Templates

Chapter 11 (11.1 and 11.2 only)

Department of CSE, BUET

Objectives

- Generic functions
- Generic classes

Generic Functions

- A generic function defines a general set of operations that will be applied to various types of data
- Allows to create a function that that can automatically overload itself !!!
- Allows to make the data type, on which to work, a parameter to the function
- General form
  - template <class Ttype1, class Ttype2, …, class TtypeN>
  - ret-type func-name(param list)
  - {
  -    // body of function
  - }
  - Here,
    - template is a keyword
    - We can use keyword “typename” in place of keyword “class”
    - “TtypeN” is the placeholder for data types used by the function

GENERIC FUNCTIONS (EXAMPLE-1)

```
template <class X>
void swapargs(X &a, X &b) {
  X temp;
  temp = a;
  a = b;
  b = temp;
}
template <class X1, class X2>
void print(X1 x, X2 y) {
  cout << x << ", " << y << endl;
}
```

```
void main() {
  int i = 10, j = 20;
  double x = 11.11, y = 22.22;

  print(i, j); // 10, 20
  swapargs (i, j); // (int, int)
  print(i, j); // 20, 10

  print(x, y); // 11.11, 22.22
  swapargs (x, y); (double, double)
  print(x, y); // 22.22, 11.11

  print(i, y); // 20, 11.11
    // (int, double)
}
```

Generic Functions (contd.)

- The compiler generates as many different versions of a template function as required
- Generic functions are more restricted than overloaded functions
  - Overloaded functions can alter their processing logic
  - But, a generic function has only a single processing logic for all data types
- We can also write an explicit overload of a template function

Generic Functions (Example-2)

- template <class X>
- void swapargs(X &a, X &b) { cout << "template version\n"; }
- void swapargs(int &a, int &b) { cout << "int version\n"; }
- void main() {
-     int i = 10, j = 20;
-     double x = 11.11, y = 22.22;
-     swapargs(i, j); // "int version"
-     swapargs(x, y); // "template version"
- }

Generic Classes

- Makes a class data-type independent
- Useful when a class contains generalizable logic
  - A generic stack
  - A generic queue
  - A generic linked list etc. etc. etc.
- The actual data type is specified while declaring an object of the class
- General form
  - template <class Ttype1, class Ttype2, …, class TtypeN>
  - class class-name
  - {
  -   // body of class
  - };

Generic Classes (Example)

- template <class **X**>
- class stack {
-   **X** stck[10];
-   int tos;
- public:
-   void init( ) { tos = 0; }
-   void push(**X** item);
-   **X** pop( );
- };

- template <class **X**>
- void stack<**X**>::push(**X** item) { … }

- template <class **X**>
- **X** stack<**X**>::pop( ) { … }

GENERIC CLASSES (EXAMPLE) (CONTD.)

```
void main( ) {
    stack<char> s1, s2;
    s1.init( );
    s2.init( );
    s1.push(‘a’);
    s1.push(‘b’);
    s2.push(‘x’);
    s2.push(‘y’);
    cout << s1.pop( ); // b
    cout << s2.pop( ); // y
```

```
    stack<double> ds1, ds2;
    ds1.init( );
    ds2.init( );
    ds1.push(1.1);
    ds1.push(2.2);
    ds2.push(3.3);
    ds2.push(4.4);
    cout << ds1.pop( ); // 2.2
    cout << ds2.pop( ); // 4.4
}
```

LECTURE CONTENTS

- Teach Yourself C++
  - Chapter 11 (11.1 and 11.2)
  - Study the examples from the book carefully