stack and returns all the memory to the freestore.

Chương trình sử dụng lớp khuôn mẫu ngăn xếp

Display 17.14 Program Using the Stack Template Class

```
1  //Program to demonstrate use of the Stack template class.
2  #include <iostream>
3  #include "stack.h"
4  #include "stack.cpp"
5  using std::cin;
6  using std::cout;
7  using std::endl;
8  using StackSavitch::Stack;
```

(continued)

Chương trình sử dụng lớp khuôn mẫu ngăn xếp

Display 17.14 Program Using the Stack Template Class

```
 9  int main( )
10  {
11      char next, ans;

12      do
13      {
14          Stack<char> s;
15          cout << "Enter a line of text:\n";
16          cin.get(next);
17          while (next != '\n')
18          {
19              s.push(next);
20              cin.get(next);
21          }
```

Chương trình sử dụng lớp khuôn mẫu ngăn xếp

```
22          cout << "Written backward that is:\n";
23          while ( ! s.isEmpty( ) )
24              cout << s.pop( );
25          cout << endl;

26          cout << "Again?(y/n): ";
27          cin >> ans;
28          cin.ignore(10000, '\n');
29      }while (ans != 'n' && ans != 'N');

30      return 0;
31  }
```

SAMPLE DIALOGUE

```
Enter a line of text:
straw
Written backward that is:
warts
Again?(y/n): y
Enter a line of text:
I love C++
Written backward that is:
++C evol I
Again?(y/n): n
```

The `ignore` member of `cin` is discussed in Chapter 9. It discards input remaining on the line.

Ngăn xếp: Push và Pop

- Thêm mục dữ liệu vào ngăn xếp → push
  - Được xem là đưa dữ liệu vào trong ngăn xếp
  - Đi vào đỉnh của ngăn xếp
- Loại bỏ mục dữ liệu từ ngăn xếp → pop
  - Được xem là lấy dữ liệu khỏi ngăn xếp
  - Loại bỏ từ đỉnh của ngăn xếp

Hàng đợi

- Một cấu trúc dữ liệu phổ biến khác
  - Vận hành dữ liệu theo cách thức vào trước/ra trước (FIFO)
  - Các mục được chèn vào cuối danh sách
  - Các mục được loại bỏ từ đầu danh sách
- Biểu diễn dạng dòng điển hình
  - Giống như dòng máy đếm tiền ở ngân hàng, dòng xếp hàng ở rạp chiếu phim, vân vân.

Giao diện cho lớp khuôn mẫu hàng đợi

Display 17.16 **Interface File for a Queue Template Class**

```
1
2   //This is the header file queue.h. This is the interface for the class
3   //Queue, which is a template class for a queue of items of type T.
4   #ifndef QUEUE_H
5   #define QUEUE_H

6   namespace QueueSavitch
7   {
8       template<class T>
9       class Node
10      {
11      public:
12          Node(T theData, Node<T>* theLink) : data(theData), link(theLink){}
13          Node<T>* getLink( ) const { return link; }
14          const T getData( ) const { return data; }
15          void setData(const T& theData) { data = theData; }
16          void setLink(Node<T>* pointer) { link = pointer; }
17      private:
18          T data;
```

This is the same definition of the template class Node *that we gave for the stack interface in Display 17.13. See the tip "A Comment on Namespaces" for a discussion of this duplication.*

(continued)

Giao diện cho lớp khuôn mẫu hàng đợi

Display 17.16 Interface File for a Queue Template Class

```
19          Node<T> *link;
20      };
21      template<class T>
22      class Queue
23      {
24      public:
25          Queue( );
26          //Initializes the object to an empty queue.
27          Queue(const Queue<T>& aQueue);
28          Queue<T>& operator =(const Queue<T>& rightSide);
29          virtual ~Queue( );
30
```

You might prefer to replace the parameter type T *with* `const T&`.

Copy constructor

The destructor destroys the queue and returns all the memory to the freestore.

Giao diện cho lớp khuôn mẫu hàng đợi

```
31          void add(T item);
32          //Postcondition: item has been added to the back of the queue.


33          T remove( );
34          //Precondition: The queue is not empty.
35          //Returns the item at the front of the queue
36          //and removes that item from the queue.


37          bool isEmpty( ) const;
38          //Returns true if the queue is empty. Returns false otherwise.
39      private:
40          Node<T> *front;//Points to the head of a linked list.
41                         //Items are removed at the head
42          Node<T> *back;//Points to the node at the other end of the linked list.
43                        //Items are added at this end.
44      };


45  }//QueueSavitch


46  #endif //QUEUE_H
```

Chương trình sử dụng lớp khuôn mẫu hàng đợi

```
12      do
13      {
14          Queue<char> q;
15          cout << "Enter a line of text:\n";
16          cin.get(next);
17          while (next != '\n')
18          {
19              q.add(next);
20              cin.get(next);
21          }


22          cout << "You entered:\n";
23          while ( ! q.isEmpty( ) )
24              cout << q.remove( );
25          cout << endl;

26          cout << "Again?(y/n): ";
27          cin >> ans;
28          cin.ignore(10000, '\n');
29      }while (ans != 'n' && ans != 'N');
```

Lớp bạn

- Việc sử dụng liên tục các hàm truy cập và biến đổi getLink và setLink:
  - Có đôi chút trở ngại
  - So sánh với việc làm cho dữ liệu public?
- Sử dụng lớp bạn
  - Làm cho lớp khuôn mẫu hàng đợi là bạn của lớp khuôn mẫu nút
  - Tất cả các thành viên liên kết private cung cấp trực tiếp trong các hàm thành viên của lớp hàng đợi

Khai báo tiến

- Mối quan hệ bạn bè đòi hỏi các lớp tham chiếu lẫn nhau
  - Biểu diễn vấn đề
  - Làm thế nào để cả hai có thể được khai báo cùng một lúc
- Đòi hỏi khai báo tiến
  - Tiêu đề lớp này nằm bên trong lớp kia
    class Queue; //Khai báo tiến.
  - Thông báo “lớp Queue tồn tại”

Biến lặp (iterator)

- Cấu trúc giúp luân chuyển trên dữ liệu
  - Giống như “duyệt”
  - Cho phép các hành động tùy ý trên dữ liệu
- Con trỏ thường được sử dụng làm biến lặp
  - Đã sử dụng khi thi hành danh sách liên kết

Con trỏ làm biến lặp

- Danh sách liên kết: cấu trúc dữ liệu nguyên mẫu
- Con trỏ: ví dụ nguyên mẫu về biến lặp
  - Con trỏ được sử dụng làm biến lặp bằng việc di chuyển qua từng nút của danh sách liên kết bắt đầu từ head
  - Ví dụ:

```
Node_Type *iterator;
for (iterator = Head; iterator != NULL;
                                iterator=iterator->Link)
    Do_Action
```

Lớp iterator

- Đa năng hơn con trỏ
- Các thao tác được nạp chồng điển hình

  ++ tiến biến lặp tới mục tiếp theo
  -- lùi biến lặp về mục phía trước
  == so sánh các biến lặp
  != so sánh khác
  * truy cập một mục

- Lớp cấu trúc dữ liệu có các thành viên:
  begin(): trả biến lặp về mục đầu tiên trong cấu trúc
  end(): trả về biến lặp để kiểm tra xem có đang ở mục cuối cùng trong cấu trúc không

Ví dụ lớp iterator

- Luân chuyển trên cấu trúc dữ liệu tên là *ds*:

```
for (i=ds.begin();i!=ds.end();i++)
      process *i  //*i là mục dữ liệu hiện thời
```

- i là tên của biến lặp

Tóm tắt

- Nút là đối tượng cấu trúc hoặc đối tượng lớp
  - Một hoặc nhiều thành viên là con trỏ
  - Các nút được liên kết bởi các con trỏ thành viên
    - Tạo ra các cấu trúc có thể mở rộng hoặc co lại ở thời điểm chạy
- Danh sách liên kết
  - Danh sách các nút trong đó mỗi nút trỏ tới nút tiếp theo
- Con trỏ NULL được sử dụng làm ký hiệu kết thúc danh sách liên kết

Tóm tắt

- Ngăn xếp là cấu trúc dữ liệu LIFO
- Hàng đợi là cấu trúc dữ liệu FIFO
- Cấu trúc biến lặp cho phép luân chuyển trên các mục dữ liệu trong một cấu trúc dữ liệu cho trước