Ngôn ngữ lập trình

Bài 10:
Các Kiểu Dữ Liệu Trừu Tượng: Danh sách liên kết, Ngăn xếp, Hàng đợi

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Bộ Môn Công Nghệ Phần Mềm – Khoa CNTT
Trường Đại Học Thủy Lợi

Nội dung

1. Các nút (Nodes) và Danh sách liên kết
    1. Tạo, tìm kiếm
2. Ứng dụng danh sách liên kết
    1. Ngăn xếp (Stack),
    2. Hàng đợi (Queue)
3. Iterators
    1. Con trỏ như iterators
4. Cây (Trees)

Bài giảng có sử dụng hình vẽ trong cuốn sách "*Absolute C++*. W. Savitch, Addison Wesley, 2002"

Giới thiệu

- Danh sách liên kết
  - Được xây dựng sử dụng con trỏ
  - Tăng giảm kích thước trong thời gian chạy
- Cây cũng sử dụng con trỏ
- ***Con trỏ là xương sống của những cấu trúc này***
  - Sử dụng biến động
- Thư viện mẫu chuẩn (STL)
  - Có những phiên bản định nghĩa sẵn của một vài cấu trúc

Cách tiếp cận

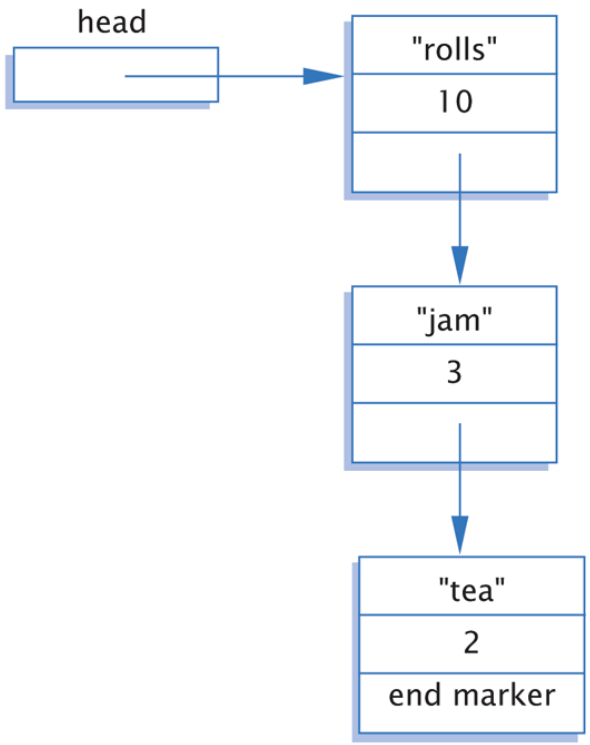
- Có 3 cách để xử lý những cấu trúc dữ liệu này
  1. Cách tiếp cận C-style: sử dụng hàm và cấu trúc toàn cục với mọi thứ đều public
  2. Sử dụng lớp với các biến thành viên private và các hàm accessor – mutator
  3. Sử dụng lớp bạn
- ***Danh sách liên kết*** sử dụng phương thức 1 (hoặc 2)
- ***Ngăn xếp, Hàng đợi*** sử dụng phương thức 2
- ***Cây*** sử dụng phương thức 3

Nút và danh sách liên kết

- Danh sách liên kết
  - Một ví dụ đơn giản của "cấu trúc dữ liệu động"
  - Bao gồm nhiều nút
- Mỗi nút là một biến kiểu cấu trúc hoặc đối tượng của lớp (có thể tạo tự động với lệnh *new*)
  - Nút cũng bao gồm con trỏ trỏ tới những nút khác
  - Cung cấp "sự 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;
```

- Chú ý sự quay vòng (circularity)

Con trỏ head

- Đối tượng với nhãn "head" không phải là một nút: *ListNodePtr head;*
  - Là một con trỏ đơn giản tới một nút
  - Trỏ tới nút đầu tiên trong danh sách
- *Head* được sử dụng để lưu trữ vị trí đầu tiên trong danh sách
- Cũng được sử dụng như đối số truyền vào hàm

Ví dụ về truy cập nút

- *(*head).count = 12;*
  - Đặt biến thành viên *count* của nút trỏ bởi con trỏ head bằng 12
- Toán tử thay thế ->
  - Được gọi là toán tử mũi tên (*arrow operator*)
  - Kí hiệu viết tắt là sự kết hợp của hai toán tử * và .
  - Viết lại câu lệnh trên bằng: *head->count=12;*
- *cin>>head->item*:
  - Gắn chuỗi nhập vào cho biến thành viên *item*

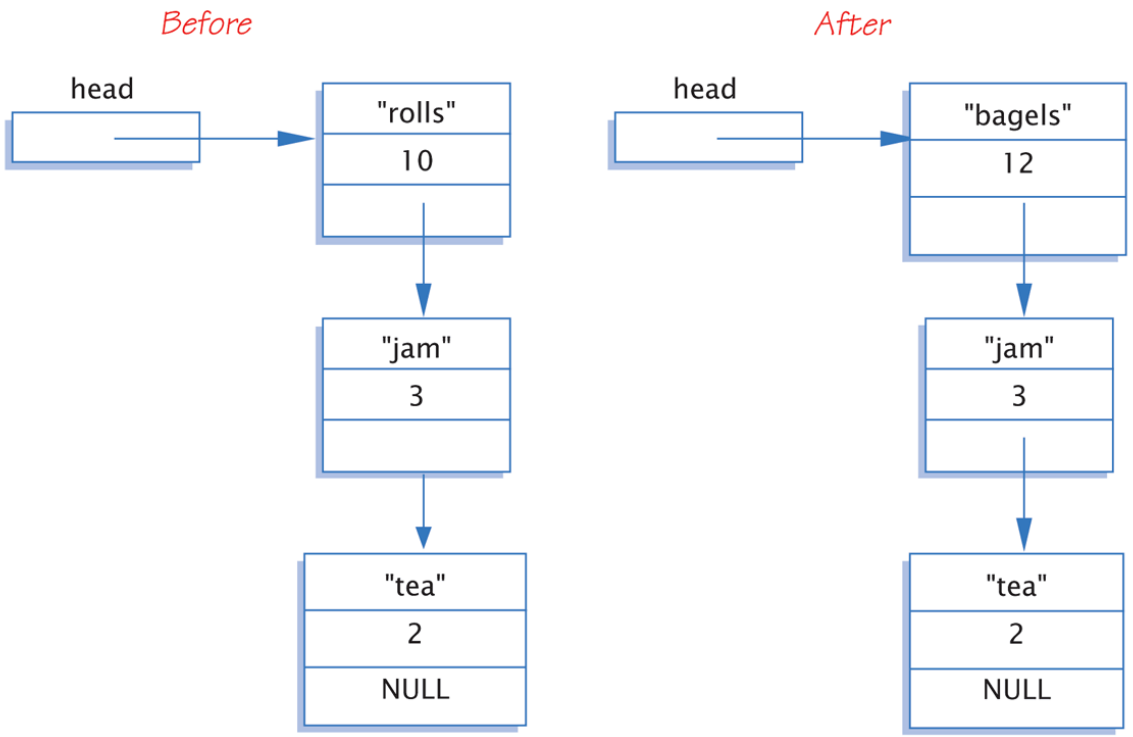
Dấu hiệu kết thúc (end markers)

- Sử dụng NULL cho con trỏ nút
  - Được xem như "*lính canh*" (sentinel) cho các nút
  - Chỉ định rằng không còn liên kết sau nút này
- Cung cấp dấu hiệu kết thúc

Truy cập dữ liệu trong nút

Display 17.2 **Accessing Node Data**

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



Danh sách liên kết

- Nút đầu tiên gọi là *head*
  - Được trỏ tới bởi con trỏ tên là *head*
- Nút cuối cùng cũng đặc biệt
  - Biến con trỏ thành viên của nó là NULL
  - Dễ dàng kiểm tra kết thúc của 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()          {return link;}
            int getData()               {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

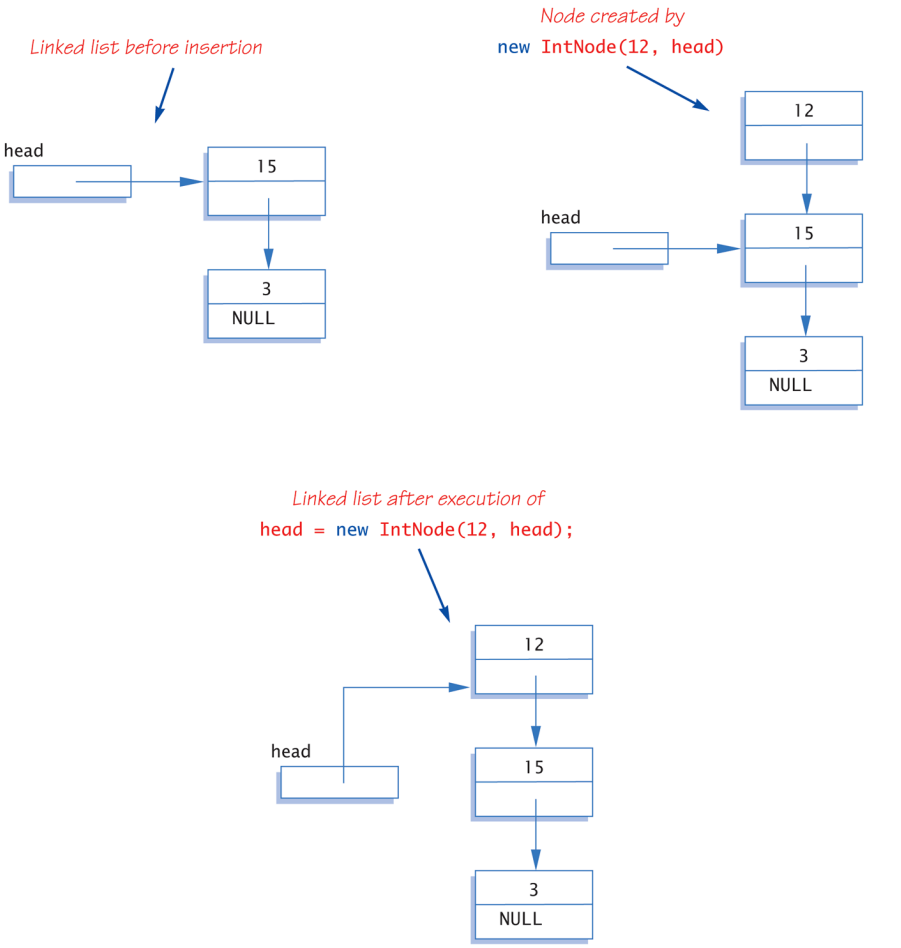
- Chú ý hàm khởi tạo 2 tham số
  - Cho phép tạo các nút với dữ liệu riêng biệt và thành viên liên kết được chỉ định
  - Ví dụ: *IntNodePtr p2 = new IntNode(42,p1);*

Tạo nút đầu tiên

- *IntNodePtr head;*
  - Khai báo biến con trỏ *head*
- *head = new IntNode;*
  - Cấp phát động cho nút mới
  - Nút đầu tiên đã trong danh sách và được gán cho *head*
- *head->setData(3);*
  head->setLink(NULL);
  - Đặt dữ liệu cho nút head
  - Đặt liên kết của nút đầu là NULL, bởi chúng ta mới chỉ có 1 nút!

Minh họa thêm một nút cho head của danh sách liên kết

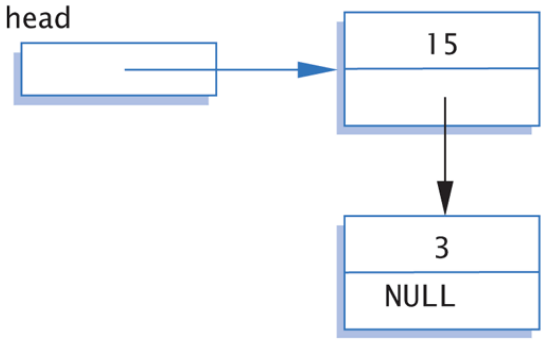
Display 17.3 Adding a Node to the Head of a Linked List



Trường hợp bị mất nút

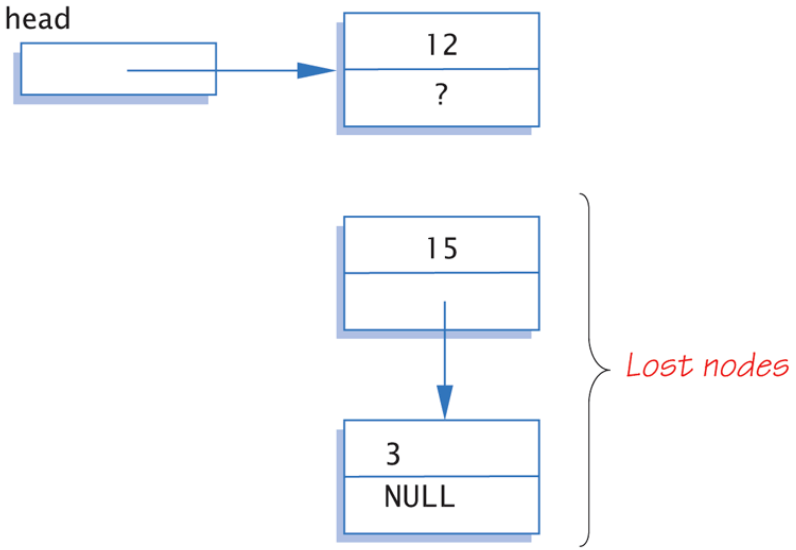
Display 17.5 **Lost Nodes**

Linked list before insertion



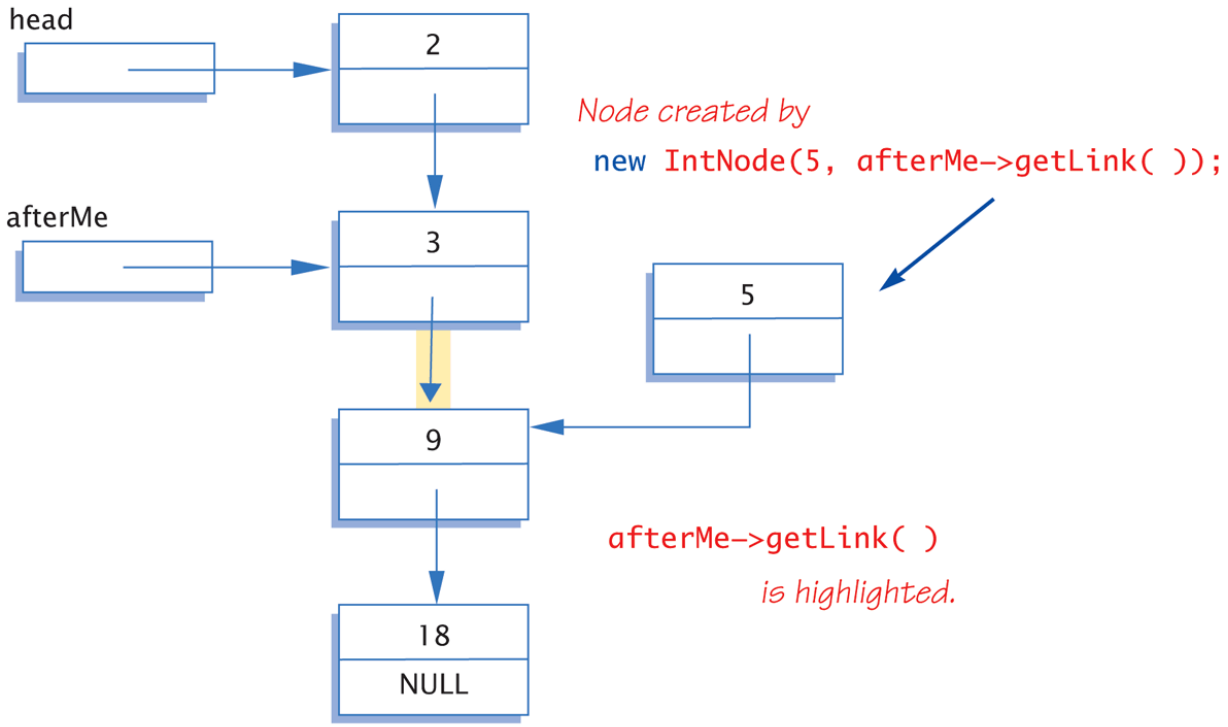
Situation after executing

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

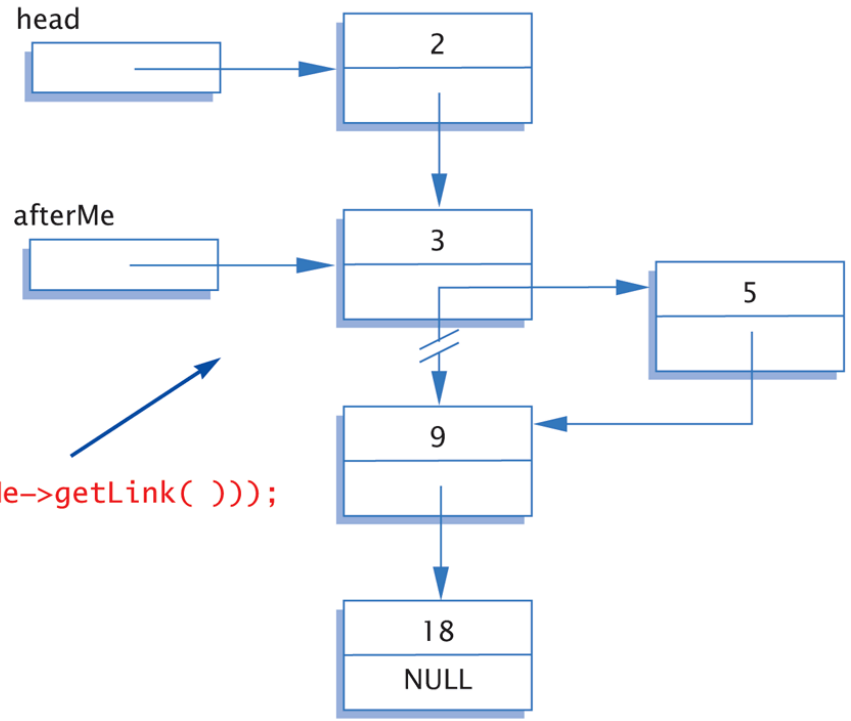


Chèn một nút vào giữa danh sách liên kết (1/2)

Display 17.6 Inserting in the Middle of a Linked List



Chèn một nút vào giữa danh sách liên kết (2/2)

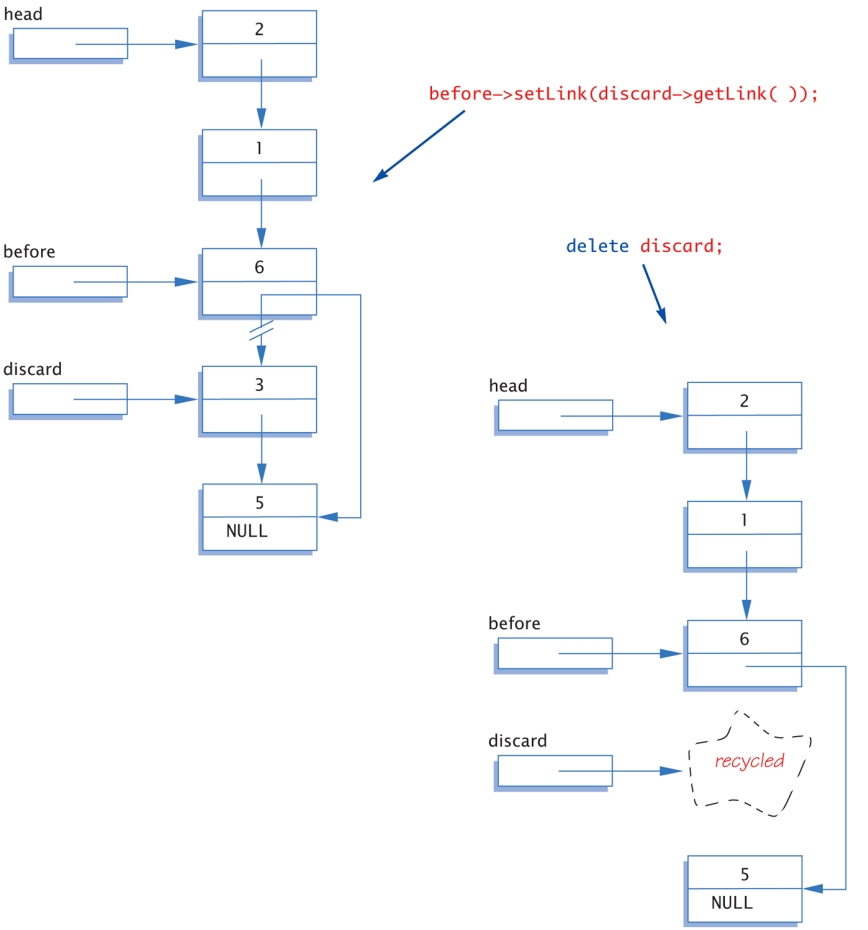


Final result of

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

Xóa một nút

Display 17.7 **Removing a Node**



Tìm kiếm trong danh sách liên kết

- Hàm với hai đối số

  IntNodePtr search(IntNodePtr head, int target);

  // Điều kiện trước: con trỏ *head* trỏ tới đầu danh sách liên kết

  // Con trỏ của nút cuối cùng là NULL.

  // Nếu danh sách rỗng, *head* là NULL

  // Trả về con trỏ tới nút đầu tiên chứa giá trị *target*

  // Nếu không tìm thấy, trả về NULL

- Đơn giản là duyệt qua (traversal) danh sách
  - Giống như duyệt mảng

Mã giả cho hàm tìm kiếm (Pseudocode)

while (con trỏ *here* chưa trỏ tới nút đích hoặc nút cuối)
{
 dịch chuyển con trỏ *here* tới nút tiếp theo trong danh sách
}
if (nút được trỏ bởi *here* chứa giá trị đích)
    return con_trỏ;
else
    return NULL;

Thuật toán cho hàm tìm kiếm

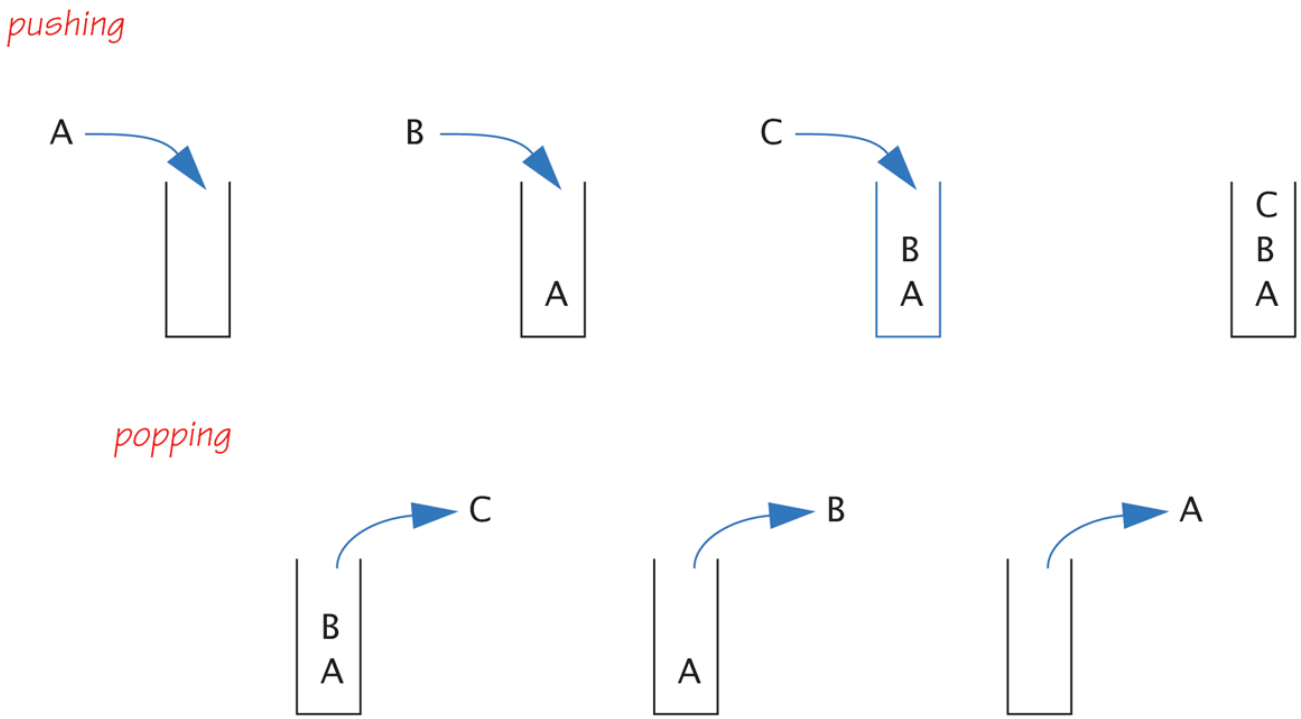
```
while (here->getData() != target &&
                here->getLink() != NULL)
        here = here->getLink();

if (here->getData() == target)
        return here;
else
        return NULL;
```

Ngăn xếp (Stack)

- Cấu trúc dữ liệu ngăn xếp
  - Lấy ra dữ liệu theo thứ tự ngược với khi được lưu trữ
  - LIFO – Last-in/First-out (vào sau, ra trước)
- Ngăn xếp được sử dụng cho nhiều tác vụ
  - Truy vết những lời gọi hàm
  - Quản lý bộ nhớ
- Sử dụng danh sách liên kết để cài đặt ngăn xếp
- Thao tác **Push**: thêm dữ liệu vào ngăn xếp
  - Đẩy vào từ vị trí đầu tiên của ngăn xếp
- Thao tác **Pop**: lấy dữ liệu ra khỏi ngăn xếp
  - Lấy ra từ vị trí đầu tiên của ngăn xếp

Minh họa ngăn xếp

Display 17.12 **A Stack**



File giao diện của một lớp khuôn mẫu ngăn xếp (1/2)

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&`.

File giao diện của một lớp khuôn mẫu ngăn xếp (2/2)

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.

Driver lớp khuôn mẫu ngăn xếp chương trình mẫu (1/3)

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)

Driver lớp khuôn mẫu ngăn xếp chương trình mẫu (2/3)

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          }
```

Driver lớp khuôn mẫu ngăn xếp chương trình mẫu (3/3)

```
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.

Hàng đợi (Queue)

- Một cấu trúc dữ liệu phổ biến khác
  - Xử lý dữ liệu theo cách First-in/First-out (vào trước/ra trước - FIFO)
  - Dữ liệu được chèn vào cuối danh sách
  - Dữ liệu được lấy ra từ đầu danh sách

File giao diện của một lớp khuôn mẫu hàng đợi (1/3)

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)

File giao diện của một lớp khuôn mẫu hàng đợi (2/3)

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.

File giao diện của một lớp khuôn mẫu hàng đợi (3/3)

```
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
```

Driver lớp khuôn mẫu giao diện: chương trình mẫu

```
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');
```

Iterators

- Xây dựng để duyệt dữ liệu
  - Cho phép bất kỳ hành động nào được yêu cầu trên dữ liệu
- Con trỏ thường được sử dụng như iterator
- Nhớ lại: danh sách liên kết – một cấu trúc dữ liệu nguyên mẫu/điển hình (prototypical)
- Con trỏ: ví dụ điển hình của iterator, duyệt qua danh sách từ vị trí đầu tiên

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

Lớp iterator

- Linh hoạt hơn so với con trỏ
- Các toán tử được nạp chồng điển hình như
  - ++ dịch chuyển tiến iterator sang vị trí tiếp theo
  - -- dịch lùi iterator về vị trí trước
  - == so sánh bằng với iterator
  - != so sánh khác với iterator
  - * truy cập một vị trí/mục
- Lớp cấu trúc dữ liệu sẽ có 2 hàm thành viên
  - *begin()*:trả iterator về vị trí đầu tiên trong cấu trúc
  - *end()*:kiểm tra iterator có ở vị trí cuối không

Ví dụ lớp iterator

- Duyệt qua cấu trúc dữ liệu có tên *ds*:

```
for (i=ds.begin();i!=ds.end();i++)
        process *i  //*i is current data item
```

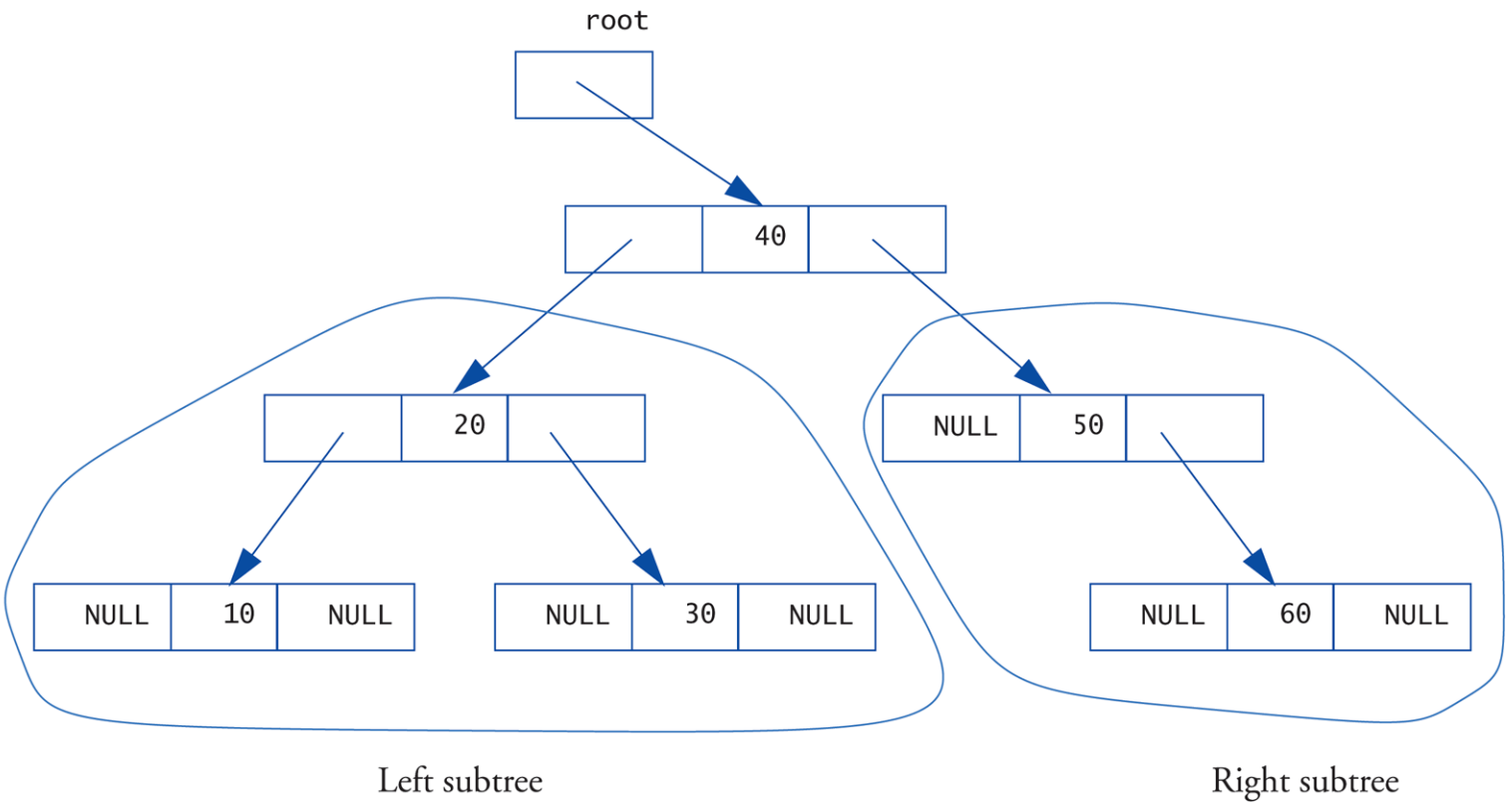
- *i* là một iterator

Giới thiệu về cây

- Cây là một cấu trúc dữ liệu phức tạp
- Giới thiệu những điều cơ bản trong bài học này
  - Xây dựng, thao tác với cây
  - Sử dụng nút và con trỏ
- Nhớ lại danh sách liên kết: các nút chỉ có một con trỏ tới vị trí nút tiếp theo
- Cây có 2 con trỏ, hoặc thậm chí nhiều hơn, tới những nút khác

Cây nhị phân (1/2)

Display 17.23 **A Binary Tree**



Cây nhị phân (2/2)

```
class IntTreeNode
{
public:
    IntTreeNode(int theData, IntTreeNode* left, IntTreeNode* right)
        : data(theData), leftLink(left), rightLink(right){}
private:
    int data;
    IntTreeNode *leftLink;
    IntTreeNode *rightLink;
};


IntTreeNode *root;
```

Thuộc tính của cây

- Đường đi
  - Từ đỉnh tới một nút bất kỳ
  - Không quay vòng, đi theo con trỏ sẽ đến vị trí cuối cùng
- Chú ý ở đây mỗi nút có 2 liên kết
  - Được gọi là cây nhị phân (***binary tree***)
  - Kiểu phổ biến nhất của cây
- Nút gốc (root node)
  - Tương tự như *head* của danh sách liên kết
- Nút lá (leaf nodes)
  - Cả hai biến liên kết đều là NULL (không có cây con)

Cây và đệ quy

- Thấy rằng: cây có cấu trúc đệ quy
- Mỗi cây có 2 cây con
  - Mỗi cây con lại có hai cây con của nó …
- Có thể sử dụng các thuật toán đệ quy để tìm kiếm!

Xử lý cây – 3 phương pháp (Tree processing)

1. Xử lý preorder (preorder processing)
   - Xử lý dữ liệu ở nút gốc
   - Xử lý cây con bên trái
   - Xử lý cây con bên phải
2. Xử lý in-order
   - Xử lý cây con bên trái
   - Xử lý dữ liệu ở nút gốc
   - Xử lý cây con bên phải
3. Xử lý post-order
   - Xử lý cây con bên trái
   - Xử lý cây con bên phải
   - Xử lý dữ liệu ở nút gốc

Lưu trữ cây

- Ví dụ của chúng ta lưu giá trị theo một cách đặc biệt
  - Quy luật lưu trữ dữ liệu trong cây nhị phân
    - Dữ liệu ở cây con bên trái nhỏ hơn dữ liệu gốc
    - Dữ liệu ở cây con bên phải lớn hơn dữ liệu gốc
    - 2 quy tắc trên được áp dụng đệ quy với từng cây con
- Cây sử dụng cơ chế lưu trữ:
  - Được gọi là cây nhị phân tìm kiếm (Called binary search tree - BST)
  - Duyệt cây:
    Inorder → values "in order"
    Preorder → "prefix" notation
    Postorder → "postfix" notation

Tóm tắt

- Nút là cấu trúc hoặc đối tượng của lớp
  - Một hoặc nhiều thành viên là con trỏ
  - Các nút được kết nối bởi con trỏ thành viên
  - Tạo nên cấu trúc mà kích thước có thể thay đổi trong lúc chạy chương trình
- Danh sách liên kết
  - Danh sách các nút mà mỗi nút trỏ tới phần tử tiếp theo
- Kết thúc của danh sách liên kết với con trỏ NULL
- Ngăn xếp có cấu trúc LIFO
- Hàng đợi có cấu trúc FIFO
- Xây dựng iterator cho phép duyệt qua phần tử dữ liệu trong cấu trúc dữ liệu
- Cấu trúc dữ liệu cây
  - Mỗi nút có nhiều hơn 2 con trỏ thành viên
  - Mỗi con trỏ trỏ tới nút khác hoặc cây con khác
- Cây nhị phân tìm kiếm
  - Một vài quy luật lưu trữ đặc biệt giúp cho việc tìm kiếm nhanh hơn

Giáo trình Tham khảo

- **Giáo trình chính: W. Savitch, *Absolute C++*, Addison Wesley, 2002**
- Tham khảo:
  - A. Ford and T. Teorey, *Practical Debugging in C++*, Prentice Hall, 2002
  - Nguyễn Thanh Thủy, *Kĩ thuật lập trình C++*, NXB Khoa học và Kĩ Thuật, 2006