SubProgram pass by value: The value of the actual parameter is used to initialize the corresponding formal parameter. advantage: fast, disadvantage: additional storage **Example** Java;
pass by reference pass an access path. advantage no copying or duplicate storage. disadvantage: slower access, potential unwanted side effects. **Example** C++
pass by value-result: combination of pass by value and pass by result. disadvantage is copy storage, **Example** Fortrans
pass by name: by textual substitution. Formals are bound to an access method at the time of the call, but actual binding to a value or address takes place at the time of a reference or assignment. Allows flexibility in late binding. Implementation requires that the referencing environment of the caller is passed with the parameter, so the actual parameter address can be calculated. Disadvantages: Very tricky hard to read and understand. Essentially, the body of a function is interpreted at call time after textually substituting the actual parameters into the function body. In this sense the evaluation method is similar to that of C preprocessor macros. By substituting the actual parameters into the function body, the function body can both read and write the given parameters. In this sense the evaluation method is similar to pass-by-reference. The difference is that since with pass-by-name the parameter is evaluated inside the function, a parameter such as a[i] depends on the current value of i inside the function, rather than referring to the value at a[i] before the function was called.

Example: Algol 60

Example home work 4 and problem 9

SubProgram as parameter Why do we do that, what is the advantage and disadvantage:
advantage: very flexible, like the call back function that make the server works faster. disadvantage: hard to understand, bad readability.

```
function sub1() {
var x;
function sub2() {
alert(x); // Creates 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();
};


Consider the execution of sub2 when it is called in sub4. For shallow
binding, the referencing environment of that execution is that of sub4, so the
reference to x in sub2 is bound to the local x in sub4, and the output of the
program is 4. For deep binding, the referencing environment of sub2 s execution
is that of sub1, so the reference to x in sub2 is bound to the local x in
sub1, and the output is 1. For ad hoc binding, the binding is to the local x in
sub3, and the output is 3.
```

Overloaded subprogram: An overloaded subprogram is one that has the same name as another subprogram in the same referencing environment but with different or unique protocols. Ada, Java, C++, and C# allow users to write multiple versions of subprograms with the same name. **advantage**: very flexible, you can use a function with just the protocols you need, therefore the programmer don't have to memorize the header and signature of the method.**A polymorphic subprogram** takes parameters of different types on different activations.**Subtype polymorphism** means that a variable of type T can access any object of type T or any type derived from T.

Generic SubProgram: Parametric polymorphism is provided by a subprogram that takes generic parameters that are used in type expressions that describe the types of the parameters of the subprogram. Different instantiations of such subprograms can be given different generic parameters, producing subprograms that take different types of parameters. Parametric definitions of subprograms all behave the same. Parametrically polymorphic subprograms are often called generic subprograms.**Advantage:** very flexible. **Disadvantages** very hard to validate because different type of parameters can be passed, it is required a lot of testing in order to test the validity of these SubProgram.

Call by Name Examples

```
P(x) {x = x + x;}
What is: Y = 2;
P(Y);
write(Y)
F(m) {m = m + 1; return m;}
What is: int A[10];
m = 1;
P(A[F(m)])
becomes P(A[F(m)])    A[F(m)] = A[F(m)]+A[F(m)]
             A[m++] = A[m++]+A[m++]
              A[2] = A[3]+A[4]


Usual way - swap(x,y) { t=x; x=y; y=t; }
 Cannot do it with call by name!
Reason
Cannot handle both of following
swap(A[m], m)
swap(m, A[m])
One of these must fail
swap(A[m], m)     t = A[m] ; A[m] = m; m = t;
swap(m, A[m])     t = m ; m = A[m]; A[m] = t; // fails!
```

SubProgram as parameters advantage: Make it very flexible for the programmer.Javascript used the call back function in order to make the code non-blocking function mySandwich(param1, param2, callback) alert('Started eating my sandwich' it has 'param1' 'param2'); callback();

NEed to go back to learn about template

ADTAn abstract data type (ADT) is a user-defined data type that satisfies the following two conditions:-The representation of, and operations on, objects of the type are defined in a single syntactic unit-The representation of objects of the type is hidden from the program units that use these objects, so the only operations possible are those provided in the type's definition

Can you name a difference in Java ADT vs. C++ ADT (if any)? C++:classes and struct classes allocate them on heap or stack Java: heap only

C++:single and multiple inheritance Java: single inheritance, multiple inheritance through Interfaces which is equal to Java as the virtual methods. C++: template is the name for generic classes Java: generic classes.

To compare implementation of language Readability easily understood? Writability easy to write? Simplicity, orthogonality High expressive power, flexibility Reliability Safety Cost

```
Java Generic List
public class GenSet<E> {

    private Object[] a;

    public GenSet(int s) {
        a = new Object[s];
    }

    E get(int i) {
        @SuppressWarnings("unchecked")
        final E e = (E) a[i];
        return e;
    }
}
```

Inheritance: inheritance is a way to establish Is-a relationships between classes or objects.In classical inheritance where objects are defined by classes, classes can inherit attributes and behavior from pre-existing classes called base classes, superclasses, or parent classes

polymorphism: Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.So polymorphism is the ability (in programming) to present the same interface for differing underlying forms (data types).

Encapsulation: In programming languages, encapsulation is used to refer to one of two related but distinct notions, and sometimes to the combination thereof:

A language mechanism for restricting access to some of the object's components.[3][4] A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.[5][6]

Abstract class: They are the base classes. They are classes that cannot be instantiated, and are frequently either partially implemented, or not at all implemented. Classes extend from Abstract class might or might not implement all the abstract methods.

Dynamic binding:C++ does not allow value variables (as opposed to pointers or references)to be polymorphic. When a polymorphic variable is used to call a member function overridden in one of the derived classes, the call must be dynamically bound to the correct member function definition. Member functions that must be dynamically bound must be declared to be virtual functions by preceding their headers with the reserved word virtual, which can appear only in a class body. The objects of C++ can be static, stack dynamic, or heap dynamic. Explicit deallocation using the delete operator is required for heap-dynamic objects, because C++ does not include implicit storage reclamation.

In C++, a method must be defined as virtual to allow dynamic binding. **In Java,** all method calls are dynamically bound unless the called method has been defined as final, in which case it cannot be overridden and all bindings are static. Static binding is also used if the method is static or private, both of which disallow overriding. **(3) How does Java support cooperation synchronization? How does Java support competition synchronization?**

Cooperation synchronization in Javais achieved via wait, notify, and notifyAll methods All methods are defined in Object, which is the root class in Java, so all objects inherit them The wait method must be called in a loop The notify method is called to tell one waiting thread that the event it was waiting has happened The notifyAll method awakens all of the threads on the objects wait list

A method that includes the synchronized modifier disallows any other method from running on the object while it is in execution public synchronized void deposit(int i) public synchronized int fetch() The above two methods are synchronized which prevents them from interfering with each other If only a part of a method must be run without interference, it can be synchronized thru synchronized statement synchronized (expression) statement

Threads, synchronization **Cooperation:** Task A must wait for task B to complete some specific activity before task A can continue its execution, e.g., the producer-consumer problem **Competition:** Two or more tasks must use some resource that cannot be simultaneously used, e.g., a shared counter Competition is usually provided by mutually exclusive access (approaches are discussed later)

Java Threads: A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. **Java Exception**

Binding an exception to a handler is simpler in Java than it is in C++ An exception is bound to the first handler with a parameter is the same class as the thrown object or an ancestor of it An exception can be handled and rethrown by including a throw in the handler (a handler could also throw a different exception)

If no handler is found in the try construct, the search is continued in the nearest enclosing try construct, etc. If no handler is found in the method, the exception is propagated to the methods caller If no handler is found (all the way to main), the program is terminated To insure that all exceptions are caught, a handler can be included in any try construct that catches all exceptions Simply use an Exception class parameter Of course, it must be the last in the try construct

finally Clause Purpose: To specify code that is to be executed, regardless of what happens in the try construct

Exceptions of class Error and RunTimeException and all of their descendants are called unchecked exceptions; all other exceptions are called checked exceptions Checked exceptions that may be thrown by a method must be either: Listed in the throws clause, or Handled in the method

Java hierarchy Throwable
Error
Not to be used by the programmers
Exception
IOException
EOFException, FileNotFoundException, MalformedURL Exception, UnknownHostException
RuntimeException
ArithmeticException, ClassCastException, IllegalArgumentException (-> NumberFormatException), IllegalStateException, IndexOutofBoundsException (-> ArrayIndexOutOfBoundsException),
NoSuchElementException (->InputMismatchException),
NullPointerException
ClassNotFoundException
CloneNotSupported Exception

```
 function template
#include <iostream>
using namespace std;

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
```