|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Практическая работа № 2 | | |
| по дисциплине «Информационные технологии и основы программирования» | | |
| **структура данных: стеки и очереди** | | |
|  | | |
|  | Бригада №3 | Ерощенко Артем, Кауфман Яна, Ковалева МАрия |
| Группа ПМИ-33 |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватель | Дворецкая виктория константиновна |
|  |  |
| Новосибирск,2024 | | |

**1.** **Цель работы:** сформировать практические навыки организации таких распространенных структур, как стеки и очереди, и их использования при решении задач.

**2. Задание:** Для указанных ниже представлений очереди, опишите на Си соответствующий тип *очередь*(считая, что все элементы имеют некоторый простой тип) и реализуйте в виде процедуры или функции перечисленные ниже операции над очередью (если операция по тем или иным причинам не может быть выполнена, подпрограмма должна возвратить 1 – в случае переполнения очереди, 2 – в случае исчерпания очереди).

в) Линейный массив с двумя указателями (на начало и на конец очереди). Если указатель на конец очереди достиг конца массива, а в начале очереди есть свободные позиции (очередь псевдополна), то конец очереди перенести в ее начало (т.е. массив как бы склеивается в кольцо – циклическая очередь);

конец начало

0 1 N-1

э3 э4••• э1 э2

N-2

1 22222

beginend

г) Линейным односвязным списком с двумя указателями (на начало и на конец очереди)

началоконец

NULL

. . .

beg

end

Для указанных ниже представлений стека, опишите на Си соответствующий тип *стек*(считая, что все элементы имеют некоторый простой тип) и реализуйте в виде процедуры или функции перечисленные ниже операции над стеком (если операция по тем или иным причинам не может быть выполнена, подпрограмма должна возвратить 1 – в случае переполнения стека, 2 – в случае исчерпания стека).

б) Линейным односвязным списком с указателем на вершину стека (см. рис.2.2.);

NULL

●●●

вершина стека

**3. Анализ задачи:**

**1(в)**

**Входные данные:** переменные типа int

**Выходные данные:** элементы очереди

**Метод решения:**

**1(г)**

**Входные данные:** переменные типа int

**Выходные данные:** элементы очереди

**Метод решения:** Программа реализует линейный односвязный список с двумя указателями (на начало и на конец очереди): структура “Node” элемента списка, которая содержит в себе переменную типа int, а так же указатель на следующий элемент списка; подпрограмма “ initQueue”, которая создает нулевую очередь или очищает уже существующую; подпрограмма “ isEmpty” проверяет очередь на пустоту; подпрограмма “isFull” проверяет очередь на переполненность; подпрограмма “printQueue” выводит все элементы очереди в консоль; подпрограмма “enqueue” добавляет введенный элемент в очередь; подпрограмма “dequeue” удаляет элемент из очереди.

**2(б)**

**Входные данные:**

**Выходные данные:**

**Метод решения:**

**4. Текст программы с пояснениями:**

**1(в)**

#include <iostream>

struct Queue {

int\* data; // Указатель на массив элементов очереди

int N, size; // Размер массива

int beg, end; // Индексы начала и конца очереди

Queue() : data(nullptr), N(0), beg(-1), end(-1) {}

bool isEmpty() {

return beg == -1;

}

bool isFull() {

return (beg == 0 && end == N - 1) || (beg == end + 1);

}

void enqueue(int value) {

if (isFull()) {

std::cout << "The queue is full!\n";//Очередь переполнена!

return;

}

if (beg == -1) {

beg = 0;

end = 0;

}

else {

if (end == N - 1) {

end = 0;

}

else {

end++;

}

}

data[end] = value;

}

int dequeue() {

if (isEmpty()) {

std::cout << "The queue is empty!\n";//Очередь пуста!

return -1;

}

int value = data[beg];

if (beg == end) {

beg = end = -1;

}

else {

if (beg == N - 1) {

beg = 0;

}

else {

beg++;

}

}

return value;

}

void printQueue() {

if (isEmpty()) {

std::cout << "The queue is empty!\n";//Очередь пуста!

}

else {

std::cout << "Items in the queue:";//Элементы в очереди:

if (end < beg)

for (int i = 0; i < N; i++) {

std::cout << " " << data[i];

if (i == N - 1) break;

}

else for (int i = beg; i < N; i++) {

std::cout << " " << data[i];

if (i == end) break;

}

std::cout << std::endl;

}

}

};

int main() {

Queue queue;

int size, choice, value;

std::cout << "Enter the queue size:";//Введите размер очереди:

std::cin >> size;

queue.data = new int[size];

queue.N = size;

std::cout << "Enter the queue items (enter -1 to end):\n";//Введите элементы очереди (для окончания введите -1):

do {

std::cin >> value;

if (value != -1) {

queue.enqueue(value);

}

} while (value != -1);

do {

std::cout << "Select an operation:" << std::endl;//Выберите операцию:

std::cout << "1. Create an empty queue" << std::endl;//1. Создать пустую очередь

std::cout << "2. Check if the queue is empty" << std::endl;//2. Проверить, является ли очередь пустой

std::cout << "3. Add an item to the queue" << std::endl;//3. Добавить элемент в очередь

std::cout << "4. Take an item from the queue" << std::endl;//4. Взять элемент из очереди

std::cout << "0. Exit" << std::endl;//0. Выйти

std::cin >> choice;

switch (choice) {

case 1:

delete[] queue.data;

queue.data = nullptr;

queue.N = 0;

queue.beg = -1;

queue.end = -1;

std::cout << "The queue has been successfully cleared\n";//Очередь успешно очищена

break;

case 2:

queue.printQueue();

break;

case 3:

std::cout << "Enter the value to add:";//Введите значение для добавления:

std::cin >> value;

queue.enqueue(value);

break;

case 4:

std::cout << "The item was taken from the queue:" << queue.dequeue() << std::endl;//Взят элемент из очереди:

break;

case 0:

std::cout << "Exiting the program\n";//Выход из программы

break;

default:

std::cout << "Wrong choice\n";//Неверный выбор

}

} while (choice != 0);

return 0;

}

**1(г)**

#include <iostream>

struct Node { //структура ячейки очереди

int data;

Node\* next;

};

struct Queue { //структура очереди

Node\* front;

Node\* rear;

int size;

int capacity;

};

void initQueue(Queue& q, int capacity) { //создание очереди нулевой длины/очистка очереди

q.front = nullptr;

q.rear = nullptr;

q.size = 0;

q.capacity = capacity;

}

bool isEmpty(Queue q) { //проверка очереди на пустоту

return q.front == nullptr;

}

bool isFull(Queue q) { //проверка очереди на переполненость

return q.size == q.capacity;

}

void printQueue(Queue q) { //вывод очереди

if (isEmpty(q)) {

std::cout << "Queue is empty.\n";

return;

}

std::cout << "Queue: ";

Node\* current = q.front;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << "\n";

}

int enqueue(Queue& q, int value) { //добавление элемента в очередь

if (isFull(q)) {

return 1; //очередь переполнена

}

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (isEmpty(q)) {

q.front = newNode;

} else {

q.rear->next = newNode;

}

q.rear = newNode;

q.size++;

printQueue(q);

return 0;

}

int dequeue(Queue& q, int& value) { //удаление элемента из очереди

if (isEmpty(q)) {

return 2; //очередь пуста

}

Node\* temp = q.front;

value = temp->data;

q.front = q.front->next;

delete temp;

q.size--;

if (isEmpty(q)) {

q.rear = nullptr;

}

printQueue(q);

return 0;

}

int main() {

Queue q;

int capacity, choice, element, result, exit;

bool end = true;

std::cout << "Enter queue size: ";

std::cin >> capacity;

initQueue(q, capacity);

while (end){

std::cout << "To clear the queue, enter 1.\n" << //чтобы очистить очередь, введите 1

"To check whether the queue is empty, press 2.\n" << //чтобы проверить является ли очередь пустой, введите 2

"To add an item to the queue, press 3.\n" << //чтобы добавить элемент в очередь, нажмите 3

"To remove an item from the queue, enter 4. \n"; //чтобы удалить элемент из очереди, введите 4

std::cin >> choice;

if(choice == 1){

initQueue(q, capacity);

std::cout << "Queue is empty!\n"; //очередь пуста

}else if (choice == 2){

if (isEmpty(q)) {

std::cout << "Queue is empty!\n"; //очередь пуста

} else {

std::cout << "Queue is not empty.\n"; //очередь не пуста

}

}else if (choice == 3){

std::cout << "Enter element for add: "; //введите элемент для добавления

std::cin >> element;

result = enqueue(q, element);

if (result == 1) {

std::cout << "Queue is full!\n"; //очередь переполнена

} else {

std::cout << "Element in queue.\n"; //элемент добавлен в очередь

}

}else if (choice == 4){

int extractedElement;

result = dequeue(q, extractedElement);

if (result == 2) {

std::cout << "Queue is empty!\n"; //очередь пуста

} else {

std::cout << "Takes element: " << extractedElement << "\n"; //элемент для удаления

}

}else{

std::cout << "Incorect value.\n"; //некорректное значение

end = false;

}

std::cout << "Enter 1 if you want to continue working with the queue and 0 if you don't want to. \n"; //введите 1, если хотите продолжить работу с очередью, и 0, если хотите закончить

std::cin >> exit;

if (exit == 0) end = false;

}

return 0;

}

**2(б)**

#include <iostream>

namespace S {

typedef struct Node {

int data;

Node \*next;

} Node;

typedef struct stack {

int max\_e, n;

Node \*top;

} stack;

stack\* create(int max\_e){ // Create stack and fill it with initial values

stack \*q = new stack;

q->max\_e = max\_e;

q->n = 0;

q->top = NULL;

return q;

};

char free(stack \*q){

Node \*buff;

while(q->n > 0){ // Consistenly free all nodes

buff = q->top;

q->top = q->top->next;

q->n --;

delete buff;

}

return 0;

};

char is\_empty(stack \*q){

return q->n == 0;

}

char insert(stack \*q, int data){ // Insert new node with data at the top of the stack

if(q->n + 1 > q->max\_e) return 1; // Check whether the stack is full

Node \*new\_top = new Node;

new\_top->next = q->top;

new\_top->data = data;

q->top = new\_top;

q->n ++;

return 0;

}

char get\_top(stack \*q, Node &out){ // Return top element and remove it from stack

if(is\_empty(q)) return 2; // Check whether the stack is empty

out = \*(q->top);

Node \*buff = q->top;

q->top = q->top->next;

q->n --;

delete buff;

return 0;

}

void print\_stack(stack \*q){ // Consistent print of the stack

Node \*buff = q->top;

std::cout << "\n top: ";

for(int i = 0; i < q->n; i++){

std::cout << buff->data << " -> ";

buff = q->top->next;

}

std::cout << "NULL";

}

};

int main(){

S::stack\* stack = S::create(5);

int mode, val;

S::Node buff;

// Gain user input -> mode and execute that mode

std::cout << "Enter the mode(1 - insert, 2 - get top, 3 - exit): ";

while(scanf("%d", &mode) > 0 && mode != 3){

if(mode == 1){

std::cout << "Enter the data of a node: ";

scanf("%d", &val);

if(S::insert(stack, val) == 1) std::cout << "Overflow!!!\n";

} else if(mode == 2){

if(S::get\_top(stack, buff) == 2){

std::cout << "Stack is empty!\n";

} else {

std::cout << "top value: " << buff.data << '\n';

}

}

S::print\_stack(stack);

std::cout << "\nEnter the mode(1 - insert, 2 - get top, 3 - exit): ";

}

S::free(stack);

return 0;

}

**5. Тестирование программы:**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Примечание |
|  |  |  |  |