VE281

Data Structures and Algorithms

Generic Programming and Stacks

Review

- Linked List Optimization
- **getSize()**
  - Add a size data member
- **appendNode()**
  - Double-ended linked list
- **removeNode()**
  - Doubly-linked list
  - Remove the previous node
  - Copy the value from the next node and delete the next node
- Reverse a Linked List
- Speeding-up Allocation/De-Allocation: Free List

Outline

- Generic Programming
- Stacks

Traversing Linked Lists and Arrays

- Print all the elements in an array or a linked list requires traversal.

```
void Array::print(){
  int *curr = data;
  while (curr!=data+size){
    cout << *curr << endl;
    curr++;
  }
}!
```

```
void LinkedList::print(){
  node *curr = first;
  while (curr != NULL){
    cout << curr->value
      << endl;
    curr = curr->next;
  }
}
```

Can we write code that works for both an array and a linked list?

Generic Programming

- Generic programming is a data-type independent way of programming.

- The Generic Programming process focuses on finding commonality among similar implementations of the same algorithm, then providing suitable abstractions so that a single, generic algorithm can cover many concrete implementations.

Generic Programming

Example

```
// Generic code to print a series of items
template<class T>
void genPrint(T itBegin, T itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
// new print code for Arrays
void Array::print(){
  genPrint(this->begin(), this->end());
}
// new print code for Linked Lists
void LinkedList::print(){
  genPrint(this->begin(), this->end());
}
```

Iterator

```
// Generic code to print a series of items
template<class Iter>
void genPrint(Iter itBegin, Iter itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
```

We rename the type variable **T** as **Iter**

- The above code is based on an "iterative abstraction", or "iterator".
- Iterator allows its user to iterate over the members of a container using a set of operators (at least the increment (++) and dereference (*) operators).
- In principle, any container class can support an iterator.

Iterators for Linked Lists

- Suppose we define an iterator class for linked list
  `class ListIter { … };`

- Suppose that **`LinkedList::begin()`** and

```
void LinkedList::print(){
  genPrint(this->begin(), this->end());
}
```

Iterators for Linked Lists

```
void LinkedList::print(){
  genPrint(this->begin(), this->end());
}
```

ListIter ty

```
template<class Iter>
void genPrint(Iter itBegin, Iter itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
```

- What operations of **ListIter** are required?
  - Relation **!=**
  - Dereference *****
  - Increment **++**

Iterators for Linked Lists

```
class ListIter {
  node *ptr; // Point to the current node in the
             // linked list.
public:
  ListIter(node *n = NULL):ptr(n){}

  bool operator!=(const ListIter &iter)
  { return ptr != iter.ptr; }

  int& operator*() // dereferencing operator *
  { return ptr->value; }

  void operator++(int) // itr++
  { ptr = ptr->next; }
  void operator++() // ++itr
  { ptr = ptr->next; }
};
```

Prefix and Postfix Increment Operator

- These operators change the state of the object, so we prefer to make them member function of the class.

- To distinguish between the prefix (++x) and postfix (x++) increment operators, we let the postfix take an extra unused parameter of type int:

```
void operator++(int); // postfix
```

  - Do not use this extra parameter in the implementation of postfix operator! Its sole purpose is to distinguish postfix version from prefix version.
  - The **int** parameter is not used, so we do not give it a name

Iterators for Linked Lists

- Next add these to the **LinkedList** class:

```
ListIter LinkedList::begin() {
  return ListIter(first);
}
ListIter LinkedList::end() {
  return ListIter();
}
```

Question: Why does **end()** return an empty **ListIter**?

```
template<class Iter>
void genPrint(Iter itBegin, Iter itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
```

Iterators for Arrays

- Do we need to define iterators for arrays?

```
void Array::print(){
  genPrint(this->begin(), this->end());
}
```

```
template<class Iter>
void genPrint(Iter itBegin, Iter itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
```

Iterator for Arrays

```
void Array::print(){
  genPrint(this->begin(), this->end());
}
```

```
template<class Iter>
void genPrint(Iter itBegin, Iter itEnd){
  while (itBegin != itEnd){
    cout << *itBegin << endl;
    itBegin++;
  }
}
```

- We don't need to define iterators for arrays.
- Define **begin()** and **end()** for **Array** class as

```
int *Array::begin() { return data; }
// data is the beginning of the array
```

Iterator for Arrays

```
int *Array::begin() { return data; }
// data is the beginning of the array
int *Array::end() { return data+size; }
```

- Array iterators are just regular pointers.
- Regular pointers already have support for **!=**, **++** and *****.

Outline

- Generic Programming
- Stacks

Stacks

- A “pile” of objects where new object is put on top of the pile and the top object is removed first.
  - LIFO access: last in, first out.
  - Restricted form of a linear list: insert and remove only at the front of the list.

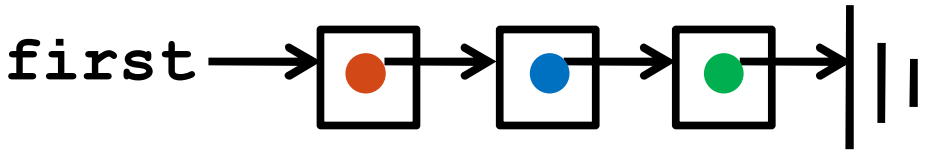
Methods of Stack

- **size()**: number of elements in the stack.
- **isEmpty()**: checks if stack has no elements.
- **push(Object o)**: add object **o** to the top of stack.
- **pop()**: remove the top object if stack is not empty; otherwise, throw **stackEmpty**.
- **Object &top()**: return a reference to the top element.

Stacks Using Arrays

- Maintain an integer **size** to record the size of the stack.
- **size()**: **return size;**
- **isEmpty()**: **return (size == 0);**
- **push(Object o)**: add object **o** to the end of the array and increment **size**. Allocate more space if necessary.
- **pop()**: If **isEmpty()**, throw **stackEmpty;** otherwise, decrement **size.**
- **Object &top()**: return a reference to the top element **Array[size-1]**

Stacks Using Linked Lists

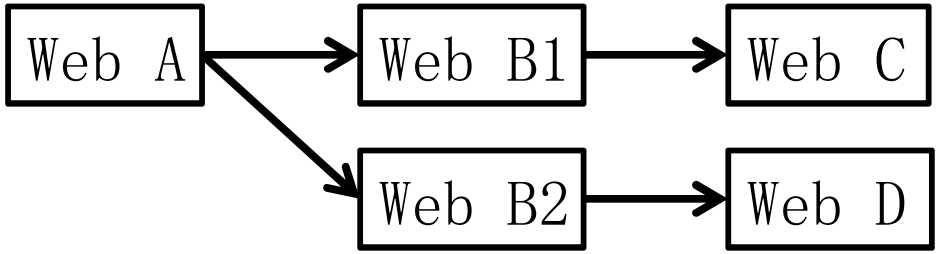


- **size()**: **LinkedList::size();**
- **isEmpty()**: **LinkedList::isEmpty();**
- **push(Object o)**: insert object at the beginning **LinkedList::insertFirst(Object o);**
- **pop()**: remove the first node **LinkedList::removeFirst();**
- **Object &top()**: return a reference to the object stored in the first node.

Application of Stacks

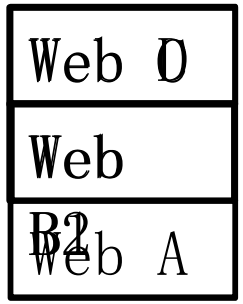
- Function calls in C++
- Web browser’s “back” feature
- Parentheses Matching

Web Browser's "back" Feature



Visiting order

- Web A
- Web B1
- Web C
- Back (to Web B1)
- Back (to Web A)
- Web B2
- Web D



Parentheses Matching

- Output pairs (u, v) such that the left parenthesis at position u is matched with the right parenthesis at v.

```
( ( a + b ) * c + d - e ) / ( f + g )
0 1 2 3 4 5 6 7 8 9 10  12  14  16  18
```

  - Output is: (1, 5); (0, 12); (14, 18);

```
( a + b ) ) * ( ( c + d )
0 1 2 3 4 5 6 7 8 9 10  12
```

  - Output is

    (0, 4);

    Right parenthesis at 5 has no matching left parenthesis;

    (8, 12);

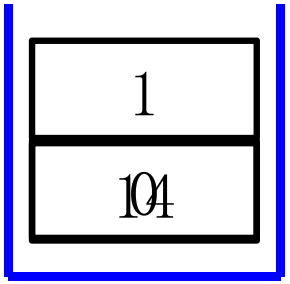
Parentheses Matching

```
( ( a + b ) * c + d - e ) / ( f + g )
0 1 2 3 4 5 6 7 8 9 10  12  14  16  18
```

- Scan expression from left to right.
- When a left parenthesis is encountered, push its position to the stack.
- When a right parenthesis is encountered, pop the top position from the stack, which is the position of the matching left parenthesis.
  - If the stack is empty, the right parenthesis is not matched.
- If string is scanned over but the stack is not empty, there are not-matched left

Parentheses Matching

```
( ( a + b ) * c + d - e ) / ( f + g )
0 1 2 3 4 5 6 7 8 9 10  12  14  16  18
```



(1, 5) (0, 12)(14, 18)

Campus Talk of Cadence

- Cadence Design Systems, Inc.: An **Electronic Design Automation (EDA)** Company
  - Produce **software** for designing **VLSI circuits**, e.g., Virtuoso

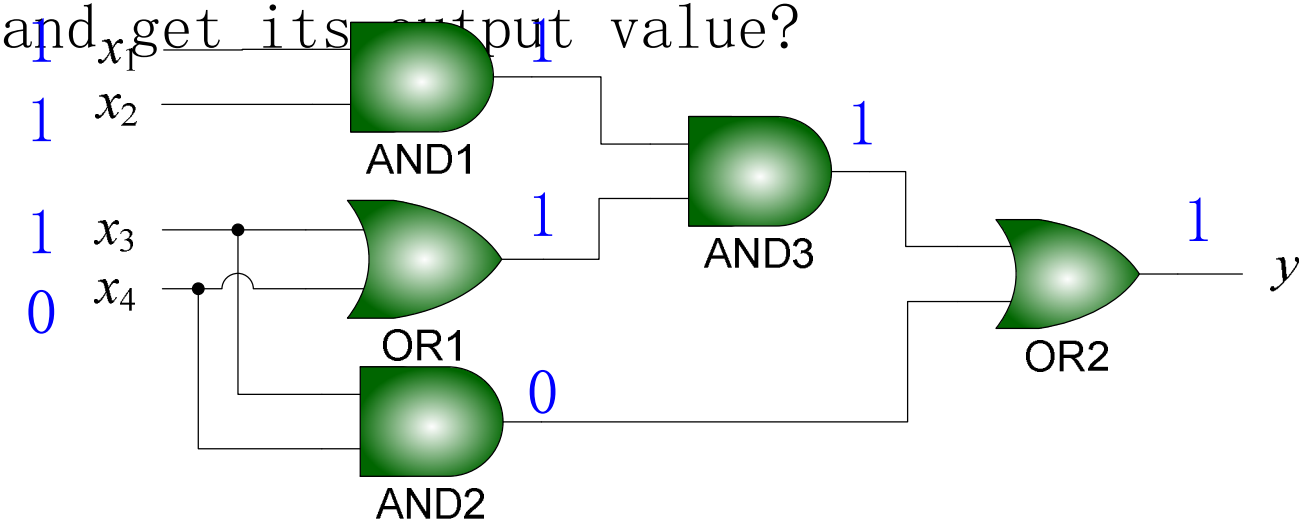
- Many problems in circuit design use computers to find solutions.
  - Example: how do you find the optimal multilevel circuit for implementing the following Boolean function?

$$a(b+c)+bce+\bar{b}d(\bar{a}+e)$$

cadence®

Data Structures and Algorithms and EDA

- Data structure and algorithm plays an important role in EDA
  - E.g., how do you represent a digital circuit in computer?
  - E.g., how to you traverse a digital circuit and get its output value?



Details of Campus Talk

- Time: 4 pm - 5:30 pm on Oct. 10th (Wednesday)
- Location: Dong Shang Yuan 401.

- I encourage you to attend the campus talk to learn about some basics of a software engineering company.