American University of Beirut
CMPS 258
Programming Languages
Fall 2005-06

Final Exam

Date: Jan. 28th 3:00 – 5:00pm. **Name: ----------**
Instructor: Chiraz Ben Abdelkader

- This exam is closed-book and closed notes. You are allowed to bring **2 A4 papers of notes**, front and back.
- There are **9 pages** and **4 sets** of questions, for a total of **100 points**.
- The questions are broken down into four main parts as shown below; plan your time wisely based on the distribution of points and difficulty of the questions.
- Write as legibly as you can, otherwise I might NOT mark your answer!

Subprograms …… / 20 points
Functional Programming ……/ 25 points
Object-oriented programming ……/ 30 points
Miscellaneous ……/ 25 points

Total……/100 points

Part I: Subprograms *(20 points)*

1. Given an example of when pass-by-reference might lead to unpredictable program behavior. Explain your answer.

2. Consider the following program written in C-like syntax. Assume *static scoping*.

```
float C[3] = {0.1,0.2,0.3};
int i;
void foo(float x) {
   i++;
   x = 0.7;
   C[1] = 0.5;
   i--;
}

void main() {
   for (i=0; i<3; i++)
      foo(C[i]);
   printf("%d, %d, %d, %d \n", C[0], C[2], C[3]);
}
```

   (a) What is the output if function parameters are passed by value?

   (b) What is the output if function parameters are passed by reference?

   (c) What is the output if function parameters are passed by value-result?

3. Consider the following three, slightly different programs in C:

Program 1

```
#define N 100000   /* this is how constants are defined in C */
int A[N];

void Initialize() {
   int i;
   for (i=0;  i<N; i++)
        A[i] = 1;
}

int main() {
   float *x;
   Init();
}
```

Program 2

```
void Initialize(int *p, int n) {

   int i;
   for (i=0;  i<n; i++)
        *p++ = 1;
}
int main() {

  float *x;
  int A[100000];
  Initialize(A, 100000);
}
```

Program 3

```
void Initialize(int *p, int n) {
   int i;
   for (i=0;  i<n; i++)
        *p++ = 1;
}
int main() {
  float *x;
  int *A = malloc(100000*sizeof(int));
  Initialize(A, 100000);
  free(A);
}
```

a) For each program, show the contents of the *runtime stack* while function **`Initialize`** is active, i.e. is being executed. You may assume that *global* variables are stored at the bottom of the stack.

b) For each program, give all variables that are *bound to storage* at the **four** execution points given in table below; for each variable, specify <u>amount of storage</u> in bytes.

	Start of main	Start of Initialize	End of Initialize	End of main
Program#1				
Program#2				
Program#3				

Part II: Functional Programming Paradigm *(25 points)*

1. Explain the relationship between the concept of *referential transparency*, and the fact that in many Scheme functions, the order of evaluating function arguments in a function evaluation expression does not matter.

2. What will the following Scheme s-expressions evaluate to?

```
'(QUOTE QUOTE)                          =>

'QUOTE                                  =>

((if 'A CAR CDR) (list A B))            =>

((CAR '(+ -)) 5 2)                      =>

((CAR (list / *)) 5 2)                  =>

(define foo(x,y)

    ( (lambda(a b) (if (< a b) a b)) (- (* x x 7) 10) (* y y y) )  )

(foo -1 5)                              =>

(map list? '( (A B C) D 3 (()) ) )  =>

(map (lambda (z) (floor (/ z 5))) (list (+ 26 3) -1) )  =>
```

3. Consider the following Scheme function:

```
(define foo(i)

   (cond ( (not (integer? i) '() )

         ( (< i 0) '() )

         ( else (cons i (foo (- i 1))) )  ))
```

   (a) Describe in one sentence what this function computes in general.

   (b) What is the value of this s-expression: `(foo 7)` =>

4. Write a Scheme function named **`listmax`** that takes a list, **`L`**, as a parameter and returns the largest number that appears as an element of **`L`**. If **`L`** contains any non-numeric elements, then the function should return `()`. For example, **`(listmax '(3 9 1 9 7))`** => `9` Obviously, you may not use the Scheme built-in **`max`** function.

Note: You can check whether an element is a number using the function **`number?`**

YOU ONLY NEED TO DO ONE QUESTION OF THE FOLLOWING TWO (#5 & #6)

5. A *palindrome* is a sentence that reads the same from left to right and from right to left. For example "the dog is dog the" is a palindrome. Write a Scheme function **`palindrome?`** that takes a list of words **`L`** as an argument, and returns **`#t`** if it is a palindrome, and **`()`** otherwise.

Hint: you may assume you have a function last that returns the last element of a list, and a function reverse that returns the elements of a list in reverse order.

6. Write a Scheme function **`remove`** that takes two lists as arguments, **`L1`** and **`L2`**, both of which are assumed to be lists of words, and returns **`L1`** but with words that also appear in **`L2`** removed. For example:

```
(remove '(the wings of the dove) '(the of)) => (wings dove)
```

Hint: you might want to start by writing a helper function that is called in **`remove`**.

Part III: Object-Oriented Programming Paradigm *(30 points)*

1. Give **two** distinct OOP features/constructs in the C++ language that violate data abstraction principles, but which are good from the viewpoint of either software reuse or programming flexibility.

2. Consider the following two design problems:

   i. You need to implement multiple variants of the Queue data structure as part of some software application. These Queue variants support the same three operations (Enqueue, Dequeue, IsEmpty), but only differ in the type of data objects.

   ii. You want to design a class for each breed of cats, as part of a database application for some pet store. The only thing cats do is eat and sleep. But each breed of cat has different eating and sleeping habits.

Assuming you want to code in C++, explain for each one of the above problems whether it is best to implement it using templated classes or inheritance.

3. What is the output of the following code?

```
#include <iostream>

class Pet {
   public:
      Pet(int id, string b) { ID = id;  breed = b;  }
      virtual void printinfo() = 0;
      void Add()  { "Adding a Pet" << endl; }
   protected:
      int ID;
      string breed;
};
class Cat: public Pet {
   public:
      void Add()  { "Adding a Cat" << endl; }
      void printinfo() {
         cout << ID << ": a cat of type " << breed << endl;  }
};
```

```
class Dog: public Pet {
   public:
      void Add()  { "Adding a Dog" << endl; }
      void printinfo() {
         cout << ID << ": a dog of type " << breed << endl;  }
};

void main() {
   Pet *c, *d;
   c = new Cat(98182,"Siamese");
   d = new Dog(10293,"Labrador");
   c->Add();
   d->Add();
   c->printinfo();
   d->printinfo();
}
```

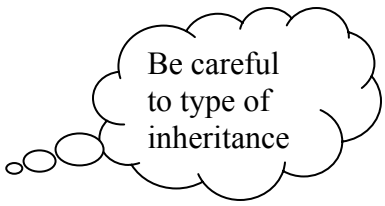
4. Consider the following C++ code:

```
class A {
public:
     void f1();
     virtual void f2();
     void f3();
};

class B : public A {
public:
      void f2();
      void f3();
protected:
      void f4();
};
```

```
class C: public B {
public:
      void f1();
      void f2();
};
class D: private B {
public:
      void f1();
      void f2(int, float);
friend class C;
};

void main() {
   A a; B b; C c; D d; A *pA;
   ...
}
```



Indicate in the table below which methods are visible to objects **a, b, c**, and **d** in **main** when: **(a)** they are accessed directly, and **(b)** they are accessed through the pointer **pA**. Use the scope operator to indicate the class of each method; for example **A::f1().**

	Object a	Object b	Object c	Object d
(a)				
(b)				

Part IV: Misc. Short-Answer Questions *(25 points)*

1. In recursive descent parsing, why is that a *left recursive* rule leads to an infinite loop, while a *right recursive* rule does not? Illustrate your answer with an example.

2. Classify each of the following errors as either a **lexical**, **syntax**, or **semantic error**. For partial credit, briefly justify your answer in one sentence. You may assume the language has same syntax as C, C++, or Java.

   (i) Programmer mistyped a keyword, for e.g. **`esle`** instead of **`else`**.

   (ii) Programmer mistyped **`if(x=y)`** instead of **`if(x==y)`**

   (iii) Programmer mistyped **`if(x <> y)`** instead of **`if(x < y)`**

   (iv) Programmer forgot to initialize a variable to zero before using it.

   (v) Program contains an infinite loop.

   (vi) Programmer forgot the right quotation mark of a string, such as:

   ```
   char s[] = "Hello;
   ```

3. Contrast static type binding and dynamic type binding: what are the advantages and drawbacks of each?

4. Can a variable be bound to storage at compile time? Why or why not?

5. Give two reasons why the compiler needs to maintain/store descriptors for all variables in a program (during compilation)?

6. Why is array access via subscripting potentially slower than array access via pointers?

7. List all the pointer problems in this piece of code.

```
struct {
  int *arr;
  int x;
} MyStr;

void foo (MyStr *s) {
  int A[100];
  int B = new int[100];
  s->arr = A;
  s->x = B[0];
}

void main() {
  MyStr t;
  foo(&t);
  free(t.arr);
}
```