INF-134 Estructuras de Datos, 2024-2

Tarea 1

Profesores: Andrés Navarro, Ricardo Salas, Juan-Pablo Castillo
andres.navarrog@usm.cl, ricardo.salas@usm.cl, juan.castillog@sansano.usm.cl

Ayudantes:
Gabriel Escalona (gabriel.escalona@usm.cl)
Tomás Barros (tomas.barros@sansano.usm.cl)
Jaime Inostroza (jinostroza@usm.cl)
Juan Alegría (juan.alegria@usm.cl)
Carlos Lagos (carlos.lagosc@usm.cl)

Fecha de entrega: 01 de Septiembre, 2024.
Plazo máximo de entrega: 1 día.

1. Reglas

- La presente tarea debe hacerse en equipos de dos personas. Toda excepción a esta regla está sujeta a autorización del ayudante coordinador Gabriel Escalona.
- Debe usarse el lenguaje de programación **C++**. Al realizar la evaluación, las tareas serán compiladas usando el compilador **g++**, usando la línea de comando `g++ archivo.cpp -o output -Wall`. No se aceptarán variantes o implementaciones particulares de **g++**, como el usado por **MinGW** (normalmente ocupado en **Dev C++**). Se deben seguir los tutoriales disponibles en Aula USM.
- No se permite usar la biblioteca STL, así como ninguno de los contenedores y algoritmos definidos en ella (e.g. `vector, list`, etc.). Está permitido usar otras funciones de utilidad definidas en bibliotecas estándar, como por ejemplo `math.h, string, fstream, iostream,` etc.
- Recordar que una única tarea en el semestre puede tener nota menor a 30. El incumplimiento de esta regla implica reprobación del curso.

2. Objetivos de aprendizaje

Comprender y familiarizarse con las estructuras y tipos de datos básicos que provee el Lenguaje de Programación C++. Entre los conceptos mas importantes, se encuentran:

- Paso de parámetros por valor.
- Paso de parámetros por referencia.
- Asignación de memoria dinámica.
- Manipulación de punteros.
- Manejo de Archivos.

Además se fomentará el uso de las buenas prácticas y el orden en la programación de los problemas correspondientes.

3. Problema a resolver

En esta sección deben implementarse funciones y clases en C++, de acuerdo a las siguientes descripciones. Su propuesta de solución debe ser entregada en un archivo(s) `.cpp` (y correspondientes `.hpp` de ser necesario). Su función `main` debe ser capaz de leer un programa según lo especificado a continuación y recuerde que se probarán múltiples ejemplos para verificar si su solución ejecuta bien los programas que recibe.

3.1. Instrucciones

En la siguiente tarea crearemos nuestra propia versión de un curioso lenguaje basado en punteros. Para esto utilizarán lo que han aprendido sobre TDA y punteros.

3.2. El Lenguaje

Consiste de un puntero que apunta a un arreglo de largo n, de números enteros con valor inicial 0. Este lenguaje solo posee 9 operadores:

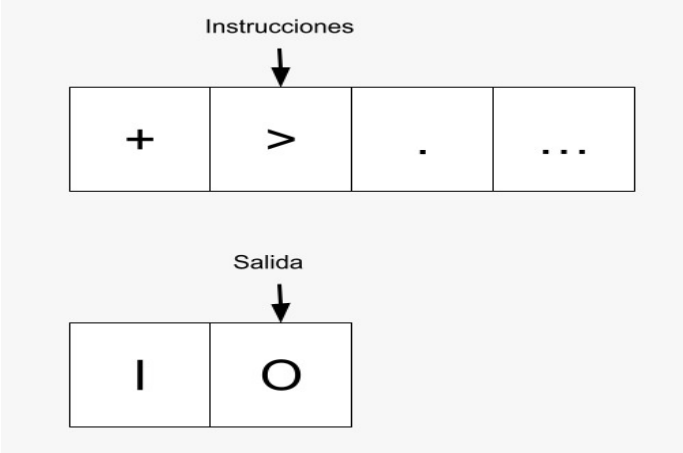
Operador	Descripción	Equivalencia C++
>	Incrementa el puntero.	++ptr
<	Decrementa el puntero.	- -ptr
+	Incrementa en uno lo apuntado.	++(*ptr)
-	Decrementa en uno lo apuntado.	- -(*ptr)
.	Muestra en pantalla el valor numérico de lo apuntado.	*ptr
:	Muestra en pantalla el carácter de lo apuntado.	(char)*ptr
[	Inicio de bucle, entra si lo apuntado no es 0.	while(*ptr){
]	Fin de bucle, vuelve a [si lo apuntado no es 0.	}
!	Fin de la ejecución.	return 0

Para simplificar el problema **NO** se anidarán bucles y se usará la siguiente tabla de valores para pasar de entero a carácter

Valores	Caracteres
0	espacio
[1-26]	[a - z]
[27-52]	[A - Z]
[53-62]	[0 - 9]
[63-71]	{.:+-<>[]!}

Hints: Use un arreglo de char para almacenar las codificaciones de la tabla. Para convertir un número que no está en la tabla debe utilizar el módulo (positivo) de 72.

3.3. La Tarea



Su tarea debe ser capaz de utilizar un intérprete para cargar y ejecutar distintos programas, estos programas se ingresarán mediante a la lectura de un archivo llamado Programas.txt y se almacenarán al principio de la ejecución. Luego, mediante a entrada por consola, se ingresaran instrucciones para la ejecución de estos programas. Las instrucciones son las siguientes:

- **c n**: Cargar programa, donde n es el índice de los programas en el archivo
- **e**: Ejecutar programa cargado.
- **m**: Mostrar el programa cargado.
- **s**: Terminar la ejecución del ejecutable.

El archivo Programas.txt debe tener el siguiente formato:

- La primera línea del archivo contiene el largo máximo que puede tener la salida.
- La segunda línea contiene la cantidad de programas que desea guardar.
- Desde la tercera línea se presentan cada uno de los programas.

3.4. Implementacion

Se usarán dos TDA's para esta tarea

```
class Programa{
private:
    int largo_operaciones;
    char* puntero_operaciones;
    int* puntero_salida;
    char* operaciones;

public:
    void ejecutar_operador();
    void mover(char dir);
    void asignar(int valor, int* salida);
    char obtener();
    void terminar_programa();
    Programa(int largo_operaciones);
    void ejecutar();
}
```

La primera clase es Programa, la cual consiste en un valor `int` el cual almacena el largo del arreglo de operaciones, dos punteros los cuales señalan la posición actual en la que se está ejecutando y la posición que se está escribiendo en la salida, además de un puntero a un arreglo que almacenará las operaciones.
También se deberán implementar los siguientes métodos:

- `Programa(int largo_operaciones)`: El constructor de la clase, en la cual se debe iniciar los arreglos a utilizar.
- `ejecutar_operador()`: Lee el operador que se encuentra en la posición del puntero_operaciones y la ejecuta.
- `mover(char dir)`: mueve el puntero de salida en la dirección dir (>(derecha), <(izquierda))
- `asignar(int valor, int* salida)`: asigna valor numérico a posición actual de puntero_salida.
- `obtener()`: Obtiene el caracter de la posición actual.
- `terminar_programa()`: Libera la memoria de utilizada por la clase.
- `ejecutar()`: comienza la ejecución del programa.

```
class Interprete{
private:
    int cant_programas;
    int largo_salida;
    int cargado;
    Programa* programas;
    int* salida;

public:
    Interprete(int cant_programas, int largo_salidas);
    void cargar_programa(int n);
    void ejecutar_programa();
    void mostrar_programa_cargado();
    void terminar_ejecucion();
}
```

La segunda clase es Interprete, la cual va a almacenar la cantidad de programas, el largo de la salida, el programa cargado actualmente, un puntero a un arreglo con la salida y otro puntero a un arreglo de programas, los cuales van a ser los programas leídos del archivo.
El arreglo de salida deberá ser reiniciado cada vez que se cargue un programa. Para esta clase se deberá implementar los siguientes métodos:

- `Interprete(int cant_programas, int largo_salidas)`: Constructor de la clase, encargado de crear arreglo programas.
- `cargar_programa(int n)`: Carga el programa número n.
- `ejecutar_programa()`: Ejecuta el programa cargado.
- `mostrar_programa()`: Muestra el programa cargado.
- `terminar_ejecucion()`: Encargado de liberar la memoria utilizada durante la ejecución.

Para ambas clases se pueden agregar atributos extra y crear métodos auxiliares si es que son necesarios. A los métodos que sí deben implementar no se les pueden agregar parámetros adicionales a los definidos.

También se pueden crear funciones auxiliares si es que son necesarias.

3.5. Ejemplos de entrada y salida

Ejemplo 1:

A continuación se presenta un ejemplo de entrada y salida de los datos. Primero se muestra el contenido del archivo "Programas.txt":

```
10
2
+>.--.++:---:!
.>++++++++.[<+>-]<.>.!
```

Luego se muestra un ejemplo de ejecución, utilizando el archivo de ejemplo (>significa input)

```
> c 0
> e
0-2 c
> c 1
> e
0880
> e
0880
> s
```

Ejemplo 2:
Archivo Programas.txt:

```
15
3
+++++[>++++<-]>-:<+++++:<++:-:>>:+:<++++:<:>>------:!
.>++++++++.[<+>-]<.>.!
+++[>+++<-]>+:+++++:>++++++++:<-:!
```

Ejecución:

```
> m
sin instrucciones
> e
> c 1
> m
. > + + + + + + + + .[< + > -] < . > .!
> c 0
> e
sebastian
> c 2
> e
john
> s
```

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido denominada `tarea1-apellido1-apellido2.zip` (reemplazando sus apellidos según corresponda), en cuyo interior debe estar la tarea dentro de una carpeta también llamada `tarea1-apellido1-apellido2`, a la página `aula.usm.cl` del curso, el cual contenga:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- El archivo **nombres.txt** que debe contener Nombre, ROL y Paralelo de cada miembro del equipo detallando qué programó cada uno.
- **README.txt**, Instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

Considere además que:

- El **apellido1** es el primer apellido de uno de los integrantes y el **apellido2** es el primer apellido del otro integrante. Esto significa que el **apellido1** y **apellido2** no son los dos apellidos de alguno de los integrantes.
- Solo una persona del grupo debe subir la tarea al aula, no hacer esto, puede conllevar un descuento.

5. Entrega mínima

La entrega mínima permite obtener una nota 30 si cumple con los siguientes requisitos:

- Se manejan archivos (Se cargan los programas y se muestran por pantalla)
- Se implementan las instrucciones de +,- y .

6. Restricciones y Consideraciones

- Se permite un único día de atraso en la entrega de la tarea. La nota máxima que se puede obtener en caso de entregar con un día de atraso es nota 50.
- Las tareas que no compilen no serán revisadas y deberán ser re-corregidas por el ayudante coordinador.
- Debe usar **obligatoriamente** alguna de las formas de compilación indicadas en los tutoriales entregados en Aula USM.
- Por cada Warning en la compilación se descontarán 5 puntos en la nota.
- Si se detecta **COPIA** la situación será revisada por ayudante y profesor coordinador.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos ítems no se cumple.

7. Directrices de programación

Las directrices corresponden a buenas prácticas de programación, las cuales serán tomadas en consideración para la evaluación de la tarea. El código fuente del programa debe estar estructurado adecuadamente en archivos (separados de ser necesario). Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota.

Cada función programada debe tener comentarios de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Por cada comentario faltante, se restarán 5 puntos.

Por último, la indentación (1 TAB o 4 espacios), es muy importante. Por **cada bloque mal indentado, se quitarán 10 puntos**.