

Introduction to the C programming language

From C to C++: Stack and Queue

Giuseppe Lipari

<http://retis.sssup.it/~lipari>

Scuola Superiore Sant'Anna – Pisa

February 23, 2010

Outline

- 1 From struct to classes
- 2 First data structure: stack
- 3 Queue

Abstract data types

- An important concept in programming is the *Abstract Data Type* (ADT)
- An abstract data type is a user-defined type, that can be used similarly to built-in data types
- An ADT defines
 - What kind of values the data type can assume (*domain*)
 - What operations we can perform on the data type
- How the data and the operations are implemented is *hidden* to the user, and it is part of the implementation

ADT in C

- ADT are a general concept that can be supported in any language, including Assembler, Basic, C
- Example of ADT in C

```
struct point {  
    double x, y;  
    int z;  
};  
  
void point_read(istream &in, point *p);  
void point_save(ofstream &out, point *p);  
void point_print(point *p);
```

- The structure defines the *domain* (i.e. how the data is composed by three components)
The three functions define the *operations* we can do on the data
- It is not very nice to program ADT in C, because there is little support from the language. ADT are well supported in *Object Oriented* (OO) languages

Example in C++

- C++ is the OO version of C
- It maintains a similar syntax, adding new keywords and constructs
- How the previous class can be expressed in C++?

```
class Point {  
    double x, y;  
    int z;  
public:  
    Point(double x1, double y1);  
    Point();  
    void read(istream &in);  
    void save(ofstream &out);  
    void print();  
    double get_x();  
};
```

new keyword class instead of struct

this data is *private*, i.e. can only be used from the functions defined in the class

keyword public indicated that the following data and functions are public, i.e. that can be used by any other function

This is the constructor: it is used to initialize an *object* with proper data values

There can be more than one constructor (many different ways to construct the same object)

this function is part of the class, i.e. it can access all private data of the class

Usage

- This is an example of how the class Point can be used in a program.

```
int main()  
{  
    Point p(2,0);  
    Point q;  
  
    p.print();  
  
    p.x;  
}
```

Declares, defines and initialize a object of type Point. The constructor is invoked

The default constructor is invoked

Access a *public member* of class Point on the object p. In this specific case, invoked the function print() of class Point on object p.

This is a compilation error: x is a private member of Point and cannot be accessed from the other parts of the program.

Implementation

- Implementation usually goes into a separate class
point.cpp

c-cplusplus/point.cpp

```
#include <iostream>
#include "point.hpp"

Point::Point() : x(0), y(0), z(0)
{}

Point::Point(double x1, double y1) :
    x(x1), y(y1), z(0)
{}

void Point::print()
{
    cout << "(" << x << "," << y << ")";
}
```

c-cplusplus/point.cpp

```
void Point::read(istream &in)
{
    in >> x >> y >> z;
}

void Point::save(ostream &out)
{
    out << x << " " << y << " "
        << z << endl;
}

double Point::get_x()
{
    return x;
}
```

- Notice how we specify the functions, and how we access the member variables (i.e. variables defined inside the class).

Dynamic memory allocation

C language

```
int *p = (int *)malloc(sizeof(int));

int *a = (int *)malloc(10*sizeof(int));

...

free(p);
free(a);
```

C++ language

```
int *p = new int(0);

int *a = new [10] int;

...

delete p;
delete a;
```

- C++ uses the special keyword new, and a more automatic syntax (you can specify the type, and there is no need to specify the size)

Is that all?

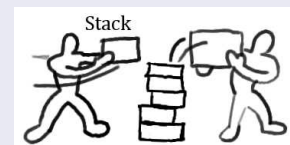
- C++ is a complex language, and we have just seen a few very basic concepts
- We have no time to present C++ in details. However, these very basic things should be necessary to start reasoning on data structures
- We will see more features as we go on.

Stack

- A stack is a very simple data structure.
- A stack can hold a set of uniform data, like an array (for example, integers)
- Data is ordered according to the LIFO (Last-In-First-Out) strategy

Two main operations are defined on the data structure:

- Push: a new data is inserted in the stack
- Pop: data is extracted from the stack



Usually, we can also read the element at the top of the stack with a Top operation

Stack interface

- Let's start by defining a stack of integers of fixed size
 - Initially, we will allow only a maximum number of elements in the stack

c-cplusplus/stack.hpp

```
#ifndef __STACK_HPP__
#define __STACK_HPP__

class Stack {
    int array[10];
    int top;
public:
    Stack();
    void push(int elem);
    int pop();
    int query();
    void print();
};

#endif
```

Stack implementation

- Here is the implementation

c-cplusplus/stack.cpp

```
#include <iostream>
#include "stack.hpp"

using namespace std;

Stack::Stack() : top(0)
{
}

void Stack::push(int elem)
{
    if (top < 10) array[top++] = elem;
    else cerr << "Stack is full, push operation failed" << endl;
}

int Stack::pop()
{
    if (top > 0) return array[--top];
    else cerr << "Stack::pop() : is empty" << endl;
}
```

Stack implementation - 2

c-cplusplus/stack.cpp

```
int Stack::query()
{
    if (top > 0) return array[top-1];
    else cerr << "Stack::query() : is empty" << endl;
}

void Stack::print()
{
    int i;
    cout << "[";
    for (i=0; i<top; i++) cout << array[i] << ",";
    cout << "]" << endl;
}
```

Usage

- This is only an example of how to use the Stack class.

c-cplusplus/stackmain.cpp

```
#include "stackdyn.hpp"
#include <iostream>

using namespace std;

int main()
{
    Stack s;
    int i;

    s.push(5);
    for (i=0; i<12; i++) s.push(2*i);

    s.print();

    for (i=0; i<5; i++) cout << s.pop() << endl;
}
```

Stack: unlimited size

- Let's remove the limitation on the size
 - We want a stack that enlarges itself dynamically

c-cplusplus/stackdyn.hpp

```
#ifndef __STACKDYN_HPP__
#define __STACKDYN_HPP__

class Stack {
    int *array;
    int cursize;
    int top;
public:
    Stack();
    ~Stack();
    void push(int elem);
    int pop();
    int query();
    void print();
};

#endif
```

Destructor

- The function `~Stack()` is called destructor
- It is automatically called when an object of type stack is destroyed
- As we will see in the implementation, in our case we need to deallocate the memory allocated by new with a corresponding delete.

Constructor and Destructor in stackdyn

c-cplusplus/stackdyn.cpp

```
1  #include <iostream>
2  #include "stackdyn.hpp"
3
4  using namespace std;
5
6  #define INC_SIZE 5
7
8  Stack::Stack() : top(0), cursize(INC_SIZE)
9  {
10     array = new int[INC_SIZE];
11 }
12
13 Stack::~Stack()
14 {
15     delete array;
16 }
```

Dynamic size stack implementation

c-cplusplus/stackdyn.cpp

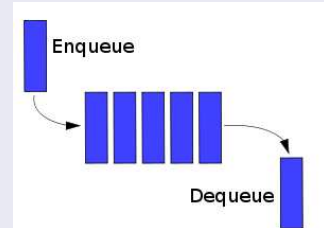
```
19 void Stack::push(int elem)
20 {
21     if (top >= cursize) {
22         int i;
23         int *temp = new int[cursize + INC_SIZE];
24         for (i=0; i<top; i++) temp[i] = array[i];
25         delete array;
26         array = temp;
27         cursize += INC_SIZE;
28     }
29     array[top++] = elem;
30 }
31
32 int Stack::pop()
33 {
34     if (top > 0) return array[--top];
35     else cerr << "Stack::pop() : is empty" << endl;
36 }
```

Queue

- Let us now implement a queue of integers
- The policy for inserting / extracting elements is FIFO (First-In-First-Out)

Two operations:

- enqueue inserts a new element in the queue
- dequeue extracts an element from the queue



Circular array

- Let's start again from a fixed size array

c-plusplus/queue.hpp

```
#ifndef __QUEUE_HPP__
#define __QUEUE_HPP__

class Queue {
    int array[10];
    int head;
    int tail;
    int num;
public:
    Queue();
    void enqueue(int elem);
    int dequeue();
    void print();
};

#endif
```

Queue implementation

c-cplusplus/queue.cpp

```
1  #include "queue.hpp"
2  #include <iostream>
3
4  using namespace std;
5
6  Queue::Queue() : head(0), tail(0), num(0)
7  {
8  }
9
10 void Queue::enqueue(int elem)
11 {
12     if (num < 10) {
13         array[head] = elem;
14         head = (head + 1) % 10;
15         num++;
16     }
```

Queue implementation - 2

c-cplusplus/queue.cpp

```
17 }
18
19 int Queue::dequeue()
20 {
21     int ret = 0;
22     if (num > 0) {
23         ret = array[tail];
24         tail = (tail + 1) % 10;
25         num--;
26     }
27     else cerr << "Queue::dequeue() : queue is empty" << endl;
```

Queue implementation - 3

cplusplus/queue.cpp

```
31 }  
  
33 void Queue::print()  
34 {  
35     int i;  
36     cout << "[";  
37     for (i=0; i<num; i++) cout << array[(tail + i)%10] << ",";  
38     cout << "]" << endl;  
39 }
```