**Part I**

Implement the following program fragment to call the correct function based on the input. This is very close to how polymorphism is implemented in C++ with vtables and pointers.

# Call a function based on an input number from 1...5

# If input == 1, call f1, if input == 2, call f2, etc.

.text

.globl main

main:

#prompt for input

li $v0, 4

la $a0, prompt

syscall

li $v0, 5

syscall

move $s0, $v0

subi $s0, $s0, 1 # array offset is in $s0 (not zero based)

#Check input

slt $t0, $s0, $0

sgt $t1, $s0, 4

or $t0, $t0, $t1

beqz $t0, validInput

li $v0, 4

la $a0, error

syscall

b callExit

validInput:

# Call the function at the correct array position by

# calculating the array address, loading the value at

# that address (which is the function pointer or address),

# and doing a jalr to run the function and print the

# correct output.

**la $t1, funcArr # Load address of array**

**mul $t2, $s0, 4 # Calculate offset**

**add $t1, $t1, $t2 # Calculate the address**

**lw $t4, 0($t1) # Load value at that address**

**jalr $t4 # Jump to function**

callExit:

li $v0, 10

syscall

.data

prompt: .asciiz "Enter a number from 1..5: "

error: .asciiz "Number must be from 1...5"

funcArr:

.word f1

.word f2

.word f3

.word f4

.word f5

.text

.globl f1

f1:

li $v0, 4

la $a0, f1Output

syscall

jr $ra

.data

f1Output: .asciiz "In f1"

.text

.globl f2

f2:

li $v0, 4

la $a0, f2Output

syscall

jr $ra

.data

f2Output: .asciiz "In f2"

.text

.globl f3

f3:

li $v0, 4

la $a0, f3Output

syscall

jr $ra

.data

f3Output: .asciiz "In f3"

.text

.globl f4

f4:

li $v0, 4

la $a0, f4Output

syscall

jr $ra

.data

f4Output: .asciiz "In f4"

.text

.globl f5

f5:

li $v0, 4

la $a0, f5Output

syscall

jr $ra

.data

f5Output: .asciiz "In f5"

**Part II** - The following program shows how to set an interrupt handler (function pointer) in C for a SIGINT.

/\*\*

\* Name: Sigint.c

\* Author: Charles Kann

\* Date: August 22, 2002

\*

\* Purpose: This program shows how to override the interrupt handler

\* for a SIGINT (CTRL-C).

\*/

#include <stdio.h>

#include <signal.h>

/\*\*

\* Define the method to be called when CTRL-C is pressed.

\*/

void programInterrupted(int i)

{

printf("In the signal interrupt routine\n");

}

/\*\*

\* Main method, shows how to map another function over the CTRL-C

\*/

int main()

{

int i;

/\* The sigset sets the interrupt to another vector (method address).

\* In this case the SIGNT interrupt is set to call the

\* programInterrupted function defined above.

\*/

sigset(SIGINT, programInterrupted);

/\* Now just go into a loop for a few seconds to show what happens when

\* CTRL-C is pressed. Press Control C while this is running and you

\* will see that the interrpts are turned off and are going to the

\* interrup method and printing out an error.

\*/

for (i = 30; i < 45; i++) {

printf("fib(%d) = %d\n", i, fib(i));

}

}

int fib(int n)

{

if (n <= 1)

return 1;

return (fib(n-1) + fib(n-2));

}

Modify this program to handle the SIGALRM (instead of the SIGINT) signal, and set the alarm to go off each 1 microsecond using the ualarm function (e.g. ualarm(1)).  Each time the processAlarm function you set for the SIGALRM is called, increment the value of counter. Then print out the value of counter to measure the amount of time that it took to calculate that Fibonacci number.  Note that this is like the interrupt program we did in C.

public class Fibonacci

{

    private static int counter = 0;

    public static void main(String argv[]) {

       for (int i = 3; i < 45; i++) {

           System.out.println("Num = " + i + "  Fib = " +

             fibonacci(i) + "  time = " + counter);

        }

    }

    public static int fibonacci(int NI) {

        if (NI <= 1) return 1;

        return fibonacci(NI - 1) + fibonacci(NI - 2);

    }

}

**Part III**

Polymorphism in all languages is just using function pointers with some (very nice and useful) syntactic sugar to make sure you are using the pointers correctly. Change the following Java program to make the PolyFunction variables into an array, and call them by element number in the array.

public class PolyFunctionPtrs {

public static void main(String args[]) {

PolyFunction pf1 = new PolyFunction\_1();

PolyFunction pf2 = new PolyFunction\_2();

if (args.length == 0) {

System.out.println("Must be called with a 0 or 1");

System.exit(1);

}

int fNum = Integer.parseInt(args[0]);

if (fNum == 1)

pf1.pfunc();

else

pf2.pfunc();

}

private static interface PolyFunction {

public void pfunc();

}

private static class PolyFunction\_1 implements PolyFunction {

public void pfunc() {

System.out.println("in 1");

}

}

private static class PolyFunction\_2 implements PolyFunction {

public void pfunc() {

System.out.println("in 2");

}

}

}