ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»

Инженерная академия

Департамент механики и процессов управления

КУРСОВАЯ РАБОТА

по дисциплине “Информатика и программирование”

Тема: “Реализация динамического массива на языке программирования C”

Студенты:

Кубанцева Арина Олеговна, Вержбицкая Надежда Андреевна

Группа: ИУСбд-02-23

СОДЕРЖАНИЕ

1. Введение…………………………………………………………………………….3
2. Теоретическая часть………………………………………………………………..4
3. Практическая часть………………………………………………………………...6
4. Вывод………………………………………………………………………………19
5. Источники…………………………………………………………………………20

ВВЕДЕНИЕ:

Цель: Изучить и реализовать работу динамического массива в языке программирования C, исследовать его основные возможности и ограничения, а также разработать программу, демонстрирующую применение динамического массива для решения практических задач.

Задачи:

1. Изучить теоретические основы работы с динамической памятью в языке C, включая использование функций malloc, calloc, realloc и free.
2. Проанализировать преимущества и недостатки динамических массивов по сравнению с другими структурами данных.
3. Разработать алгоритм для управления динамическим массивом, включающий создание, изменение размера и освобождение памяти.
4. Написать программу, реализующую динамический массив с функциями добавления, удаления элементов, вычисление суммы и среднего значения элементов массива, нахождение минимального и максимального элемента, изменения количества элементов в динамическом массиве.

Динамический массив является важным инструментом в языке программирования C, позволяя эффективно работать с данными переменного размера. В отличие от статических массивов, размер которых задается на этапе компиляции, динамические массивы могут изменять свой объем во время выполнения программы, что делает их более гибкими и удобными для многих задач.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Для реализации создания и редактирования динамического массива в языке программирования C используется заголовочный файл <stdlib.h>. Основными функциями для работы с динамическим выделением памяти являются:

* malloc (memory allocation) — выделяет блок памяти заданного размера.
* calloc (contiguous allocation) — выделяет память для массива и инициализирует все элементы нулями.
* realloc (reallocation) — изменяет размер уже выделенного блока памяти.
* free — освобождает выделенную память.

Преимущества и недостатки динамического массива

Преимущества:

* Гибкость в размере: возможность изменять объем хранимых данных в процессе работы программы.
* Экономия памяти: можно выделять ровно столько памяти, сколько необходимо.

Недостатки:

* Ручное управление памятью: программист обязан следить за освобождением выделенной памяти, чтобы избежать утечек.
* Увеличение размера массива требует выделения нового блока памяти и копирования данных, что может быть ресурсоемким.

Сравнение с альтернативными структурами данных

#### 1. Динамический массив и связный список

Динамический массив обеспечивает быстрый доступ к элементам по индексу (O(1)), но добавление или удаление элементов, кроме конца, требует сдвига данных (O(n)). Связный список, напротив, позволяет легко добавлять и удалять элементы в любом месте (O(1), если известен узел), но доступ к элементу осуществляется последовательно (O(n)). Связный список лучше подходит для частого изменения структуры данных, тогда как массив удобен для работы с упорядоченными данными.

#### 2. Динамический массив и стек

Динамический массив позволяет произвольный доступ к элементам, но менее эффективен для операций с вершиной, так как добавление или удаление элементов требует перераспределения памяти. Стек оптимизирован для работы с последним добавленным элементом (операции добавления и удаления имеют сложность O(1)) и используется для задач с дисциплиной доступа LIFO (например, обработка вызовов функций).

#### 3. Динамический массив и хеш-таблица

Динамический массив предоставляет доступ к элементам по индексу, что делает его удобным для упорядоченных данных. Хеш-таблица обеспечивает быстрый доступ по ключу (O(1) в среднем), но требует больше памяти и управления для обработки коллизий. Она предпочтительнее для задач, где данные индексируются уникальными ключами.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Реализация динамического массива на языке C с функциями для работы и выполнения операций.

Код, представленный далее представляет собой файл, в котором пользователь может создать массив с динамическим выделением памяти, а также использовать операции в работе с ним.

Реализованные функции:

* Создание динамического массива и заполнение его псевдо-случайными числами или нулями.
* Добавление элемента в список с расширением памяти.
* Удаление элемента массива по индексу.
* Нахождение среднего значения в массиве.
* Нахождение максимального и минимального элемента массива.
* Вывод массива.
* Изменение размера массива.

Объявление заголовков:

#include <stdio.h>

#include <stdlib.h> //Заголовочный файл для реализации динамического массива

#include <time.h>

Создание структуры для массива:

typedef struct {

int \*array; //объявление массива

size\_t size; //объявление размера массива

size\_t capacity; //объявление вместимости

} dynamic\_array; //динамический массив

Функция создания динамического массива:

dynamic\_array\* create\_dynamic\_array(size\_t initial\_capacity) { //функция создания динамического массива

dynamic\_array \*da = (dynamic\_array\*)malloc(sizeof(dynamic\_array)); //использование функции malloc для выделения блока памяти

(\*da).array = (int\*)malloc(initial\_capacity \* sizeof(int));

(\*da).size = 0;

(\*da).capacity = initial\_capacity;

return da;

}

Функция освобождения памяти массива:

void free\_dynamic\_array(dynamic\_array \*da) { //функция освобождения выделенной памяти с помощью free

free((\*da).array);

free(da);

}

Функция добавления элемента в массив:

void append(dynamic\_array \*da, int value) { //функция добавления элемента

if ((\*da).size == (\*da).capacity) {

(\*da).capacity \*= 2; //увеличение вместимости в 2 раза

(\*da).array = (int\*)realloc((\*da).array, (\*da).capacity \* sizeof(int)); //перевыделение памяти под массив при помощи realloc

}

(\*da).array[(\*da).size++] = value;//добавление элемента в массив

}

Функция удаления элемента из массива со сдвигом:

void deletion(dynamic\_array \*da, size\_t index) { //функция удаления элемента

if (index >= (\*da).size) { //если номер индекса выходит за рамки размера массива вывести ошибку

printf("error: index out of range\n");

return;

}

for (size\_t i = index; i < (\*da).size - 1; i++) { // цикл начинает с индекса удаляемого элемента и проходит до предпоследнего элемента массива

(\*da).array[i] = (\*da).array[i + 1]; // перемещение каждого элемента массива на одну позицию влево, заменяя удаляемый элемент

}

(\*da).size--; //уменьшение размера массива на единицу после удаления элемента

}

Функция нахождения суммы значений элементов массива:

int sum(dynamic\_array \*da) { //функция поиска суммы элементов массива

int total = 0; //объявление переменной, хранящей значение суммы

for (size\_t i = 0; i < (\*da).size; i++) { //цикл для перебора всех элементов динамического массива от первого до последнего

total += (\*da).array[i]; // Суммирование значения, хранящемся в переменной со значением элемента

}

return total; //возвращение суммы

}

Функция нахождения среднего значения элементов массива:

double average(dynamic\_array \*da) { // функция нахождения среднего значения элементов массива

if ((\*da).size == 0) { // проверка массива на заполненность

return 0;

}

return (double)sum(da) / (\*da).size; //возвращение среднего значения, используя функцию суммы

}

Функция поиска минимального и максимального элемента массива:

void find\_min\_max(dynamic\_array \*da, int \*min, int \*max) { //функция поиска минимального и максимального значения

if ((\*da).size == 0) { //проверка массива на заполненность

\*min = \*max = 0;

return; //вывод нулевых значений минимума и максимума

}

\*min = \*max = (\*da).array[0]; //присвоение переменным максимума и минимума значение 1 элемента

for (size\_t i = 1; i < (\*da).size; i++) { //цикл для перебора всех элементов динамического массива от первого до последнего

if ((\*da).array[i] < \*min) { //проверка элемента "меньше минимума"

\*min = (\*da).array[i];

}

if ((\*da).array[i] > \*max) {//проверка элемента "больше максимума"

\*max = (\*da).array[i];

}

}

}

Функция вывода массива (значений массива):

void print\_array(dynamic\_array \*da) { //функция вывода массива

for (size\_t i = 0; i < (\*da).size; i++) {//цикл для перебора всех элементов динамического массива от первого до последнего

printf("%d ", (\*da).array[i]); //вывод элемента

}

printf("\n");

}

Функция (пере)выделения памяти и заполнения массива:

void resize\_array(dynamic\_array \*da, size\_t new\_size) { //функция перевыделения памяти под массив при помощи realloc

if (new\_size > (\*da).capacity) {

(\*da).capacity = new\_size;

(\*da).array = (int\*)realloc((\*da).array, (\*da).capacity \* sizeof(int));

}

if (new\_size > (\*da).size) {

for (size\_t i = (\*da).size; i < new\_size; i++) {

(\*da).array[i] = rand() % 10000; //заполнение массива псевдо-случайными числами

}

} else if (new\_size < (\*da).size) {

(\*da).size = new\_size;

}

(\*da).size = new\_size;

}

void fill\_array\_with\_zeros(dynamic\_array \*da, size\_t size) { //функция перевыделения памяти для заполненного нулями массива при помощи calloc

(\*da).array = (int\*)calloc(size, sizeof(int));

(\*da).size = size;

(\*da).capacity = size;

}

Вывод меню для исполнения операций над динамическим массивом с использованием оператора switch:

int main() {

srand(time(NULL));

dynamic\_array \*da = NULL;

int choice;

int first\_choice = 1;

//Вывод меню с доступными операциями над динамическим массивом

while (1) {

printf("1. enter array size\n");

printf("2. add element to array\n");

printf("3. delete element by index\n");

printf("4. print sum of elements\n");

printf("5. print average\n");

printf("6. print minimum and maximum\n");

printf("7. print array\n");

printf("8. change number of elements in array\n");

printf("9. exit\n");

printf("choose an operation: ");

scanf("%d", &choice);

switch (choice) { //оператор switch для исполнения различных функций в зависимости от значения, введённого пользователем

case 1:

if (da != NULL) {

free\_dynamic\_array(da);

}

size\_t initial\_size;

printf("enter the number of elements: ");

scanf("%zu", &initial\_size);

da = create\_dynamic\_array(initial\_size);

printf("choose how to fill the array:\n");

printf("1. fill with random numbers\n");

printf("2. fill with zeros\n");

int fill\_choice;

scanf("%d", &fill\_choice);

switch (fill\_choice) {

case 1:

resize\_array(da, initial\_size);

break;

case 2:

fill\_array\_with\_zeros(da, initial\_size);

break;

default:

printf("incorrect choice. filling with zeros by default.\n");

fill\_array\_with\_zeros(da, initial\_size);

break;

}

first\_choice = 0;

break;

case 2:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

int value;

printf("enter the element for addition: ");

scanf("%d", &value);

append(da, value);

break;

case 3:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

size\_t index;

printf("enter the index for deletion: ");

scanf("%zu", &index);

deletion(da, index);

break;

case 4:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

printf("sum of elements: %d\n", sum(da));

break;

case 5:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

printf("average: %.2f\n", average(da));

break;

case 6:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

int min, max;

find\_min\_max(da, &min, &max);

printf("minimum: %d\n", min);

printf("maximum: %d\n", max);

break;

case 7:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

print\_array(da);

break;

case 8:

if (first\_choice) {

printf("incorrect choice.\n");

break;

}

if (da == NULL) {

printf("error: first set the array size (select 1)\n");

break;

}

size\_t new\_size;

printf("enter the new number of elements: ");

scanf("%zu", &new\_size);

resize\_array(da, new\_size);

break;

case 9:

if (da != NULL) {

free\_dynamic\_array(da);

}

return 0;

default:

if (first\_choice) {

printf("incorrect choice.\n");

} else {

printf("incorrect choice.\n");

}

}

}

return 0;

}

Ссылка на GitHub:

ВЫВОД:

В ходе выполнения курсовой работы была реализована программа на языке Си для работы с динамическим массивом. Программа предоставляет пользователю интерфейс для выбора операций и выполняет соответствующие функции. Все операции были успешно протестированы, что подтверждает корректность реализации. Реализация динамического массива на языке Си позволяет эффективно управлять памятью и выполнять различные операции с массивом. Программа предоставляет пользователю удобный интерфейс для выбора операций и выполняет соответствующие функции, что делает её полезной для различных задач, связанных с обработкой данных.

ИСТОЧНИКИ

1. Грег Перри, Дин Миллер “Программирование на С для начинающих”, 3 издание.
2. Брайан Керниган, Деннис Ритчи “Язык программирования C”, 3 издание.