Comp151

C++:

Data Abstraction & Classes

A Brief History of C++

- Bjarne Stroustrup of AT&T Bell Labs extended C with Simula-like classes in the early 1980s; the new language was called “C with Classes”.

- C++, the successor of “C with Classes”, was designed by Stroustrup in 1986; version 2.0 was introduced in 1989.

- The design of C++ was guided by <u>three key principles</u>:
  1. The use of classes should not result in programs executing any more slowly than programs not using classes.
  2. C programs should run as a subset of C++ programs.
  3. No run-time inefficiency should be added to the language.

Topics in Today's Lecture

- Data Abstraction
- Classes: Name Equivalence vs. Structural Equivalence
- Classes: Restrictions on Data Members
- Classes: Location of Function declarations/definitions
- Classes: Member Access Control
- Implementation of Class Objects

Data Abstraction

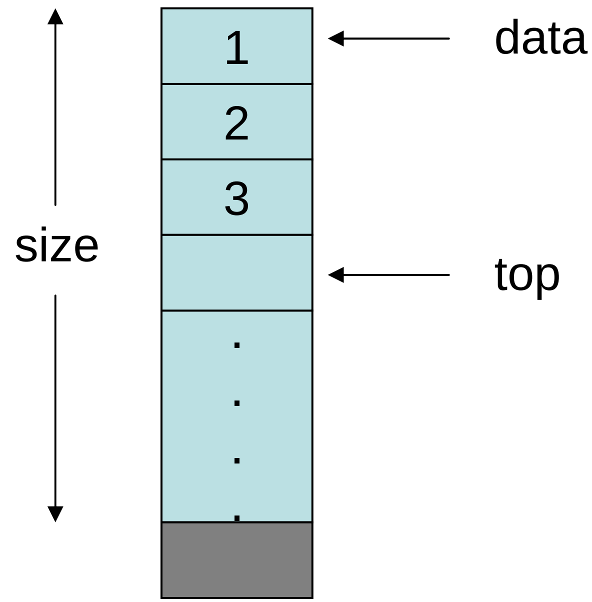
- Questions:
  - What is a ‘‘car''?
  - What is a ‘‘stack''?

- A **data abstraction** is a simplified view of an object that includes only features one is interested in while hiding away the unnecessary details.

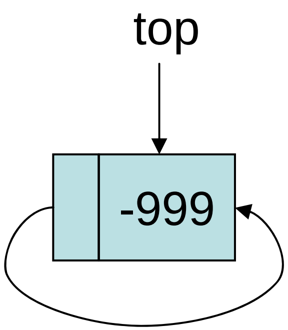
- In programming languages, a data abstraction becomes an **abstract data type** (**ADT**) or a **user-defined type**.

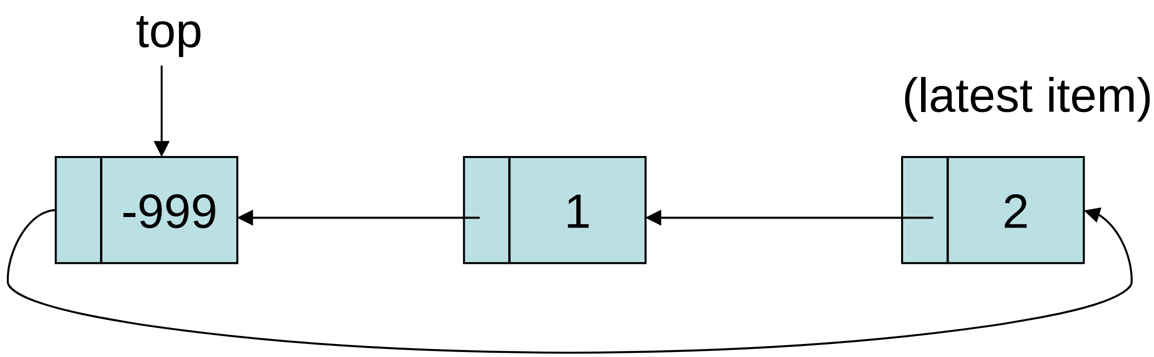
- In OOP, an ADT is implemented as a **class**.

Example: Implement a Stack with an Array



Example: Implement a Stack with a Linked List





Information Hiding

- An **abstract specification** tells us the behavior of an object independent of its implementation. i.e. It tells us <u>what</u> an object does independent of <u>how</u> it works.

- Information hiding is also known as **data encapsulation**, or **representation independence**.

- The **principle of information hiding**:

  Design a program so that the implementation of an object can be changed without affecting the rest of the program.

  – e.g., changing the implementation of a stack from an array to a linked list should have no effect on users' programs.

Example: stack_ar.h

data members

access control

```
class Stack
{
private:
    int size;                 // max size of storage
    int* top;                 // pointer to next available space
    int* data;                // data storage

public:
    Stack(int N);             // a constructor
    ~Stack();                 // destructor

    // basic operations;
    void push(int x);         // add another datum
    int pop();                // get the most recent datum

    // status operations;
    int num_elements() const;
    bool empty() const;
    bool full() const;
}
```

member functions
(public interface)

Example: stack_ll.h

data members

access control

```
class Stack
{
private:
    struct node
    {
    int data;
    node* next;
    }
    node* top;      // point to top of the stack
    int size;       // max size of storage

public:
    Stack(int N);   // a constructor
    ~Stack();       // destructor

    // basic operations;
    void push(int x);   // add another datum
    int pop();          // get the most recent datum

    //status operations;
    int num_elements() const;
    bool empty() const;
    bool full() const;
}
```

member functions
(public interface)

Class Name: Name Equivalence

- A class definition introduces a new abstract data type.
- C++ class definitions rely on name equivalence, NOT structure equivalence.
- E.g., the program below does not compile:

```
#include <iostream.h>
class X { public: int a; };
class Y { public: int a; };
void main()
{
  X x; Y y;
  x.a = 1;
  y.a = 2;
  cout << " x.a = " << x.a << ", y.a = " << y.a << endl;
  y = x;
  cout << " x.a = " << x.a << ", y.a = " << y.a << endl;
}
```

Class Name: Name Equivalence

- A class definition introduces a new abstract data type.
- C++ class definitions rely on name equivalence, NOT structure equivalence.
- On the other hand, this program compiles and works:

```
#include <iostream.h>
class X { public: int a; };
void main()
{
   X x1, x2;
   x1.a = 1;
   x2.a = 2;
   cout << " x1.a = " << x1.a << ", x2.a = " << x2.a << endl;
   x2 = x1;
   cout << " x1.a = " << x1.a << ", x2.a = " << x2.a << endl;
}
```

Data Members of a Class

- Data members can be any basic type, or any user-defined types that have already been "seen" (defined).

- A class name can be used (but only as a pointer) in its own definition:

  class Node { **public**: **int** data; Node* next; };

- It can also be used as a forward declaration for class pointers:

```
class Node;          // forward declaration
class Stack
{
   int size;
   Node* top;        // OK: points to an object with forward declaration
   Node x;           // ERROR: Node not defined!
};
```

Data Members of a Class

- But, data members can NOT be initialized inside the class definition. E.g., the program below will not compile.

```
class X {
public:
    int a = 1;          // ERROR: can't initialize member variables this way
};

void main()
{
    X x;
    cout << " x.a = " << x.a << endl;
}
```

- Instead, initialization should be done with appropriate constructors, or member functions.

Member Functions of a Class

- Class **member functions** are the functions declared inside the body of a class (they can be either public or private). They can be defined in two ways:

(1) <u>within</u> the class body, in which case, they are **inline functions**.

```
class Stack
{  …
   void push(int x) { *top = x; ++top; }
   int pop() { - - top; return (*top); }
};
```

Member Functions of a Class

(2) <u>outside</u> the class body

```
class Stack
{  …
   void push(int x);
   int pop();
};
void Stack::push(int x) { *top = x; ++top; }
int Stack::pop() { - - top; return (*top); }
```

- Question: Can we add data and function declarations to a class <u>after</u> the end of the class definition?

Member Access Control

- A member of a class can be:
  - **public** : accessible to anybody
  - **private** : accessible only to member functions and friends of the class → enforces information hiding
  - **protected** : accessible to member functions and friends of the class, as well as to member functions and friends of its <u>derived classes</u> (<u>subclasses</u>). We will discuss this in greater detail later in the course.

* We'll discuss "friends" later.

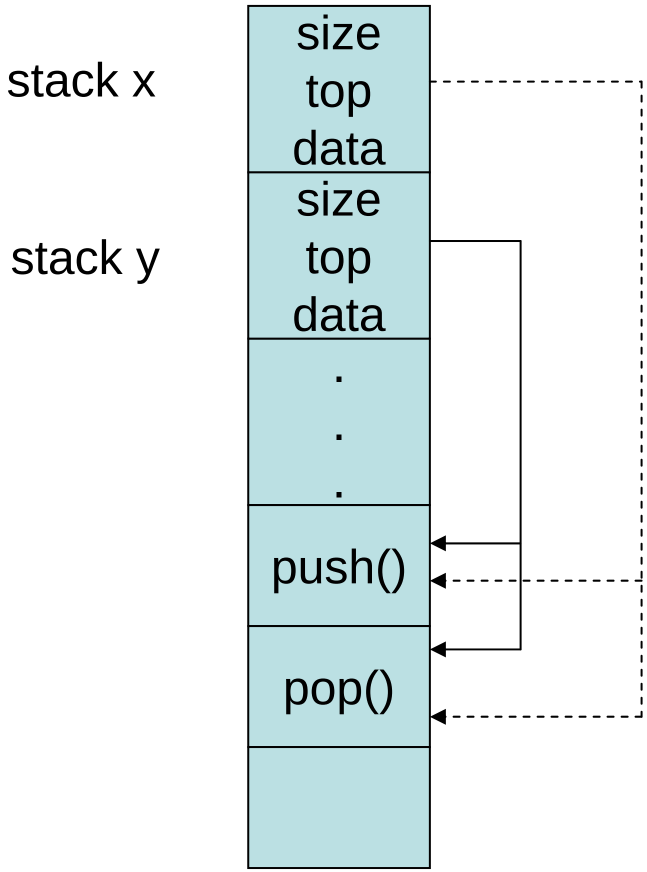
Example: Member Access Control

```
class Stack
{
private:
   Node* top;
public:
   int size;
…
};

int main()
{
   Stack x;
   cout << x.size;           // OK: size is public
   x.push(2);                // OK: push() is public
   cout << x.top->data;      // ERROR: cannot access top
}
```

How Are Objects Implemented?

- Each class object gets its own copy of the class data members.
- But all objects of the same class share one single copy of the member functions.




```
int main()
{
    Stack x, y;
    x.push(1);
    y.push(2);
    y.pop();
}
```