ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| СТЕК И ОЧЕРЕДЬ |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Иванов И.В. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Цель работы**: Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

**Задание на лабораторную работу:**

Реализовать структуры данных «стек» и «очередь» в соответствии с заданным вариантом. Дополнительно программа должна удовлетворять следующим требованиям:

1) Вывод на экран состояния моделируемой системы на каждой итерации работы (содержимое стека(ов), очереди(ей), процессора(ов));

2) Для каждой задачи из списка входных задач должно быть определено время поступления;

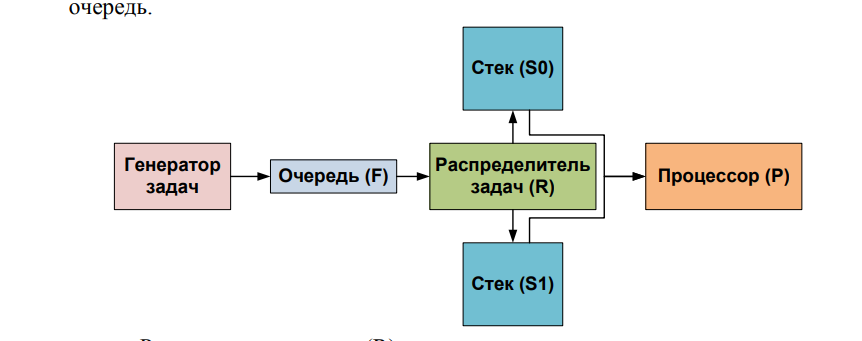
3) Необходимо наличие, как автоматического генератора задач, так и возможность ручного добавления задач, с указанием их параметров (в зависимости от задания);

4) Необходимо обработать ситуации, при которых какая-либо структура данных может быть переполнена.

**Ход работы:**

1. В соответствии с формулой рассчитываем свой вариант





Задача 8.

1. Система состоит из процессора P, двух стеков S0 и S1, очереди F и распределителя задач R. Поступающие в систему запросы, попадают в очередь. Распределитель задач (R), получает задачу из очереди и помещает ее либо в стек S0, либо в стек S1 (зависимости от приоритета задачи). Процессор P, обрабатывает задачи из стеков в порядке приоритета. Таким образом, если стек S0 пуст и процессор P свободен, то могут быть обработаны задачи из стека S
2. Разработать алгоритм и запрограммировать его

**Листинг программы**

**#include <iostream>**

**using namespace std;**

**int timer = 1;**

**const int s = 6;**

**struct Task {**

**int priority;**

**int Tasktime;**

**int Timeincoming;**

**Task\* next;**

**Task\* prev;**

**Task(int priority1, int Tasktime1, int Timeincoming1) : priority(priority1), Tasktime(Tasktime1), Timeincoming(Timeincoming1), next(nullptr), prev(nullptr) {}**

**Task() : priority(0), Tasktime(0), Timeincoming(0), next(nullptr), prev(nullptr) {}**

**Task(const Task& T) : priority(T.priority), Tasktime(T.Tasktime), Timeincoming(T.Timeincoming), next(nullptr), prev(nullptr) {}**

**};**

**struct Stack {**

**Task\* Top;**

**int s\_size = 0;**

**Stack() : Top(nullptr), s\_size(0) {}**

**bool s\_empty() {**

**return (Top == nullptr);**

**}**

**void s\_push1(int prior, int Taskti, int Timeinc) {**

**Task\* T = new Task(prior, Taskti, Timeinc);**

**if (s\_empty()) {**

**Top = T;**

**}**

**else {**

**T->next = Top;**

**Top = T;**

**}**

**s\_size++;**

**}**

**void s\_push(Task Task1) {**

**Task\* T = new Task(Task1);**

**if (s\_empty()) {**

**Top = T;**

**}**

**else {**

**T->next = Top;**

**Top = T;**

**}**

**s\_size++;**

**}**

**void s\_delete() {**

**if (s\_empty()) {**

**cout << "Стек пуст!" << endl;**

**return;**

**}**

**else {**

**if (s\_size == 1) {**

**delete Top;**

**Top = nullptr;**

**s\_size = s\_size - 1;**

**}**

**else if (s\_size > 1) {**

**Task\* T = Top;**

**Top = Top->next;**

**delete T;**

**s\_size = s\_size - 1;**

**}**

**}**

**}**

**void s\_info() {**

**Task\* T = Top;**

**if (!s\_empty()) {**

**cout << "Содержимое стека " << Top->priority << endl;**

**for (int i = 0; i < s\_size; i++) {**

**cout << "Время поступления " << T->Timeincoming << " Приоритет задачи " << T->priority << " такты задачи " << T->Tasktime << endl;**

**T = T->next;**

**}**

**}**

**}**

**};**

**struct Queue {**

**Task\* Front;**

**Task\* Back;**

**int size = 6;**

**Queue() :Front(nullptr), Back(nullptr), size(0) {}**

**bool q\_empty() {**

**return (Front == nullptr);**

**}**

**void q\_push(Task Task1) {**

**Task\* T = new Task(Task1);**

**if (q\_empty()) {**

**Front = T;**

**Back = T;**

**size++;**

**}**

**else if (size == 1) {**

**Front->next = T;**

**T->prev = Front;**

**Back = T;**

**size++;**

**}**

**else if (size > 1) {**

**Back->next = T;**

**T->prev = Back;**

**Back = T;**

**size++;**

**//Task\* Ta = Front->next;**

**//while (Ta->next != nullptr) {**

**// Ta = Ta->next;**

**//}**

**//Ta->next = T;**

**//T->prev = Ta;**

**//Back = T;**

**//size++;**

**}**

**}**

**void q\_delete() {**

**if (q\_empty()) {**

**cout << "Очередь пуста!" << endl;**

**return;**

**}**

**else if (size > 1) {**

**Task\* T = Front->next;**

**T->prev = nullptr;**

**delete Front;**

**Front = T;**

**size--;**

**}**

**else if (size == 1) {**

**delete Front;**

**Front = nullptr;**

**Back = nullptr;**

**size--;**

**}**

**//cout << "Delete" << endl;**

**}**

**void distributor(Stack S0, Stack S1) {**

**if (Front->priority == 2) {**

**S0.s\_push1(Front->priority, Front->Tasktime, Front->Timeincoming);**

**q\_delete();**

**}**

**else if (Front->priority == 1) {**

**S1.s\_push1(Front->priority, Front->Tasktime, Front->Timeincoming);**

**q\_delete();**

**}**

**}**

**void info() {**

**Task\* T = Front;**

**if (!q\_empty()) {**

**cout << "Содержимое очереди " << endl;**

**for (int i = 0; i < size; i++) {**

**cout << "Время поступления " << T->Timeincoming << " Приоритет задачи " << T->priority << " такты задачи " << T->Tasktime << endl;**

**T = T->next;**

**}**

**}**

**}**

**};**

**struct Processor {**

**Task\* Task1;**

**Processor() : Task1(nullptr) {}**

**bool ProcessorIsFree() {**

**return Task1 == nullptr;**

**}**

**int getTask(Stack S0, Stack S1) {**

**if (S0.s\_empty()) {**

**if (S1.s\_empty()) {**

**cout << "Все задачи выполнены!" << endl;**

**return 0;**

**}**

**else {**

**Task\* T = new Task(\*S1.Top);**

**Task1 = T;**

**//S1.s\_delete();**

**return 2;**

**}**

**}**

**else {**

**Task\* T = new Task(\*S0.Top);**

**Task1 = T;**

**//S0.s\_delete();**

**return 1;**

**}**

**}**

**void Processing() {**

**Task1->Tasktime--;**

**if (Task1->Tasktime == 0) {**

**delete Task1;**

**Task1 = nullptr;**

**}**

**}**

**void Procinfo() {**

**if (!ProcessorIsFree()) {**

**cout << "Содержимое процессора" << endl;**

**cout << "Время поступления " << Task1->Timeincoming << " Приоритет задачи " << Task1->priority << " такты задачи " << Task1->Tasktime << endl;**

**}**

**}**

**};**

**void TaskInQueue(Task Task1, Task Que[]) {**

**for (int i = 0; i < s; i++) {**

**if (Que[i].Tasktime == 0) {**

**Que[i] = Task1;**

**break;**

**}**

**}**

**}**

**void TaskInfo(Task T) {**

**cout << "Время поступления " << T.Timeincoming << " Приоритет задачи " << T.priority << " такты задачи " << T.Tasktime << endl;**

**}**

**void Info(Task IncomingTask[], Queue Q1, Stack S0, Stack S1, Processor Proc) {**

**cout << "Идет " << timer << " такт" << endl;**

**timer++;**

**if (!(IncomingTask[0].Tasktime == 0)) {**

**cout << "Входные задания" << endl;**

**for (int i = 0; i < s; i++) {**

**if (!(IncomingTask[i].Tasktime == 0)) {**

**TaskInfo(IncomingTask[i]);**

**}**

**}**

**}**

**Q1.info();**

**//if (!(Queue[0].Tasktime == 0)) {**

**// cout << "Содержимое очереди" << endl;**

**// for (int i = 0; i < s; i++) {**

**// if (!(Queue[i].Tasktime == 0)) {**

**// TaskInfo(Queue[i]);**

**// }**

**// }**

**//}**

**S0.s\_info();**

**S1.s\_info();**

**Proc.Procinfo();**

**cout << endl;**

**}**

**bool AllTaskGone(Task IncomingTask[], Queue Q1, Stack S0, Stack S1, Processor Proc) {**

**if (IncomingTask[1].Tasktime == 0) {**

**if (Q1.q\_empty() == 1) {**

**if (S0.s\_empty()) {**

**if (S1.s\_empty()) {**

**if (Proc.ProcessorIsFree()) {**

**return 1;**

**}**

**}**

**}**

**}**

**}**

**return 0;**

**}**

**void Incdel(Task IncomingTask[]) {**

**Task T;**

**for (int i = 0; i < s; i++) {**

**IncomingTask[i] = IncomingTask[i + 1];**

**if (i == 5) {**

**IncomingTask[i] = T;**

**}**

**}**

**}**

**int main()**

**{**

**setlocale(LC\_ALL, "RU");**

**Stack S0, S1;**

**Queue Q1;**

**Processor Proc;**

**Task IncomingTask[s];**

**//Task Queue[s];**

**Task Task1(1, 2, 1), Task2(2, 1, 2), Task3(1, 2, 3), Task4(2, 3, 4), Task5(2, 1, 5), Task6(1, 2, 6), Task7(2, 7, 7);**

**//Task Task1(2, 4, 1), Task2(2, 3, 2), Task3(1, 3, 3), Task4(2, 2, 4), Task5(1, 1, 5), Task6(1, 4, 6), Task7(2, 7, 7);**

**TaskInQueue(Task1, IncomingTask);**

**TaskInQueue(Task2, IncomingTask);**

**TaskInQueue(Task3, IncomingTask);**

**TaskInQueue(Task4, IncomingTask);**

**TaskInQueue(Task5, IncomingTask);**

**TaskInQueue(Task6, IncomingTask);**

**TaskInQueue(Task7, IncomingTask);**

**//TaskInQueue(Task1, Queue, s);**

**//Distributor(S0, S1, Queue);**

**//Proc.Processing(S0, S1);**

**//TaskInQueue(Task2, Queue, s);**

**//TaskInQueue(Task3, Queue, s);**

**//TaskInQueue(Task4, Queue, s);**

**//TaskInQueue(Task5, Queue, s);**

**//TaskInQueue(Task6, Queue, s);**

**//TaskInQueue(Task7, Queue, s);**

**Task TA;**

**int i = 0, GT = 0;**

**while (1) {**

**if (!AllTaskGone(IncomingTask, Q1, S0, S1, Proc)) {**

**//TaskInQueue(IncomingTask[0], Queue);**

**if (!(IncomingTask[0].Tasktime == 0)) {**

**Q1.q\_push(IncomingTask[0]);**

**Incdel(IncomingTask);**

**}**

**i++;**

**//Distributor(S0, S1, Queue);**

**//Q1.distributor(S0, S1);**

**if (!(Q1.q\_empty())) {**

**if (Q1.Front->priority == 1) {**

**S0.s\_push1(Q1.Front->priority, Q1.Front->Tasktime, Q1.Front->Timeincoming);**

**Q1.q\_delete();**

**}**

**else if (Q1.Front->priority == 2) {**

**S1.s\_push1(Q1.Front->priority, Q1.Front->Tasktime, Q1.Front->Timeincoming);**

**Q1.q\_delete();**

**//S1.s\_push(Queue[0]);**

**//for (int i = 0; i < s; i++) {**

**// Queue[i] = Queue[i + 1];**

**// if (i == 5) {**

**// Queue[i] = TA;**

**// }**

**//}**

**}**

**}**

**if (!Proc.ProcessorIsFree()) {**

**Proc.Processing();**

**}**

**if (Proc.ProcessorIsFree()) {**

**GT = Proc.getTask(S0, S1);**

**if (GT == 1) {**

**S0.s\_delete();**

**}**

**else if (GT == 2) {**

**S1.s\_delete();**

**}**

**else if (GT == 0) {**

**break;**

**}**

**//GT = Proc.getTask(S0, S1);**

**//switch (GT) {**

**//case 1:**

**// S0.s\_delete();**

**//case 2:**

**// S1.s\_delete();**

**//case 0:**

**// break;**

**//}**

**}**

**Info(IncomingTask, Q1, S0, S1, Proc);**

**}**

**}**

**/\* int P, T;**

**cout << "ОЧередь: " << endl;**

**Queue Q1;**

**for (int i = 0; i < 3; i++) {**

**cout << "Prior: ";**

**cin >> P;**

**cout << "Time: ";**

**cin >> T;**

**Q1.q\_push(P, T);**

**}**

**Q1.q\_delete();**

**Q1.q\_delete();**

**Q1.q\_delete();**

**cout << "стек: " << endl;**

**Stack s1;**

**for (int i = 0; i < 3; i++) {**

**cout << "Prior: ";**

**cin >> P;**

**cout << "Time: ";**

**cin >> T;**

**s1.s\_push(P, T);**

**}**

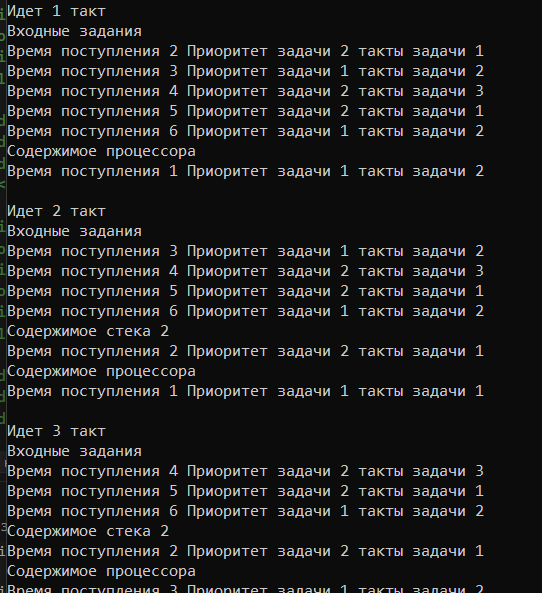
**s1.s\_delete();**

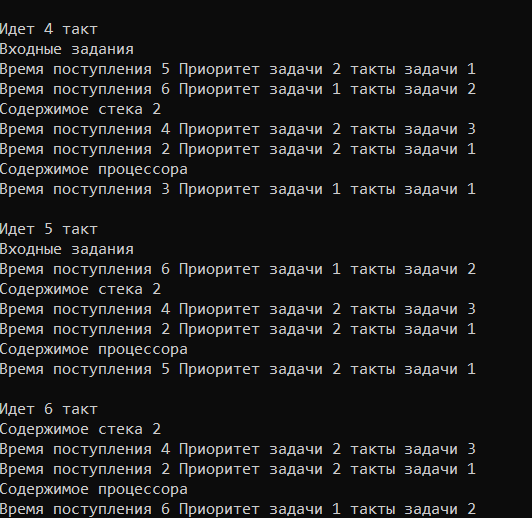
**s1.s\_delete();**

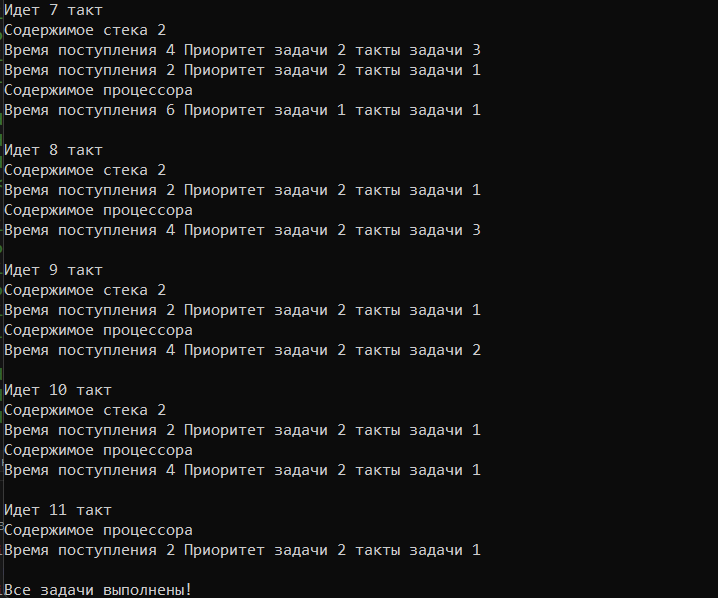
**s1.s\_delete();\*/**

**}**

**Работа программы:**

****

****

****

**Вывод:**

Я изучил структуры данных стек и очередь, а так же получил практические навыки их реализации.