Parameters that are Subprogram Names

- It is sometimes convenient to pass subprogram names as parameters
- Ada does not allow subprogram parameters;
- FORTRAN 95 and JavaScript allow.

Example of JavaScript:

```
Function sub1 { //JavaScript sample:
  var x;
    function sub2() {
      alert(x);//Create a dialog box with the value of x
    }

    function sub3(){
        var x;
        x = 3;
        sub4(sub2);}

    function sub4(subx){
        var x;
        x = 4;
        subx();    }
  x = 1;
  sub3();
};
```

Overloaded Subprograms

- An *overloaded subprogram* is one that has the same name as another subprogram

```
class Worker{
  public void doOverLoad(){
    int x = 3;
    double y = 4.2;
    System.out.println(square(x) + " "+ square(y));
  }

  public int square(int y){
    return y*y;
  }

  public double square(double y){
    return y*y;
  }
}

public class Ap079{
  public static void main(String args[]){
    new Worker().doOverLoad();
  }}
```

- C++, Java, C#, and Ada include predefined overloaded subprograms.
  Many predefined class in C++, Java or C# have overloaded constructors.

String() Initializes a newly created String object so that it represents an empty character sequence.
String(char[] value) Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
String(StringBuffer buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

- In C++, Java and C#, the return type is irrelevant to disambiguation of overloaded functions (or methods)

- In Ada, the return type of an overloaded function can be used to disambiguate calls (thus two overloaded functions can have the same parameters)

```
class Test{
  int x,y;
  int i,k;

  Test(int x, int y){
    this.x = x;
    this.y = y;
  }

  Test(int x, int y, int i, int k){
    this(x,y);// Must be in first line
    this.i = i;
    this.k = k; }
 }
```

Generic Subprograms

One way to increase the reusability of software is to lessen the need to create different subprograms that implement the same algorithm on different types of data.

```
int max ( int x, int y ){
    if ( x > y ){
        return x;
    } else {  return y;   }    }
```

If we wanted to have a similar function which works of double, we would have to define it as well

```
double max ( double x, double y ){
    if ( x > y ){
        return x;
    } else {  return y; }   }
```

This would work, but it would clearly be silly. This is where generic subprogram are useful.

- A *generic subprogram* takes parameters of different types on different activations

Generic functions in C++

Generic functions in C++ have the descriptive name of template functions.

A template function is useful when one want to be able to define a function which works in the same way for many different types.

```
template <class T>
T max ( T x, T y ){
        if ( x > y ){
            return x;
        } else { return y; }  }
```

This template definition is defining a parameter T which can represent any possible type.
Given this definition the compiler will automatically be able to build versions of the max which will find the maximum of two variables of type int, float, char *etc*.

```
cout << " max ( 1 , 2 ) = " << max(1, 2) << endl;
cout << " max ( 1.0 , 2.0 ) = " << max(1.0, 2.0) << endl;
cout << " max ( 'a' , 'z' ) = " << max('a', 'z') << endl;
```

Class Templates

Allow one to define data types which act as containers to store other types in a generic way.

```
template <class T>
class pair {
    T values [2];
  public:
    pair (T first, T second)
    {
      values[0]=first; values[1]=second;
    }
};
```

The class serves to store two elements of any valid type.

```
pair<int> myobject (115, 36);
```

Or

```
pair<float> myfloats (3.0, 2.18);
```

Generic Methods in Java 5.0

The most common generic types are container types, such as LinkedList and ArrayLisy.

The container types only store object of class objects, they cannot store primitive types.

Every time an object is removed from the structure, it must be cast to the approprite type.

```
ArrayList myArray = new ArrayList();
MyArray.add(0, new Integer(47));
Integer myInt = (Integer) myArray(0);
```

With Generics, an ArrayList can be restricted to store only Integer objects

```
ArrayList <Integer> myArray = new ArrayList <Integer>();
```

This usage eliminate the need for the cast when remove an object from the container.

```
public class Box <E> {
  E data;

  public Box(E data) {
    this.data = data;
  }

  public E getData() {
    return data;
  }
}
```

```
Box<Integer> intBox = newBox<Integer>(42);
Integer x = intBox.getData() // No cast needed
       //since the exact return type is known!

Box<String> strBox = new Box<String>("Howdy");
String s = strBox.getData()
```

Java's generic method differ from the generic subprogram of C++

(1) generic parameters must be classes. Not primitive types.

(2) restrictions can be specified on the range of classes that can be passed to the generic method as generic parameters.

```
public static <T extends Comparable> T getLast(T[] list)
{
    return list[list.length];
}
```

getLast() take an array of elements of a generic type and returns that last element of the array.

T must be a class that implements the Comparable interface.

Java 5.0 supports *wildcard types*. For example, Collection<?> is a wildcard type for collection classes (List, ArrayList, Vector, etc.).

```
void printCollection(Collection <?> c) {
   for (Object e: c) {
     System.out.println(e);
   }
}
```

This method prints the elements of any Collection class.