Advanced Programming in C++

فرشاد حکیم پور

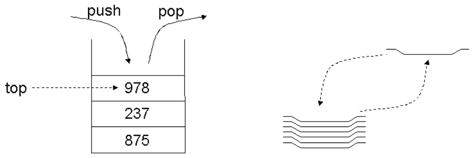
Array Structure

- An ordered (numbered) list of data elements, organized in consecutive locations in memory.
- Members of the list are accessed by an index number
- Elements of an array are [often] all of the same type

```
char str[30] = {‘a’, ‘b’, ‘c’};
char third = str[3];  //‘c’
```

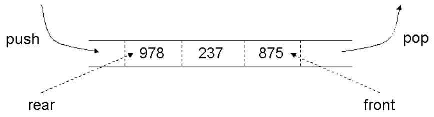
Stack Structure

- An ordered list
- Members of the list can be added or deleted only by one end of the list (*top*)
- Last In First Out (LIFO)



Queue Structure

- An ordered list
- Members of the list can be added at one end (*rear*) and deleted at the other end (*front*) of the list
- First In First Out (FIFO)



Stack Implementation

```
#include <iostream>
class Stack
{
private:
   static const int stackSize = 10;
   char stackArray[stackSize];
   int idx;     //position of the top
public:
   Stack(){ idx = 0; }  //object constructor
   bool push(char elem) {…}
   char pop() {…}
   int noOfElements() {…}
}
```

Stack Implementation

```
…
   bool push(char elem)
   {
      bool success = false;
      if (idx < stackSize)
      {
         stackArray[idx] = elem;
         idx++;
         success = true;
      }
      return success;
   }
…
}
```

Stack Implementation

```
…
   char pop()
   {
      char result = '?';
      if (idx > 0)
      {
         idx--;
         result = stackArray[idx];
      }
      return result;
   }
…
}
```

Stack Implementation

```
…
   int noOfElements()
   {
      return idx;
   }
}
```

Using the Stack Implementation

```
int main()
{
   Stack s = Stack();
   s.push('+');
   s.push('+');
   s.push('C');

   while (s.noOfElements() > 0)
      std::cout << s.pop();

   std::cout << std::endl;
   return 0;
}
```

Using the Stack Implementation

```
int main()
{
   Stack s = Stack();
   if (!s.push('+'))
      std::cout << "Stack Overflow" << std::endl;
   if (!s.push('+'))
      std::cout << "Stack Overflow" << std::endl;
   if (!s.push('C'))
      std::cout << "Stack Overflow" << std::endl;

   while (s.noOfElements() > 0)
      std::cout << s.pop();

   std::cout << std::endl;
   return 0;
}
```

How to use the Stack class

- Constructor:
  - `Stack()`
- Methods:
  - `bool push(char elem)`
    - Pushes the “elem” on top of the stack
    - Returns false for overflow and true for success
  - `char pop()`
    - Returns the value on top of the stack
    - Returns ‘?’ for underflow
  - `int noOfElements()`
    - Returns number of values in the stack

Definition vs. Declaration

- Declaration specifies that a variable or a method exists and how it looks. (The compiler allows the usage of that variable name in your code.)
- Definition says to the compiler to create (e.g. allocate memory) to the variable or function.
- Function *declaration* = Function *prototype*

Method Declaration

- Method name
- Input types
- Output type
- (Also called method signature)
- Declaration example:

  int modulo(int, int);

  int modulo(int dividend, int divisor);
- Usage example:

  reminder = modulo(v1, 3);

Method Definition

- Method name
- Input Types
- Input Names
- Output Type
- Body
- Definition example:

  int modulo(int dividend, int divisor) { … }

Functions Definition

```
#include <iostream>
#include <cmath>

double triangleArea(double a, double b, double c)
{
  double k = (a + b + c)/2;
  return sqrt(k * (k - a) * (k - b) * (k - c));
}

→ int main()
{
  std::cout << triangleArea(3, 4, 5) << std::endl;
  return 0;
}
```

Functions Definition

Error!
“triangleArea” undeclared

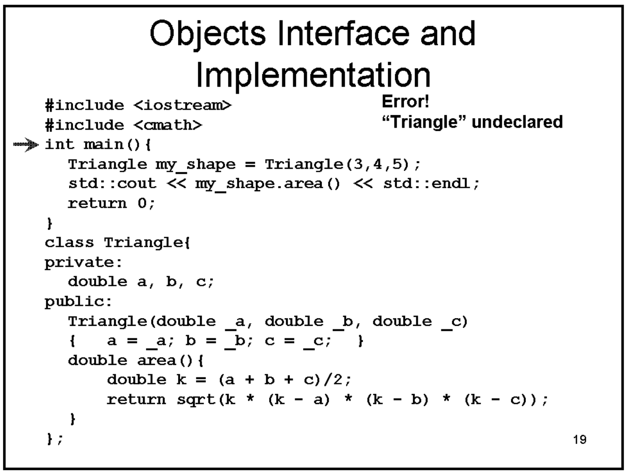
```
#include <iostream>
#include <cmath>

→ int main()
{
  std::cout << triangleArea(3, 4, 5) << std::endl;
  return 0;
}

double triangleArea(double a, double b, double c)
{
  double k = (a + b + c)/2;
  return sqrt(k * (k - a) * (k - b) * (k - c));
}
```

Functions Declaration and Definition

```
#include <iostream>
#include <cmath>

double triangleArea(double, double, double);
→ int main()
{
  std::cout << triangleArea(3, 4, 5) << std::endl;
  return 0;
}
double triangleArea(double a, double b, double c)
{
  double k = (a + b + c)/2;
  return sqrt(k * (k - a) * (k - b) * (k - c));
}
```

Objects Interface and Implementation

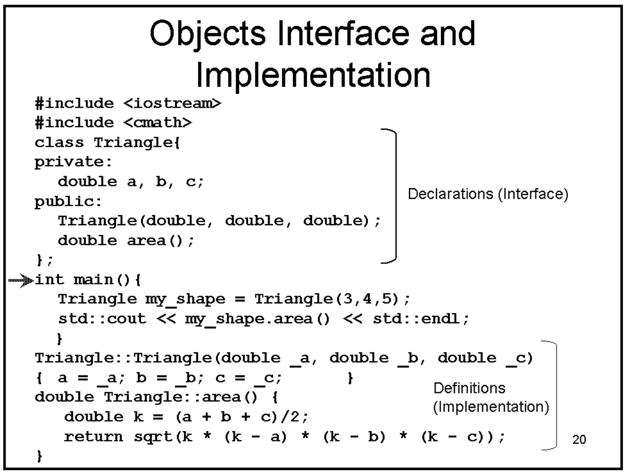
```
#include <iostream>
#include <cmath>

class Triangle{
private:
  double a, b, c;
public:
  Triangle(double _a, double _b, double _c)
  {   a = _a; b = _b; c = _c;   }
  double area(){
      double k = (a + b + c)/2;
      return sqrt(k * (k - a) * (k - b) * (k - c));
  }
};
→ int main(){
  Triangle my_shape = Triangle(3,4,5);
  std::cout << my_shape.area() << std::endl;
  return 0;
}
```



Objects Interface and Implementation

Error!
“Triangle” undeclared

```
#include <iostream>
#include <cmath>
int main(){
   Triangle my_shape = Triangle(3,4,5);
   std::cout << my_shape.area() << std::endl;
   return 0;
}
class Triangle{
private:
   double a, b, c;
public:
   Triangle(double _a, double _b, double _c)
   {   a = _a; b = _b; c = _c;   }
   double area(){
        double k = (a + b + c)/2;
        return sqrt(k * (k - a) * (k - b) * (k - c));
   }
};
```

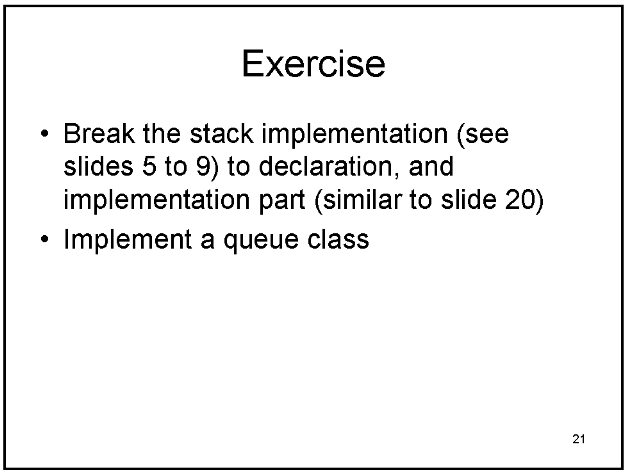


Objects Interface and Implementation

```
#include <iostream>
#include <cmath>
class Triangle{
private:
   double a, b, c;
public:
   Triangle(double, double, double);
   double area();
};
int main(){
   Triangle my_shape = Triangle(3,4,5);
   std::cout << my_shape.area() << std::endl;
   }
Triangle::Triangle(double _a, double _b, double _c)
{  a = _a; b = _b; c = _c;       }
double Triangle::area() {
    double k = (a + b + c)/2;
    return sqrt(k * (k - a) * (k - b) * (k - c));
}
```

Declarations (Interface)

Definitions (Implementation)



Exercise

- Break the stack implementation (see slides 5 to 9) to declaration, and implementation part (similar to slide 20)
- Implement a queue class