COMS W4115 Programming Languages and Translators
Homework Assignment 3

Submit this assignment online via Courseworks as a PDF file. Fill in or annotate this PDF or print it out, write on it, and scan it. Please keep your answers in the boxes.

Do this assignment alone. You may consult the instructor and the TAs, but not other students.

Name: Uni:

1. (20 pts.) For the following C array on a processor with the usual alignment rules,

```
int a[2][3];
```

(a) Show the order in which its elements are arranged in memory.

(b) Write an expression for the byte address of a[i][j] in terms of a (the address of the start of the array), i, and j.

(c) Verify parts a) and b) by writing a small C program that **tests your hypothesis**. Examine the assembly language output with the C compiler's -S flag (e.g., `gcc -O -S array.c`). Such a program should be simple and contain and access such an array, but not be so simple that the compiler optimizes most of it away. On the next page, **include in an annotated assembly listing** that explains how it verifies your hypothesis. Make sure the assembly listing is no more than about 40 lines, either by simplifying your program or trimming the output.

C program:

Assembly listing:

2. (20 pts.) For a 32-bit little-endian processor with the usual alignment rules, show the **memory layout** and **size in bytes** of the following three C variables.

```
union {
  struct {
    char  a;   /* 8-bit */
    int   b;   /* 32-bit */
    short c;    /* 16-bit */
  } s;
  struct {
    int d;    /* 32-bit */
    short e; /* 16-bit */
  } t;
} u1;
```

Layout:

Size in bytes:

```
struct {
  char a;
  int b;
  short c;
  short d;
} s1;
```

Layout:

Size in bytes:

```
struct {
  short a;
  char b;
  short c;
  char d;
  short e;
} s2;
```

Layout:

Size in bytes:

3. (20 pts.) Draw the layout of the stack just before *bar* is called in *foo*. Indicate storage for function arguments, local variables, return addresses, and stored frame pointers. Indicate where the stack and frame pointers point.

```
void bar(int x, int y, int z);

void foo(int a, int b) {
  int d, e;
  bar(2, 5, 7);
}
```

4. (20 pts.) **Draw the layouts** of s1 and s2 and the virtual tables for the Ellipse and Square classes.

```
public class Shape {
   double x, y;
   public double area() { ... }
}

class Ellipse extends Shape {
   private double height, width;
   public double area() { ... }
}

class Square extends Shape {
   private double width;
   public double area() { ... }
}

public class Main {
  public static void main() {
    Shape s1 = new Square(10, 3, 14);
    Shape s2 = new Ellipse(3, 8, 2, 6
    System.out.println( s1.area() );
  }
}
```

Square Virtual Table:

Ellipse Virtual Table:

s1 object:

s2 object:

5. (20 pts.) For the program below written in a C-like language with nested function definitions,

```
void main() {
  int x = 5;

  void bar() {
    x = x + 2;
  }

  void foo() {
    int x = 8;
    bar();
    printf("%d\n", x);
  }

  foo(); /* Body of main() */
}
```

What would it print if the language used **static scoping**?

What would it print if the language used **dynamic scoping**?