Question # 1 [**10 marks**] Recent implementations of C++ compilers refuse to compile the following code :

```
class Foo {
  .....
  Foo(Foo f) {
    ....
  }
}
```

Why? *Older versions actually did compile this with disastrous results.* **Hint:** *Feel free to answer in one sentence.*

ANSWER:

The constructor in question is a copy constructor (*it creates an instance of* `Foo` *from another instance of* `Foo`). The copy constructor is implicitly called whenever an instance of `Foo` is returned or passed *by value.* This constructor takes an instance by value, hence it causes an *infinite recursion* whenever an instance of `Foo` is passed or returned by value.

Full marks for any of:

- infinite recursion
- "should be passed as `const Foo&`" ...
- wrong argument to copy constructor ...

Question # 2 [**10 marks**] All variables and methods of a class inherit to its derived classes, including those that are private (*even if they are inaccessible directly*).

Write a very short (*5-10 line*) code segment with two classes – `Base and Derived` – which demonstrates that private instance variables of `Base` are present in `Derived`.

ANSWER:

```
class Base {
  private:
    int a;
  public:
    Base():a(5) {}
    int get_a();
}

class Derived: public Base {
}

....

Derived x;

cout << x.get_a() << ...  // this better be "5", which proves that a is
                          // present in Derived
```

Full marks for anything along this line ...

Question # 3 [**10 marks**] In a few words (*if you wish in point form*) describe what polymorphism is and describe a small application (*maybe with a tiny object model*) where its use benefits the programmer.

ANSWER:

Polymorphism is a mechanism provided by Object Oriented languages, which allows operations to have multiple implementations with the same name. The appropriate implementation of the operation is resolved by the compiler generated code or interpreter even if the decision has to be made run-time.

Using this mechanism together with inheritance, simplifies program logic by eliminating the need for run-time type checking and new subclasses with a new implementation of the operation can be readily added without the need of touching existing code.

```
     +---------+             +-------+
     | Animal |o-------------| Barn |
     +---------+             +-------+
     | speak  |
     +---------+
      |      |
+---------+  +---------+
|  Cow   |  |   Cat  |
+---------+  +---------+

Barn B; Cat fergi, ronald; Cow abel, sarah;
B.add(&fergi); B.add(&abel); B.add(&ronald); B.add(&sarah);

for(int i=0; i<B.size(); i++) {
  B.animals[i] -> speak();
}

Output: mew moo mew moo
```

Question # 4 [**10 marks**] Parametric classes. (Container & Iterator)

4.1. [**3 marks**] Why do container classes usually contain pointers to instances rather than the instances themselves?

ANSWER: 1. Run-time polymorphism only works with pointers and references.
2. Data sharing (*aliasing*) **Any earns you full marks**

4.2. [**3 marks**] What is the main reason (*in your opinion*) to implement container classes using templates?

ANSWER: Using templates, the class or type of data held in the container is a parameter. No need for separate implementations of integer, string, ... containers.

4.3. [**4 marks**] What use – other than implementing container classes and iterators – do you think are there for templates ? (*Be very brief!*)

ANSWER: Anything that makes sense to be a parameter or it makes sense to be parameterized at compile time, which includes *associations*, *default values (buffer size,...)*, or anything you can convince me of ..

Question # 5 [**10 marks**] Pointers & Arrays

5.1. [**4 marks**] Allocate a 2x3 matrix dynamically and save its address in a variable:
`int **a;`

5.2. [**3 marks**] Draw how it would look in physical memory (*that is, graphically indicate ONE possible arrangement of the rows and columns, observe adjacencies and do separate those blocks of memory that may not be contiguous.*)

5.3. [**3 marks**] On your diagram, indicate what

- `a`
- `*a[1]`
- `&a[0][2]`
- `a[0]`

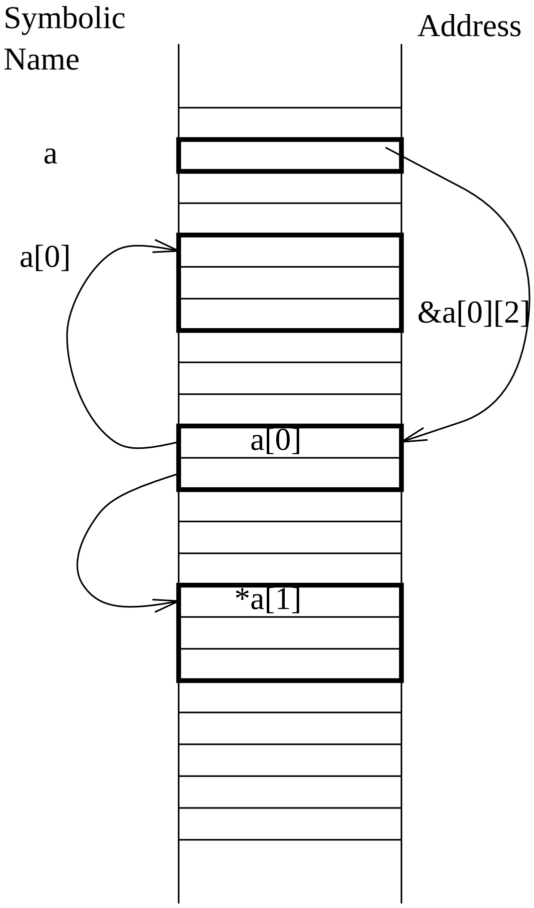
are. *Feel free to use the next page as well to answer this question.*

Continued for question 5

ANSWER:

```
int **a,row=2,col=3;

a = new int*[row];
for(int i=0;i<row;i++) {
  a[i] = new int[col];
}
```



Question # 6 [10 marks] Question about the "project".

Suppose you are asked to describe your involvement/contribution of the C++ project you had in 1998. (*The scripted drawing package.*)

6.1. [**4 marks**] In a very few words – if you wish in point form – summarize your work! *Be very brief and spare any technical jargon! You are limited to space between this and the next question!*

ANSWER:

If you did your part, you should get perfect on this ...

6.2. [**6 marks**] In a very few words, discuss how your design of the project can be extended

- to support more shapes (*pentagon, ...*)
- to support more options (*line width, ...*)

also comment on how much work you predict these changes would take. *Be very brief and down to the point. Absolutely* **no** *code required!*

ANSWER:

> **Adding a new shape:** Add a new subclass of `Shape` and implement its `draw` method. Also add code that reads the new shape. Nothing else.
>
> **Adding a new option:** Declare a new instance variable in the appropriate abstract class (`shape` *or* `drawable`) which represents the option. Add code that reads this option. Modify or implement a method which handles this option. (*if it were line drawing, only the* `line::draw` *method needs to be modified*)

Question # 7 [**10 marks**] List all situations (*you know*) where

1. The default constructor is called
2. A particular constructor is called implicitly
3. The copy constructor is called
4. The destructor is called

No code examples are needed!

ANSWER:

Default constructor: 1. Declaration of a static instance. 2. By the constructors of a derived class (unless another one specifically specified). 3. By the `new` operator.

Particular constructor implicitly: If a constructor exists which can create an instance of the class from a particular type, whenever that type is used where an instance of the class is expected, the constructor is implicitly called to create a temporary instance.

Copy constructor: Passing and returning an instance by value.

Destructor: Instance leaves its scope and by the `delete` operator.

Question # 8 [10 marks] Streams.

8.1. [5 marks] What is serialization?

ANSWER:

> Serialization is the process of writing an instance to an output stream in a form which can be used to recreate the very same logical instance from an input stream.

8.2. [5 marks] A programmer is faced with the problem to read `foo` objects from a modem. She has already obtained an implementation of `istream& operator>>(istream&,foo&)`. What does she have to do? **Hint:** *there is only one meaningful answer!*

1. modify `istream& operator>>(istream&,foo&)`
2. implement `ostream& operator<<(ostream&,const foo&)`
3. implement the subclass `imodemstream: public istream` **BINGO!**
4. implement the subclass `omodemstream: public ostream`
5. panic

Question # 9 [10 marks] True or false? If the answer is "True" just write so, if the answer is "False" give a word or so explanation why it is false.

9.1. [**2 marks**] Constructors inherit.

NO! If the mechanism were inheritance either the superclass' constructor inherits which would not initialize new instance variables of the derived class or the derived class' constructor overwrites the superclass' version, which does not initialize instance variables of the superclass.

9.2. [**2 marks**] The destructor inherits.

NO! Reasoning is analogous to the answer above, except for instance deallocation.

9.3. [**2 marks**] The assignment operator inherits.

TRUE!

9.4. [**2 marks**] A static variable of a class has the same physical address for all instances of the class.

TRUE!

9.5. [**2 marks**] `C`'s `malloc` and `free` can be used instead of `C++`'s `new` and `delete` operators with **no** side effects.

NO! `malloc` does not call a constructor and `free` does not call the destructor.

Question # 10 [10 marks] Very briefly describe an application which demonstrates how inheritance facilitates code reuse. (*A small object model may accompany your description. Absolutely no code required!*)

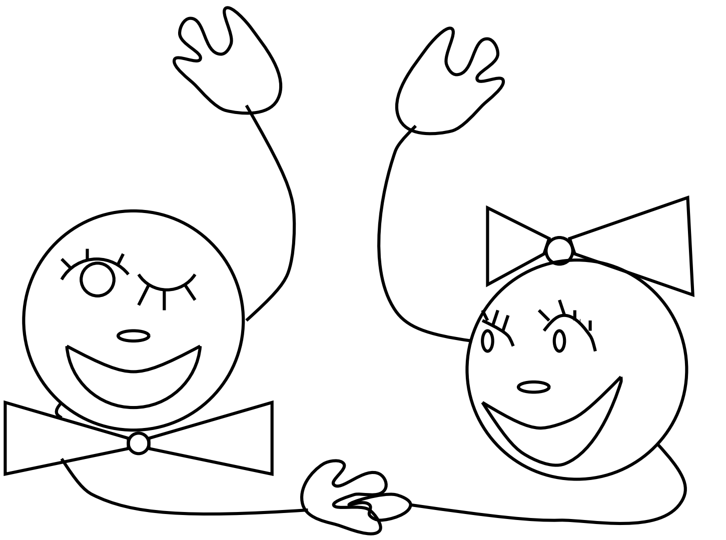
ANSWER:

A brief description of your project's `drawable hierarchy` would certainly be a perfect example.

Inheritance facilitates code reuse in many ways. First, method implementations of an operation inherit to the subclasses. This alone saves a lot of `cut` and `paste`. On top of this, inheritance together with polymorphism makes code that implements *logic* reusable. We have an example in the notes where we implement all comparisons using $<$ (*less*) in an abstract numeric class. Every subclass which implements this operator gets the rest of the comparisons *free.*

If you came up with your own example, all the better ...

Good Luck !



CSI 2172A Final Examination

April 17, 1998

Name: ________________ **Albert Einstein** ________________

Student Number: ________________ $\sqrt[2\alpha]{\beta+\gamma^2}$ ________________

- This is a 3 hours long examination.
- This is a *closed book* examination.
- Answer all questions in the space provided and use the back of the pages if it is necessary.
- This examination booklet contains **14** numbered pages including this cover page.

Question	Page	Maximum	Your Mark
1.	2	10	**10**
2.	3	10	**10**
3.	4	10	**10**
4.	5	10	**10**
5.	6	10	**10**
6.	8	10	**10**
7.	10	10	**10**
8.	11	10	**10**
9.	12	10	**10**
10.	13	10	**10**
Total:			**100**