UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

INE018 MATEMÁTICA COMPUTACIONAL

Solucionario del examen parcial
2024-1

Indicaciones generales:

- Duración: 120 minutos.
- No está permitido el uso de ningún material o equipo electrónico adicional al indicado (no celulares, no tablets, no libros).
- **La presentación, la ortografía y la gramática de los trabajos influirán en la calificación.**

Puntaje total: 20 puntos.

Pregunta 1. (2 puntos)

¿Qué caracteres son utilizados para marcar comentarios en un programa en C++?

Un comentario puede ser texto encerrado por los caracteres `/*` y `*/`. También podemos especificar comentarios que se extenderán hasta el final de una línea comenzando con los caracteres `//`.

Pregunta 2. (2 puntos)

Indique cuáles de los siguientes son nombres válidos de variables en C++:

a. `x`

b. `formula1`

c. `nota_promedio`

d. `%correcto`

e. `int`

f. `entero`

g. `hola mundo`

h. `unaVariableMuyLarga`

i. `20PuntosEnTotal`

j. `nombre-legal`

k. `a7x`

l. `_nombre_ilegal`

Un identificador en C++ satisface lo siguiente:

- El nombre debe comenzar con una letra o un guion bajo.
- El resto de caracteres en el nombre deben ser letras, dígitos o guiones bajos. No está permitido utilizar espacios ni otros caracteres especiales.
- El nombre no puede ser una palabra reservada.

Así, los únicos nombres válidos de variables son `x`, `formula1`, `nota_promedio`, `entero`, `unaVariableMuyLarga`, `a7x` y `_nombre_ilegal`.

Pregunta 3. (2 puntos)

Escriba la forma sintáctica general para cada una de las siguientes estructuras de control: `if`, `while`, `for`. También describa el rol de la sentencia `break`.

```
if (condición) sentencia
if (condición) sentencia else sentencia
while (condición) sentencia
for (inicialización; condición; actualización) sentencia
```

La sentencia `break` tiene el efecto de inmediatamente terminar el bucle contenedor más interior.

Pregunta 4. (2 puntos)

Defina los siguientes términos en relación a las funciones: *llamada*, *argumento*, *retorno*.

El acto de usar el nombre de una función para invocar su código es conocido como *llamar* a la función. Las expresiones encerradas entre paréntesis en la llamada a la función son conocidas como *argumentos*. La operación de regresar al programa que invocó a la función es conocida como *retorno*.

Pregunta 5. (2 puntos)

¿Cuál es la diferencia entre un *caracter* y una *cadena*?

Un caracter es un elemento primitivo que usualmente ocupa 8 bits cuyo valor pertenece a una variante del ISO-646, por ejemplo ASCII, representado en C++ con el tipo fundamental `char`. Una cadena es una instancia de la clase `std::string` que internamente lidia con un arreglo de caracteres, gestiona su memoria, provee métodos para su manipulación, etc.

Pregunta 6. (2 puntos)

Dado el siguiente programa:

```
void Imprimir(string y, string z, string x) {
    cout << z << " y " << x << " comen " << y << endl;
}
```

```
void Misterio(void) {
    string x = "cpp";
    string y = "Manuel";
    string z = "laptop";
    string futbol = "carapulcra";
    string cpp = "Alejandro";
    Imprimir(x, y, z);
    Imprimir(z, x, y);
    Imprimir("futbol", z, cpp);
    Imprimir(y, futbol, "x");
    Imprimir(y, y, "cpp");
}
```

Escriba la salida de cada una de las llamadas tal como aparecerían en la consola.

```
Manuel y laptop comen cpp
cpp y Manuel comen laptop
laptop y Alejandro comen futbol
carapulcra y x comen Manuel
Manuel y cpp comen Manuel
```

Pregunta 7. (2 puntos)

Para cada llamada a la función

```
int Misterio(int a, int b) {
    int c;
    if (a > b) {
        c = a;
    } else if (b % a == 0) {
        c = b;
    } else {
        c = b + (a - (b % a));
    }
    return c;
}
```

escriba el valor retornado:

```
Misterio(4, 2);  // 4
Misterio(5, 4);  // 5
Misterio(5, 13);  // 15
Misterio(5, 17);  // 20
Misterio(4, 8);  // 8
```

Pregunta 8. (2 puntos)

Dado el siguiente procedimiento

```
void Misterio(int i, int j) {
    while (i != 0 && j != 0) {
        i = i / j;
        j = (j - 1) / 2;
        cout << i << " " << j << " ";
    }
    cout << i << endl;
}
```

escriba la salida de cada una de las siguientes llamadas:

```
Misterio(5, 0);
```

```
5
```

```
Misterio(3, 2);
```

```
1 0 1
```

```
Misterio(16, 5);
```

```
3 2 1 0 1
```

```
Misterio(80, 9);
```

```
8 4 2 1 2 0 2
```

```
Misterio(1600, 40);
```

```
40 19 2 9 0 4 0
```

Pregunta 9. (2 puntos)

Escriba una función llamada `TienePuntoMedio` que acepta tres enteros como parámetros y retorna `true` si uno de los enteros es el punto medio entre los otros dos enteros. Tu función debe retornar `false` si no existe tal relación.
Los enteros pueden ser pasados en cualquier orden. El punto medio puede ser el primer, segundo o tercer elemento.
Llamadas como las siguientes deberían retornar `true`:

```
TienePuntoMedio(4, 6, 8);
TienePuntoMedio(2, 10, 6);
TienePuntoMedio(8, 8, 8);
TienePuntoMedio(25, 10, -5);
```

Llamadas como las siguientes deberían retornar `false`:

```
TienePuntoMedio(3, 1, 3);
TienePuntoMedio(1, 3, 1);
TienePuntoMedio(21, 9, 58);
TienePuntoMedio(2, 8, 16);
```

```
bool TienePuntoMedio(int x1, int x2, int x3) {
    // Encontramos el mayor y menor entre los tres números.
    int mayor = max(x1, max(x2, x3));
    int menor = min(x1, min(x2, x3));

    // Calculamos el valor medio como la suma menos los extremos.
    int medio = x1 + x2 + x3 - mayor - menor;

    // Verificamos si el valor medio equidista del mayor y el menor.
    return medio - menor == mayor - medio;
}
```

Pregunta 10. (2 puntos)

Escriba un procedimiento llamado `Armonica` que imprima los primeros términos de la siguiente serie:

$$\sum_{k=1}^{\infty} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots$$

Tu procedimiento debe aceptar un número real como parámetro representando un límite, y debe sumar e imprimir los términos de la sumatoria hasta que la suma de términos coincida o exceda el límite.

Por ejemplo, si a tu procedimiento se le pasa 2.0, imprime los términos hasta que la suma sea mayor o igual a 2. Debes redondear tu respuesta tres dígitos luego del punto decimal.

La salida de llamar `Armonica(2.0)` es la siguiente:

```
1 + 1/2 + 1/3 + 1/4 = 2.083
```

Si a tu procedimiento se le pasa un valor menor a 1, no debes imprimir nada en pantalla. Tu salida debe coincidir con el formato solicitado exactamente; note los espacios y signos más separando términos vecinos.

Llamar `Armonica(0.0)` no debe producir salida alguna.

Llamar `Armonica(1.0)` produce la siguiente salida:

```
1 = 1.000
```

Llamar `Armonica(1.5)` produce la siguiente salida:

```
1 + 1/2 = 1.500
```

Llamar `Armonica(2.7)` produce la siguiente salida:

```
1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 = 2.718
```

Observación: Considerando las sumas parciales $s_n = \sum_{k=1}^{n} 1/k$ podemos demostrar que para todo n se cumple $s_{2n} - s_n \geq 1/2$ y, por ende, la serie no es convergente. Este resultado nos permite concluir que s_n aumenta ilimitadamente conforme aumenta su número de

términos. Sin embargo, esto ocurre de forma muy lenta. De manera precisa, con un poco de conocimiento de cálculo integral es fácil obtener que

$$\ln(n+1) < 1 + \frac{1}{2} \cdots + \frac{1}{n} < \ln(n) + 1$$

para todo n mayor o igual que 2. Así, por ejemplo, son necesarios más de 10000 términos para que la suma sea mayor o igual que 10, y son necesarios más de 10^{43} términos para que la suma sea mayor o igual que 100.

```
void Armonica(long double limite) {
    // Si el límite es menor a 1, no se imprime nada.
    if (limite < 1.0L) {
        return;
    }

    // Inicializamos la suma parcial y nuestro índice.
    long double suma = 0.0L;
    int k = 0;

    // Iteramos y sumamos cada término hasta que
    // la suma alcance o exceda el límite.
    while (suma < limite) {
        k++;

        // Imprimimos el término actual en el formato adecuado.
        if (k == 1) {
            cout << '1';
        } else {
            cout << " + 1/" << k;
        }

        // Acumulamos el término actual en la suma parcial.
        suma += 1.0L / k;
    }

    // Imprimimos la suma parcial con tres dígitos de precisión.
    cout << " = " << fixed << setprecision(3) << suma << endl;
}
```

Profesor del curso: Manuel Loaiza Vasquez.

Lima, 4 de junio de 2024.