UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SANTIAGO

LENGUAJES DE PROGRAMACIÓN
Certamen Final (1)

Profesor: José Luis Martí L.
Ayudantes: Sebastián Campos M., Gabriel Carmona T., Sebastián Godínez G.

Parte 1: Verdadero o Falso (2 puntos cada una)

1.- C es un lenguaje imperativo, que permite elementos de la programación orientada al objeto. **(F)**

2.- Java es un lenguaje orientado al objeto, con ciertos elementos propios de un lenguaje imperativo. **(V)**

3.- Scheme es un lenguaje funcional, que incorpora elementos de programación imperativa. **(V)**

4.- Python es un lenguaje scripting, que no permite la orientación a objetos. **(F)**

5.- Si las patentes vehiculares de Chile se pueden describir por la expresión regular /^[A-Z]{2}\.\d{2}\.\d{2}$/, para anteponer un dígito obligatorio se tendría que usar la expresión /^[1-9]*[A-Z]{2}\.\d{2}\.\d{2}$/. **(F)**

6.- En un proceso de compilación, la tabla de símbolos ayuda a resolver las ambigüedades en los árboles sintácticos. **(F)**

7.- La transparencia referencial se refiere a que la evaluación de una función siempre produce el mismo resultado. **(V)**

8.- Una de las desventajas de la declaración implícita de los tipos de datos, es que hace más lenta la ejecución debido a la verificación dinámica de éstos. **(V)**

9.- Una función *f()* en C, que retorna un valor flotante, y que recibe como parámetros un puntero a flotante y un puntero a una función *g()* que, a su vez, recibe un valor entero y devuelve un puntero a entero tendrá como firma a la expresión: *float f(float a, int* g(int))*. **(F)**

10.- En Java para poder acceder a las variables de clase que son definidas mediante la palabra *static*, no es necesario instanciar las clases. **(V)**

11.- Las referencias a una clase hija pueden apuntar a objetos de su clase padre, pero no al revés. **(F)**

12.- Con respecto a la redefinición de métodos en la herencia, un método no se puede ocultar en una subclase. **(F)**

13.- Un ejemplo de polimorfismo dinámico son los subprogramas genéricos, donde éstos utilizan un mismo código (comportamiento) para distintos tipos de parámetros. **(V)**

14.- Un ejemplo de polimorfismo estático es la sobrecarga de operadores. donde un operador tiene el mismo comportamiento, sin importar los tipos de datos de sus operandos. **(F)**

15.- En general, la recursividad de cola es conveniente pues permite ahorrar memoria de *stack*. **(V)**

16.- En Scheme, las variables de nivel superior se definen mediante *define*. **(V)**

17.- En la programación funcional pura, la composición de funciones debe considerar el uso adecuado de variables y operaciones de asignación. **(F)**

18.- En la programación funcional, la repetición debe ser lograda con recursión. **(V)**

19.- En Prolog, no hay posibilidad de evitar que se ejecute el *backtracking* en su totalidad. **(F)**

20.- Dentro de las reglas de calce de Prolog entre dos estructuras, es suficiente que tengan el mismo *functor* y el mismo número de componentes. **(F)**

Parte 2: Alternativas (4 puntos cada una)

21.- ¿Cuál de los siguientes lenguajes no es fuertemente tipado?.
a) Pascal
b) Java
c) Prolog
d) Python

22.- ¿Cuál de los siguientes lenguajes usa comprobación de tipos estática?.
a) C
b) Scheme
c) Perl
d) Python

23.- ¿Qué mecanismo de recolección de basura se debe implantar en un lenguaje de programación donde todo el ligado de memoria es estático o dinámico de *stack*?.
a) Basado en contadores de referencias (enfoque impaciente)
b) Basado en marcas y barrido (enfoque perezoso)
c) Una combinación de ambos, un esquema para la memoria estática y el otro para el *stack*
d) Ninguna de las anteriores

24.- ¿Cuáles de las siguientes frases son ciertas sobre un método de recolección de basura impaciente?
I. Se ejecuta tan pronto se agota la memoria
II. Usa un contador de referencias
III. Se decrementa un contador cuando se pierde una referencia
IV. Requiere de mayor tiempo de ejecución y uso de memoria que un recolector perezoso

a) I y II
b) I, II y III
c) II y IV
d) II, III y IV

25.- ¿En qué situación la instrucción *malloc()* de C podría generar un *stack overflow*?.
a) Un ciclo (loop) infinito (ej.: for(;;) sin control de término)
b) Una función recursiva mal programada
c) Una lista de nodos que creció sin control
d) Nunca

26.- En la siguiente función C, es cierto que:

```
char *f(char *s)
{
    static int k = 0;
    int a = strlen(s);
    char *p = malloc(a – k++);

    return p ? strcpy(p, s+k) : NULL;
}
```

a) *a* es una variable de *stack*, y *s* una de *heap*
b) *k* es una variable estática, y *p* una de *heap*
c) *a* es una variable de *stack*, y **p* una de *heap*
d) *a* es una variable de *heap*, y *p* una de *heap*

27.- Tomar en cuenta el siguiente programa, escrito en sintaxis *tipo Javascript*:

```
// main
var x, y, z;

function sub1() {
var a, y, z;
  …
}

function sub2() {
var a, b, z;
  …
}

function sub3() {
var a, x, w;
  …
}
```

Dada la siguiente secuencia de llamadas en un ámbito dinámico: *main→sub1()→sub3()→sub2()*, ¿cuáles son todas las variables visibles durante la ejecución de *sub2()*?.

a) *a* local, *b* local, *x* de *main*, *y* de *main*, *z* local
b) *a* local, *b* local, *x* de *main*, *y* de *sub1()*, *z* local
c) *a* local, *b* local, *z* local
d) *a* local, *b* local, *w* de *sub3()*, *x* de *sub3()*, *y* de *sub1()*, *z* local

28.- ¿Cuál de las siguientes afirmaciones no es correcta, dependiendo del tipo de paso de parámetros?

```
void f(int a, int b)
{
  a = 1;
  b++;
}
```

```
int main(void)
{
  int i = 10;
  int j = 20;

  f(i, j);
  printf("%d %d", i, j);
}
```

a) Por resultado, la salida del *printf()* de la función *main()* es: 10 20
b) Por referencia, la salida del *printf()* de la función *f()* es: 1 21
c) Por valor, la salida del *printf()* de la función *main()* es: 10 20
d) Por valor-resultado, la salida del *printf()* de la función *f()* es: 1 21

29.- Si una subclase tiene un método sobreescrito con modificador de acceso *protected*, significa que el mismo método en la superclase tiene acceso:

a) Público o protegido
b) Privado o protegido
c) Protegido o de paquete
d) Protegido

30.- Dado el siguiente código en Java, ¿cómo se debe invocar el constructor *Base* para que muestre por la salida estándar el string "base constructor"?

```
class Base
{
  Base(int i)
  {System.out.println("base constructor");}
}

public class Sup extends Base
{
  public static void main(String argv[])
  {
    Sup s= new Sup();
    //Uno
  }

  Sup()
  {
    //Dos
  }

  public void derivada()
  {
    //Tres
  }
}
```

a) Una línea después de //Uno poner: Base(10);
b) Una línea después de //Uno poner: super(10);
c) Una línea después de //Dos poner: super(10);
d) Una línea después de //Tres poner: Base(10);

31.- Suponer que se tiene la siguiente clase Java:

```
public class A {
  int x;
private int y;
  protected int z;
  public int w;
 }
```

Una clase B que la extiende y que se encuentra en otro paquete, ¿cuál sería la visibilidad que tendría sobre los atributos miembros de A?

a) Puede accesar todos los atributos, menos x
b) Puede accesar todos los atributos, menos y
c) Puede accesar todos los atributos, menos x e y
d) Puede accesar todos los atributos, menos x y w

32.- ¿Cuál de los siguientes no es un uso real del modificador final de Java?

a) Indicar que un atributo tendrá un valor constante, que no podrá ser modificado
b) Establecer que un método no puede ser sobreescrito
c) Señalar que una clase no puede ser extendida
d) Reforzar que se trata del último método a ejecutar por un objeto a eliminar

33.- Dadas las siguientes declaraciones:

```
interface I { . . . }
class A implements I { . . . }
class B extends A { . . . }
```

¿cuál de las siguientes sentencias es la única correcta?

a) I x = new I ();
b) I x = new B ();
c) B x = new I ();
d) B x = new A ();

34.- Considerando la siguiente definición de una función:

```
(define funcion
  (lambda (numero)
    (let loop ( (n numero) (resultado 1))
      (cond ( (= n 0) resultado)
            (else (loop (- n 1) (* resultado n)))))))
```

se puede afirmar que:

a) No hay uso del stack, pues sólo se utiliza la memoria estática
b) El stack podría tener problemas, por la alta cantidad de registros de activación incurrida
c) No se presenta ningún problema, porque todo se desarrolla con un único registro de activación
d) No hay uso del stack, pues todo se hace en el heap

35.- Dado el siguiente predicado en Prolog, ¿cuántos llamadas recursivas y qué resultado entregará, para una lista de *n* números enteros y *R* como variable?

```
misterio([ ], R):- R is 0, !.
misterio([X | T] , R):- misterio(T, R1), R is X + R1.
```

a) 1 llamada recursiva, resultado: “True”
b) 1 llamada recursiva, resultado: “False”
c) n llamadas recursivas, resultado: “True”
d) n llamadas recursivas, resultado: “False”

JLML/jlml.
090821.