Россиский университет дружбы народов

Курсавая работа по информатике и програмированию

Фио: нхага кевин карлинью хосе

Группа: ИУСБД-01-23

Тема: динамичуский массив

Москва 24

1. Введение: Почему работа актуальна для меня

Работа с динамическими массивами является важной темой в программировании, потому что динамические массивы предоставляют нам гибкость в управлении данными, их размерами и памятью. В отличие от статических массивов, которые требуют заранее определённого размера, динамические массивы могут адаптироваться к изменениям данных. Это становится особенно актуальным в ситуациях, когда количество элементов заранее неизвестно, например, при обработке входных данных в реальном времени или при работе с большими объемами данных.

Почему это важно для меня как программиста:

Гибкость: Динамические массивы позволяют работать с данными, количество

которых может изменяться во время выполнения программы.

Эффективность использования памяти: Программирование с динамическими массивами позволяет оптимизировать использование памяти и избежать переполнения или излишков памяти.

Часто используемая структура: Динамические массивы — это основа множества более сложных структур данных (например, стека, очереди, списка).

2. Основная часть: Теория

Что такое динамический массив?

Динамический массив — это структура данных, которая позволяет хранить элементы, размер которой можно изменять во время выполнения программы. В отличие от статического массива, где размер фиксируется при создании, динамический массив может расширяться или сокращаться по мере необходимости, что делает его более гибким.

Основные характеристики динамического массива:

Гибкость: Размер массива может увеличиваться или уменьшаться в зависимости от текущих потребностей.

Управление памятью: Динамический массив эффективно использует память, выделяя её только тогда, когда это необходимо.

Перераспределение памяти: Когда массив достигает своей емкости, происходит перераспределение памяти — старые элементы копируются в новый массив, который может быть в два раза больше по размеру.

Как работает динамический массив?

Инициализация массива: Когда создается динамический массив, ему назначается начальная емкость, которая может быть небольшой.

Добавление элемента: Когда добавляется новый элемент, если текущий размер массива равен его емкости, массив увеличивается (обычно в два раза). Это делается для того, чтобы минимизировать частоту перераспределений, так как каждый раз копирование массива — это дорогостоящая операция.

Удаление элемента: Если количество элементов уменьшается, то размер массива может быть уменьшен, чтобы эффективно использовать память.

Перераспределение массива: Когда массив переполняется, создается новый массив с увеличенной емкостью, и элементы копируются в новый массив. Обычно емкость увеличивается в два раза.

Преимущества динамического массива:

Гибкость: Легко добавлять новые элементы, не зная заранее их точное количество.

Эффективность: За счет уменьшения частоты перераспределений, операция добавления элемента становится эффективной.

Использование памяти: Массивы могут быть растянуты только тогда, когда в этом есть потребность.

Недостатки динамического массива:

Перераспределение памяти: Когда массив переполняется, происходит дорогостоящая операция — копирование элементов в новый массив.

Невозможность использования памяти напрямую: В некоторых случаях, если память перераспределяется слишком часто, это может повлиять на производительность программы.

3. Практическая часть: Реализация динамического массива

Для практического понимания теории динамических массивов, давайте рассмотрим пример реализации динамического массива на языке C++.

Код на C++:

```cpp
#include <iostream>
using namespace std;

class DynamicArray {
private:
    int* arr;        // Указатель на массив
    int size;        // Текущий размер массива
    int capacity;    // Ёмкость массива

public:
    // Конструктор, инициализирует массив с заданной начальной ёмкостью
    DynamicArray(int initial_capacity = 2) {
        size = 0;
        capacity = initial_capacity;
        arr = new int[capacity];  // Выделение памяти для массива
    }
```

```cpp
    // Метод для добавления элемента в массив
    void add(int value) {
        if (size == capacity) {  // Если массив переполнен
            resize();            // Увеличиваем размер массива
        }
        arr[size++] = value;  // Добавляем элемент в конец
    }

    // Метод для изменения размера массива
    void resize() {
        capacity *= 2;  // Увеличиваем ёмкость массива в два раза
        int* new_arr = new int[capacity];  // Создаём новый массив с больш
        for (int i = 0; i < size; i++) {
            new_arr[i] = arr[i];  // Копируем старые элементы в новый масс
        }
        delete[] arr;  // Освобождаем па[illegible] старого массива
```

```
        arr = new_arr;  // Присваиваем указатель на новый массив
    }

    // Метод для получения элемента по индексу
    int get(int index) {
        if (index >= 0 && index < size) {
            return arr[index];  // Возвращаем элемент по индексу
        }
        throw out_of_range("Index out of range");  // Если индекс выходит за п
    }

    // Метод для получения текущего размера массива
    int getSize() {
        return size;
    }

    // Деструктор для освобождения памяти
    // Деструктор для освобождения памяти
    ~DynamicArray() {
        delete[] arr;  // Освобождаем память массива при уничтожении объекта
    }
};

int main() {
    DynamicArray arr;

    // Добавляем элементы в динамический массив
    arr.add(10);
    arr.add(20);
    arr.add(30);

    // Выводим все элементы массива
    for (int i = 0; i < arr.getSize(); i++) {
```

аапар

```
    // Выводим все элементы массива
    for (int i = 0; i < arr.getSize(); i++) {
        cout << arr.get(i) << " ";  // Печатаем каждый элемент
    }

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>  // Para as funções malloc, realloc e free

// Estrutura para representar o array dinâmico
typedef struct {
    int* arr;        // Ponteiro para o array
    int size;        // Tamanho atual do array
    int capacity;    // Capacidade máxima do array
} DynamicArray;

// Função para inicializar o array dinâmico
void initDynamicArray(DynamicArray* da, int initial_capacity) {
    da->size = 0;
    da->capacity = initial_capacity;
    da->arr = (int*)malloc(da->capacity * sizeof(int));  // Aloca memória
    if (da->arr == NULL) {
        printf("Erro de alocação de memória.\n");
        exit(1);  // Finaliza o programa se não for possível alocar memória
    }
}

// Função para redimensionar o array
void resize(DynamicArray* da) {
    da->capacity *= 2;  // Dobra a capacidade
    da->arr = (int*)realloc(da->arr, da->capacity * sizeof(int));  // Realoca memória
    if (da->arr == NULL) {
        printf("Erro de alocação de memória.\n");
        exit(1);  // Finaliza o programa se não for possível alocar memória
    }
}

// Função para adicionar um elemento ao array
void add(DynamicArray* da, int value) {
    if (da->size == da->capacity) {  // Se o array estiver cheio
        resize(da);  // Redimensiona o array
    }
    da->arr[da->size++] = value;  // Adiciona o valor e incrementa o tamanho
}

// Função para obter um elemento no índice especificado
```

```c
// Função para obter um elemento no índice especificado
int get(DynamicArray* da, int index) {
    if (index >= 0 && index < da->size) {
        return da->arr[index];  // Retorna o valor no índice
    } else {
        printf("Índice fora do alcance.\n");
        exit(1);  // Finaliza o programa em caso de índice inválido
    }
}

// Função para obter o tamanho atual do array
int getSize(DynamicArray* da) {
    return da->size;
}

// Função para liberar a memória do array
void freeDynamicArray(DynamicArray* da) {
    free(da->arr);  // Libera a memória alocada
}

int main() {
    DynamicArray da;

    // Inicializa o array dinâmico com capacidade 2
    initDynamicArray(&da, 2);

    // Adiciona elementos ao array dinâmico
    add(&da, 10);
    add(&da, 20);
    add(&da, 30);  // Esse vai forçar o redimensionamento do array

    // Imprime os elementos do array
    for (int i = 0; i < getSize(&da); i++) {
        printf("%d ", get(&da, i));  // Exibe o valor no índice i
    }
    printf("\n");

    // Libera a memória do array dinâmico
    freeDynamicArray(&da);
```

```c
    // Libera a memória do array dinâmico
    freeDynamicArray(&da);

    return 0;
}
```

4. Литература

Для углубленного понимания динамических массивов использовал следующие источники:

1. Т. Х. Кормен, Ч. Е. Лейзерсон, Р. L. Ривест, К. Stein. "Алгоритмы: построение и анализ". — М.: "Мир", 2002. — Книга, посвященная основным алгоритмам и структурам данных, включая динамические массивы.
2. Р. Седжвик. "Алгоритмы на С". — СПб.: "БХВ-Петербург", 2005. — Описание алгоритмов и структур данных с использованием языка программирования С.
3. Б. Страуструп. "Язык программирования С++". — М.: "Вильямс", 2005. — Руководство по языку С++ и основам работы с памятью и структурами данных.

Заключение

Динамический массив — это одна из самых популярных и важных структур данных, предоставляющая гибкость в работе с памятью и эффективностью обработки данных. Правильное использование динамических массивов позволяет создавать более производительные и адаптивные программы, что делает эту тему важной для каждого разработчика.