Práctica 09
Introducción a Ciencias de la Computación 2024

Ayudante: David Antonio de la Rosa Hernández
david.delarosa@cimat.mx

18 de octubre de 2024

Evaluación

La fecha de entrega es el **domingo 20 de octubre a las 11:59 p. m.**, sin envíos tardíos. Esta es una práctica para aplicar conceptos. En caso de no tener la práctica completa, enviar su avance dentro del tiempo. Recibirán calificación y retroalimentación. Para dudas, pueden contactarme por correo.

Entrega

Entregar ÚNICAMENTE un ZIP con el siguiente formato **Apellidos_CC_Practica09.zip**
(ej. DelaRosaHernandez_CC_Practica09.zip), que contenga:

- **(lista_tuprimerapellido.c)**: El archivo en **C** con el código que resuelve el problema planteado. NO ENTREGAR EJECUTABLES NI ARCHIVOS CPP.

Simulación de una Lista de Python en C

En este ejercicio vamos a simular una lista dinámica similar a las listas en Python, usando estructuras en C. Las listas en Python pueden cambiar de tamaño dinámicamente, permiten agregar, modificar, acceder y obtener elementos. Implementaremos una estructura en C para lograr este comportamiento.

Objetivos

1. Crear una estructura que simule una lista dinámica de enteros en C.
2. Implementar las funciones básicas para trabajar con la lista (agregar, modificar, obtener elementos, y obtener el tamaño y la capacidad).
3. Practicar el uso de **memoria dinámica** y **punteros**.

Ejercicios

1. Implementa una estructura llamada **Lista**, que contenga:
   - Un apuntador a un array dinámico de enteros (**int *data**).
   - Un entero que indique la cantidad actual de elementos (**int size**).
   - Un entero que indique la capacidad máxima actual (**int capacity**).
2. Implementa las siguientes funciones:
   a) **list**: Inicializa la lista con una capacidad dada.
   b) **append**: Agrega un nuevo elemento al final de la lista, en caso de necesitar más espacio reserva el doble de memoria, copia los datos y libera la memoria obsoleta.
   c) **set**: Asigna un elemento en una posición específica.
   d) **get**: Obtiene el elemento en una posición específica.
   e) **size**: Retorna el número actual de elementos en la lista.

f) **capacity**: Retorna el número máximo de elementos almacenables en la lista.

g) **clear**: Libera la memoria usada por la lista y deja la estructura en un estado vacío.

3. Escribe un programa en C que:
   - Cree una lista con capacidad inicial de 4 elementos.
   - Agregue los valores 10, 20, 30, 40 y 50 a la lista usando la función **append**.
   - Cambia el valor del elemento en el indice 3 usando la función **set**
   - Imprima el tamaño actual y máximo de la lista.
   - Imprima el valor del elemento con el índice 1 usando la función **get**.

Ejemplo de Código

A continuación, se proporciona un ejemplo de código para guiarte en la implementación:

```c
#include <stdio.h>
#include <stdlib.h>

// Definición de la estructura de la lista
typedef struct {
    int *data;
    int size;
    int capacity;
} Lista;

// Función para inicializar la lista
Lista* list(int capacidad_inicial) {
    Lista *lista;
    return lista;
}

// Función para agregar elementos (append)
void append(Lista *lista, int valor) {
}

// Función para cambiar un elemento (set)
void set(Lista *lista, unsigned int indice, int valor) {
}

// Función para obtener un elemento (get)
int get(Lista *lista, unsigned int indice) {
}

// Función para obtener el tamaño actual (size)
int size(Lista *lista) {
}

// Función para obtener el tamaño máximo (capacity)
int capacity(Lista *lista) {
}

// Función para liberar la memoria de la lista (clear)
void clear(Lista *lista) {
    free(lista->data);
    free(lista);
}

// Programa principal
int main() {
```

```
    Lista *mi_lista = list(4);

    append(mi_lista, 10);
    append(mi_lista, 20);
    append(mi_lista, 30);
    append(mi_lista, 40);
    append(mi_lista, 50);

    set(mi_lista, 3, 20);

    printf("Tamaño actual: %d\n", size(mi_lista));

    printf("Elemento en el índice 2: %d\n", get(mi_lista, 1));

    clear(mi_lista);
    return 0;
}
```

Consideraciones

- Usa **memoria dinámica** y gestiona los apuntadores con cuidado.
- Libera toda la memoria utilizada al finalizar.
- Implementa chequeos de error en las funciones (índices fuera de rango, elementos no encontrados).