INFORMÁTICA

Trabajo Práctico N°2 (parte 1) - Introducción a la programación

A – Responda las siguientes preguntas teóricas acerca del lenguaje de programación C.

1) ¿Qué son las palabras reservadas en ANSI C y cuántas existen?
   Las palabras reservadas en ANSI C son identificadores que tienen un significado especial para el compilador y no pueden ser usadas como nombres de variables. Existen 32 palabras reservadas.
2) ¿Cuál es la estructura general de un programa en C?
   Un programa C se organiza en uno o mas archivos fuentes o módulos. Cada archivo se organiza con comentarios al inicio, directivas del procesador, declaraciones de variables y funciones, y por último, definiciones de funciones.
3) ¿Qué es un archivo de encabezado (.h) y cuál es su propósito en C?
   Un archivo .h es un archivo que contiene definiciones de funciones, macros, constantes, tipos de datos, y declaraciones de variables o estructuras que pueden ser utilizadas en múltiples archivos de código. El propósito es facilitar la modularización y reutilización del código.
4) ¿Qué es el preprocesador en C y cuáles son algunas de sus directivas más comunes?
   El preprocesador es la parte del compilador que realiza la primera etapa de traducción o compilación de un archivo C ANSI en instrucciones maquina. El preprocesador procesa el archivos fuente y actua sobras las ordenes, denominadas directivas del preprocesador, incluidas en el programa. Entre las directivas mas comunes se encuentran #include, #define, #if, #ifdef, #else, #endif, etc.
5) ¿Cuáles son los tipos de datos primitivos en C y sus tamaños típicos?
   Los tipos de datos primitivos en C son *enteros, reales y carácter.*
   El tipo char se utiliza para representar caracteres o valores integrales. Las constantes de tipo char pueden ser caracteres encerrados entre comillas ('A', 'b', 'p'). Caracteres no imprimibles (tabulación, avance de página, etc.) se pueden representar con secuencias de escape ('\t', '\f').

Tipo de dato	Tamaño en bytes	Tamaño en bits	Valor mínimo	Valor máximo
`signed char`	1	8	–128	127
`unsigned char`	1	8	0	255
`signed short`	2	16	–32.768	32.767
`unsignet short`	2	16	0	65.535
`signed int`	2	16	–32.768	32.767
`unsigned int`	2	16	0	65.535
`signed long`	4	32	–2.147.483.648	2.147.483.647
`unsigned long`	4	32	0	4.294.967.295

Tabla D.3. Tipos de datos de coma flotante

Tipo de dato	Tamaño en bytes	Tamaño en bits	Valor mínimo	Valor máximo
`float`	4	32	3.4E – 38	3.4E + 38
`double`	8	64	1.7E – 308	1.7E + 308
`long double`	10	80	3.4E – 4932	3.4E + 4932

6) ¿Cómo se manejan las variables globales y locales en C y cuál es su alcance y visibilidad?

   Las variables globales son variables definidas de las cuales pueden hacer uso cualquier función, mientras que las variables locales son variables definidas para una función específica. De esta forma las variables locales están protegidas a la modificación por otras funciones. Son visibles únicamente para la función para la que fue definida.

7) ¿Qué son los operadores en C y cuáles son algunos ejemplos de operadores aritméticos, de asignación y lógicos?

   Los operadores en C son símbolos que permiten realizar operaciones sobre datos y variables.

   Los operadores aritméticos: “+”, “-”, “*”, “div”, “mod”, etc.

   Los operadores de asignación: “=”, “+=”, “*=”, etc.

   Los operadores lógicos: “and”, “not”, “or”, etc.

8) ¿Qué son las estructuras de control del flujo de ejecución en C y cuáles son sus tipos principales?

   Las estructuras de control del flujo de ejecución en C son mecanismos que permiten dirigir el flujo del programa según ciertas condiciones o repetición de acciones. Estas estructuras son fundamentales para implementar la lógica de control en los programas, permitiendo que el código tome decisiones, repita acciones, o salte entre diferentes secciones de código.

9) ¿Qué es una subrutina en C y cómo se define una función?

   Una subrutina es un módulo independientemente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal. Definir una función implica especificar su tipo de retorno, nombre, lista de parámetros y cuerpo de la función.

```
tipo_de_retorno nombre_de_la_funcion(tipo_parametro1 parametro1,
tipo_parametro2 parametro2, ...) {
  // Cuerpo de la función: conjunto de instrucciones
  // que realizan una tarea específica.

  // [opcional] Instrucción return para devolver un valor
  return valor_de_retorno;
}
```

10) ¿Qué es la recursión en C y cómo se estructura un procedimiento recursivo?

    Una función o procedimiento que se puede llamar a sí mismo se llama recursivo. La recursión (recursividad) es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos. Un procedimiento recursivo tiene tres partes: Una condición de salida, el procesamiento del paso recursivo actual y la llamada recursiva al paso siguiente.

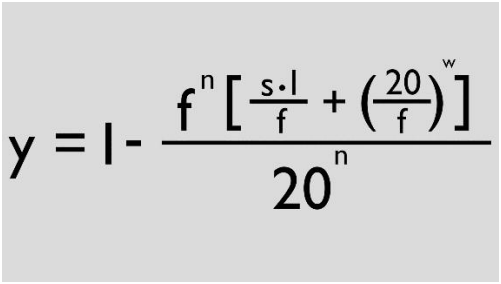
B – Implementación práctica: realice las siguientes actividades referidas al uso del lenguaje de programación C.

1) Identificadores válidos en C Indique cuáles de los siguientes identificadores no son válidos en C y justifique su respuesta:
    a) _data123
    b) 2start No es valido, no puede comenzar con un numero
    c) fluencia0.2 No es valido, no puede contener “.”
    d) mi_var
    e) return No es valido por ser un ANSI C
    f) VarName
    g) var-name No es valido, no puede contener “-”
    h) floatVar
    i) const1
    j) while No es valido por ser un ANSI C
    k) stop@end No es valido, no puede contener “@”
2) Investigue sobre buenas practicas a la hora de nombrar variables (en C y en los lenguajes de programación en general. ¿Qué es el camelCase?¿Cómo deben ser los nombres de las variables?

    Nombrar variables de manera adecuada es crucial para la claridad y mantenibilidad del código.

    camelCase es un estilo de escritura de identificadores (como variables, funciones, etc.) en el que el primer elemento del identificador comienza en minúscula y cada palabra subsiguiente se une sin espacios y comienza con una letra mayúscula.
3) Asigne el tipo de dato más adecuado en C para cada uno de los siguientes valores constantes:
    a) 98.6 float
    b) "hola" char
    c) X char
    d) 1234567890 long
    e) 0 int
    f) “44876523” char
    g) “1.000.000” char
4) Indique cuáles de las siguientes constantes no son válidas en C y justifique su respuesta:
    a) 3.14
    b) -99
    c) 8 + 4 No valido, no es una constante sino que es una expresion
    d) "true"
    e) 1,234 No valido, se usa punto y no coma
    f) “pepe” + “honguito” No valido, no se pueden concatenar cadenas
    g) “dos” * 2 No valido, no se puede multiplicar una cadena por un numero utilizando “ * ”

5) Indique si las siguientes expresiones son válidas y el tipo de dato que retornan en caso de ser válidas. Para las expresiones válidas, realice implementaciones en C que demuestren lo afirmado:
    a) 8 / 2 * 3 + 1 Valido, entero
    b) 10 == 10 || 5 < 3 Valido, int
    c) 7 > 2 > 5 No valido, las comparaciones se evalúan de izquierda a derecha y de forma secuencial, por lo que no se obtendrá el resultado esperado.
    d) 4.5 * “2” No valido, 2 es un char mientras que 4.5 es un float
    e) 15 % 4 = 3 No valido, se esta usando el operador incorrecto, debería ser “==”
    f) 7 - 3 + 2.5 Valido, float
6) Escriba la siguiente expresión algebraica como expresión algorítmica usando el menor número de paréntesis. Considere la función `pow(base, exponente)` para el cálculo de una potencia y la función `sqrt(valor)` para la raíz cuadrada:



pow(f, n-1-w)*(pow(f, w)*s*1+f*pow(20,n))/pow(20,n)

7) Revise los siguientes fragmentos de código, indique si son correctos o existe algún error. De ser correctos indique el resultado.

a)

```
1.      int main() {
2.      int a = 5, b = 10, c;
3.      c = a + b;
4.      printf("%d\n", c);
5.      return 0;
6.      }
```

Es correcto, la salida es 15

b)

```
1.          int main() {
2.          int x = 10;
3.          if (x > 5)
4.          printf("x is greater than 5\n");
5.          else
6.          printf("x is less than or equal to 5\n");
7.          return 0;
8.          }
```

Es correcto, se imprime en pantalla x is greater than 5

c)

```
1.          int main() {
2.          int i;
3.          for (i = 0; i < 10; i++) {
4.          printf("%d ", i);
5.          }
6.          return 0;
7.          }
```

Es correcto, el resultado es: 0 1 2 3 4 5 6 7 8 9

d)

```
1.      int main() {
2.      int a = 5, b = 0;
3.      printf("%d\n", a / b);
4.      return 0;
5.      }
```

Es incorrecto, al dividir por cero se genera un error en tiempo de ejecución.

e)

```
1. int main() {
2. int arr[5];
3.      arr[1] = 1;
4.      arr[2] = 2;
5.      arr[3] = 3;
6.      arr[4] = 4;
7.      arr[5] = 5;
8.      return 0;
9.      }
```

Es incorrecto, se quiere acceder a un elemento fuera de los limites del arreglo.