ГУАП

# КАФЕДРА №43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  | 12.10.2023 |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3 |
| СТЕК И ОЧЕРЕДЬ |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232 |  | 12.10.2023 |  | М.В.Куриш |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Цель работы:**

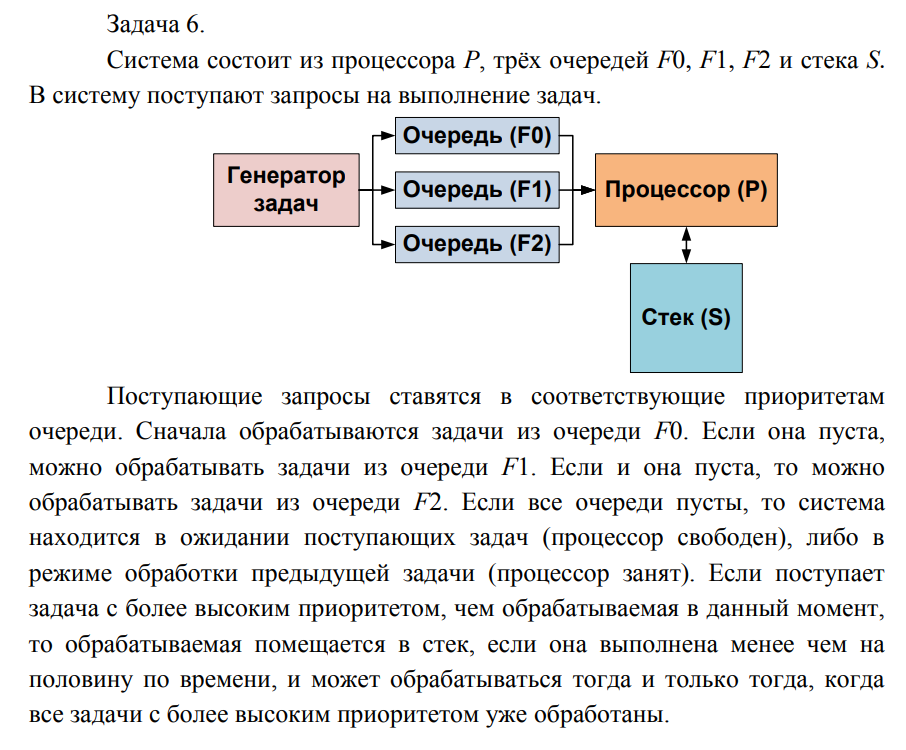
Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

1. **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. Варианты задания приведены в таблице 3 (формулировки задач приведены после таблицы).

Вариант 22.

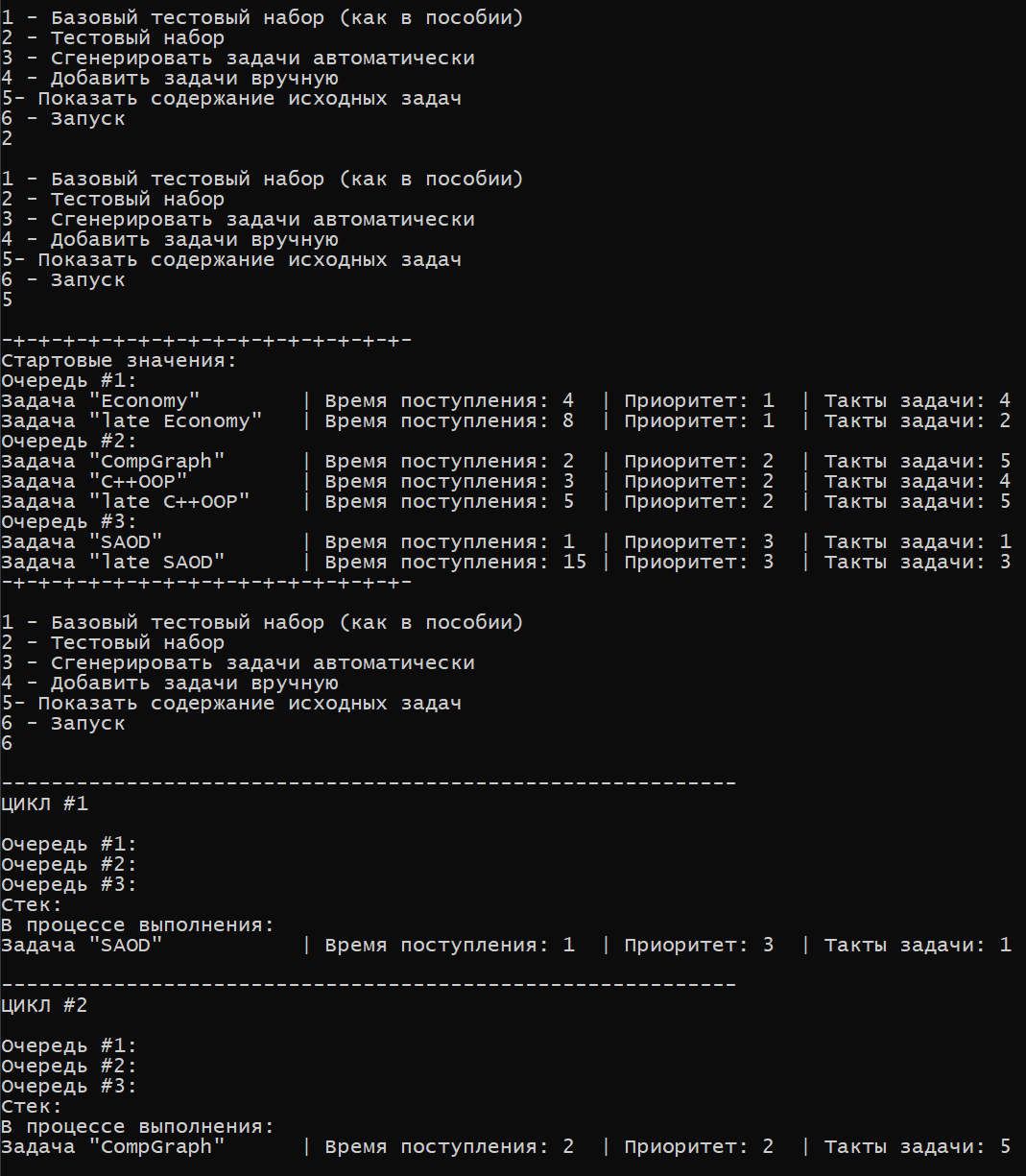


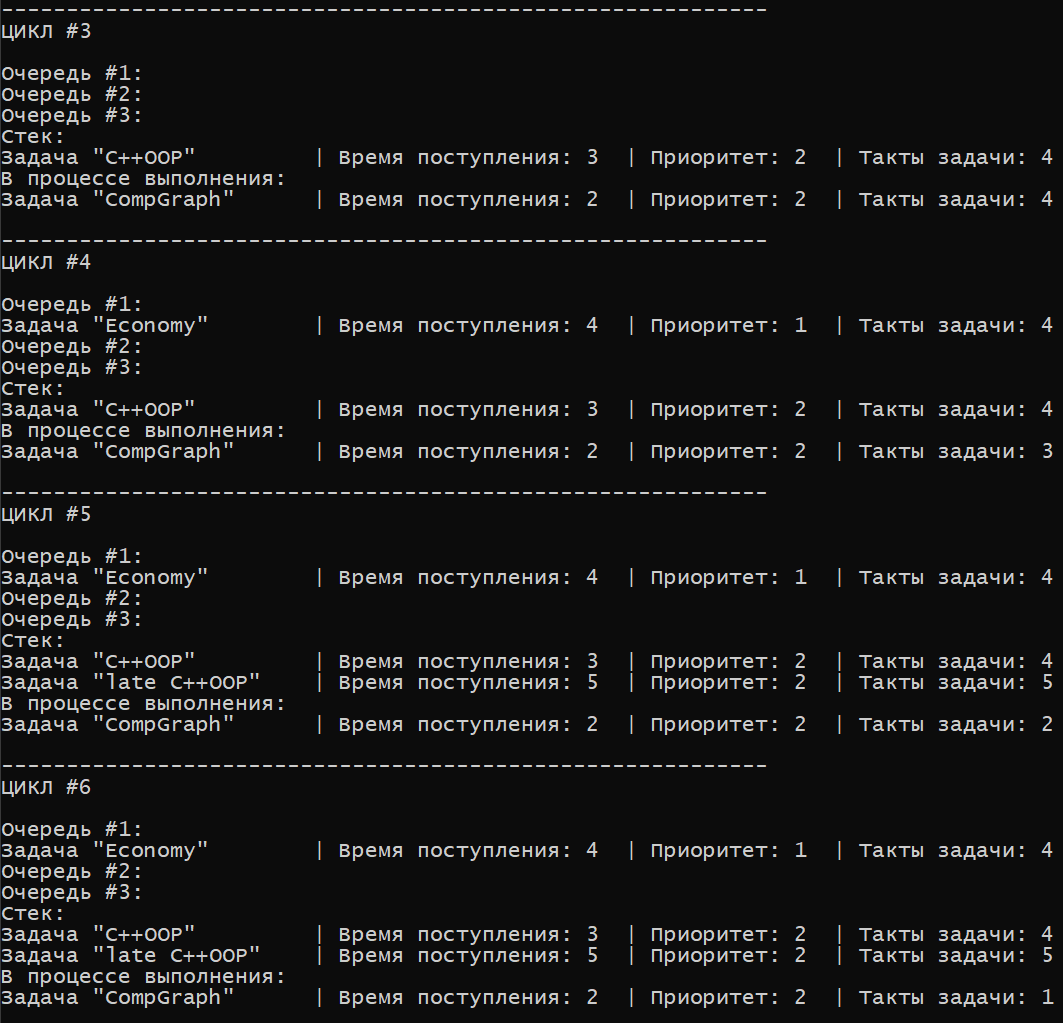


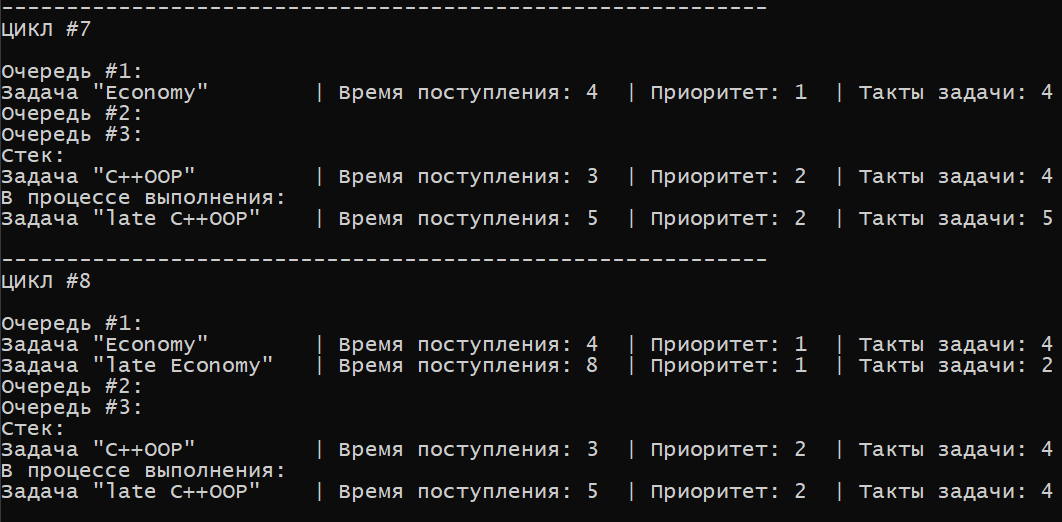
1. **Листинг программы:**

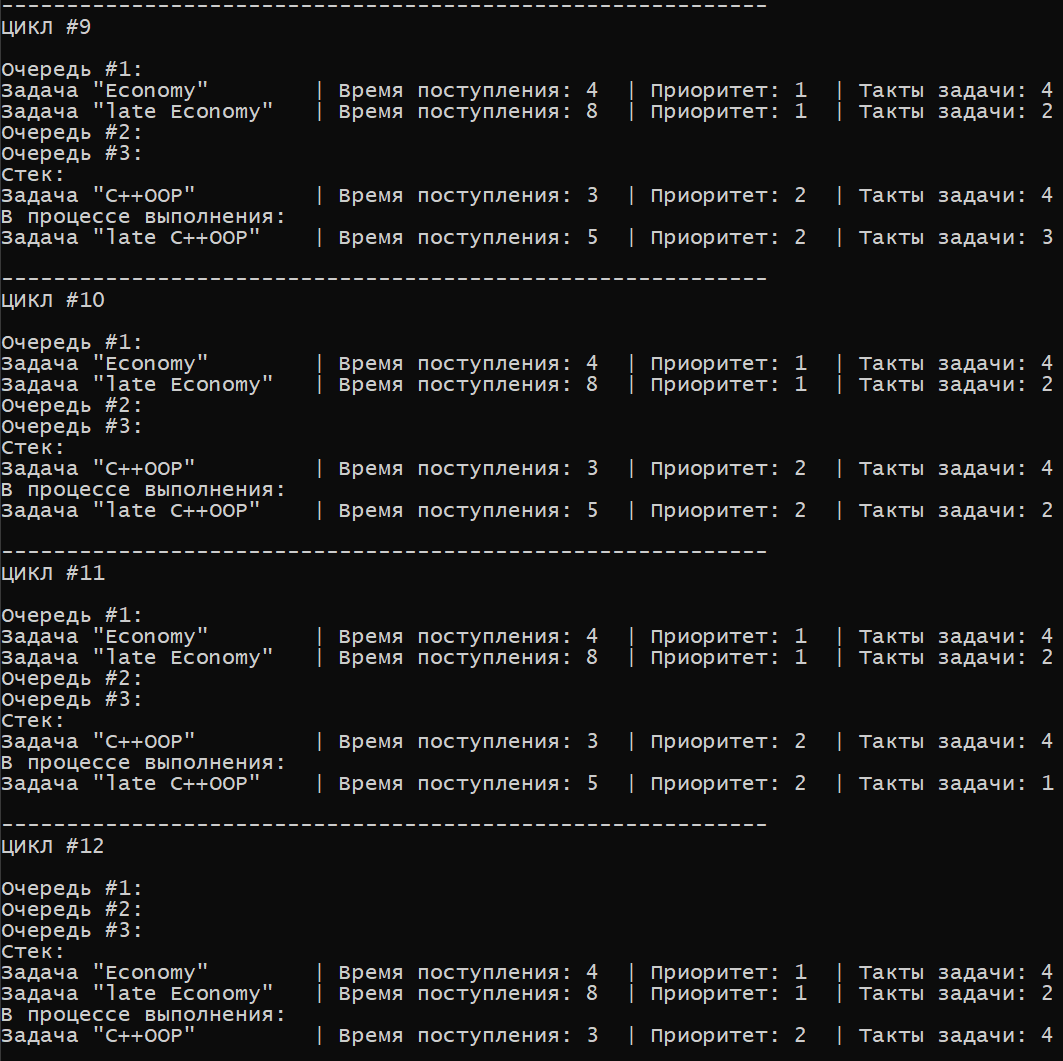
|  |
| --- |
| #include <iostream>  #include <atomic>  #include <cstdarg>  #include <stdexcept>  #include <random>  #include <thread>  #include <chrono>  using namespace std;  int intInput() {  int a;  cin >> a;  while (!cin) {  cout << "Неверный ввод. (Ожидается число)\n";  cin.clear();  cin.ignore(999, '\n');  cout << "Введите заново: ";  cin >> a;  }  return a;  }  class Task {  public:  unsigned short priority = 65535;  unsigned short taskTime = 65535;  unsigned short durationTime = 65535;  unsigned short restDuration = 65535;  string name = "NULL\_NAME";  Task() {}  Task(unsigned short priority,  unsigned short taskTime,  unsigned short durationTime,  string name = "NULL\_NAME") {  this->priority = priority;  this->taskTime = taskTime;  this->durationTime = durationTime;  this->restDuration = durationTime;  this->name = name;  }  void showTask() {  cout << "Задача \"" << name << "\"";  if (name.size() < 7)  cout << "\t";  cout << "\t| Время поступления: " << taskTime << "\t| Приоритет: " << priority << "\t| Такты задачи: " << restDuration << endl;  }  }; |
| class TasksStack {  private:  unsigned short topInd = 0;  Task innerStack[100];  public:  Task peek()  {  if (!isEmpty())  return innerStack[topInd - 1];  throw runtime\_error("Попытка чтения из пустого стека");  }  Task pop()  {  if (!isEmpty())  return innerStack[topInd-- - 1];  throw runtime\_error("Попытка чтения из пустого стека");  }  void push(Task t)  {  if (topInd < 100)  innerStack[topInd++] = t;  else  cout << "Стек переполнен!\n";  }  void clear()  {  while (!isEmpty())  pop();  }  bool isEmpty()  {  return topInd == 0;  }  void show()  {  cout << "Стек:\n";  for (unsigned short i = 0; i < topInd; i++) {  innerStack[i].showTask();  }  }  };  class TasksQueue {  private:  class QueueNode {  public:  Task task;  QueueNode\* next;  QueueNode(Task& task) {  this->task = task;  this->next = nullptr;  }  };  QueueNode\* head;  QueueNode\* tail;  public:  TasksQueue() {  head = nullptr;  tail = nullptr;  }  Task peek()  {  if (!isEmpty()) {  return tail->task;  }  throw runtime\_error("Попытка чтения из пустой очереди");  }  unsigned short head\_peek\_taskTime()  {  if (head != nullptr) {  return head->task.taskTime;  }  else {  cout << 1 << endl;  return 1;  }  }  Task pop()  {  if (!isEmpty()) {  QueueNode\* tempNode = tail;  tail = tail->next;  Task tempTask = tempNode->task;  delete tempNode;  if (tail == nullptr)  head = nullptr;  return tempTask;  }  throw runtime\_error("Попытка чтения из пустой очереди");  }  void push(Task t)  {  if (isEmpty()) {  head = new QueueNode(t);  tail = head;  }  else {  head->next = new QueueNode(t);  head = head->next;  }  }  void clear()  {  while (!isEmpty()) {  pop();  }  }  bool isEmpty()  {  return tail == nullptr;  }  void show(int num)  {  cout << "Очередь #" << num << ":\n";  QueueNode\* curr = tail;  while (curr != nullptr) {  curr->task.showTask();  curr = curr->next;  }  }  }; |
| void processorLoop(int count...) {  TasksStack s;  Task\* inProcess = nullptr;  bool allPreviousQueuesEmpty = true;  va\_list vlst;  va\_start(vlst, count);  unsigned short\* timer = va\_arg(vlst, unsigned short\*);  bool\* zeroIncomingTasks = va\_arg(vlst, bool\*);  atomic<TasksQueue>\*\* queueList = new atomic<TasksQueue>\*[count];  for (int i = 0; i < count; i++)  queueList[i] = va\_arg(vlst, atomic<TasksQueue>\*);  va\_end(vlst);  TasksQueue tempQueue;  while (true) {  cout << "\n-----------------------------------------------------------\n";  cout << "ЦИКЛ #" << \*timer << "\n\n";  allPreviousQueuesEmpty = true;  for (int i = count - 1; i >= 0; i--) {  if (queueList[i]->load().isEmpty() || !allPreviousQueuesEmpty)  continue;  while (queueList[i]->load().peek().taskTime <= \*timer) {  if (inProcess == nullptr) {  if (!s.isEmpty()) {  if (s.peek().priority > queueList[i]->load().peek().priority) {  inProcess = new Task(s.pop());  if (!s.isEmpty()) {  if (s.peek().priority <= queueList[i]->load().peek().priority) {  tempQueue = queueList[i]->load();  s.push(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  }  else {  tempQueue = queueList[i]->load();  s.push(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  }  else {  tempQueue = queueList[i]->load();  inProcess = new Task(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  }  else {  tempQueue = queueList[i]->load();  inProcess = new Task(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  }  else {  if (queueList[i]->load().peek().priority > inProcess->priority && inProcess->restDuration > inProcess->durationTime / 2) {  s.push(\*inProcess);  tempQueue = queueList[i]->load();  \*inProcess = tempQueue.pop();  queueList[i]->exchange(tempQueue);  }  else {  if (!s.isEmpty()) {  if (s.peek().priority <= queueList[i]->load().peek().priority) {  tempQueue = queueList[i]->load();  s.push(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  else  break;  }  else {  tempQueue = queueList[i]->load();  s.push(tempQueue.pop());  queueList[i]->exchange(tempQueue);  }  }  }  if (queueList[i]->load().isEmpty()) break;  }  if (!queueList[i]->load().isEmpty()) {  allPreviousQueuesEmpty = false;  }  }  if (inProcess == nullptr && !s.isEmpty())  inProcess = new Task(s.pop());  if (!(inProcess == nullptr && allPreviousQueuesEmpty && s.isEmpty())) {  for (int i = 0; i < count; i++) {  queueList[i]->load().show(i + 1);  }  s.show();  cout << "В процессе выполнения:\n";  if (inProcess != nullptr)  inProcess->showTask();  if (inProcess != nullptr) {  inProcess->restDuration--;  if (inProcess->restDuration == 0) {  delete inProcess;  inProcess = nullptr;  }  }  }  if (inProcess == nullptr && allPreviousQueuesEmpty && s.isEmpty() && \*zeroIncomingTasks) {  break;  }  (\*timer)++;  this\_thread::sleep\_for(chrono::milliseconds(500));  }  } |
| Task taskGenerator(int queuesCount) {  random\_device rd;  mt19937 rng(rd());  uniform\_int\_distribution<int> uniPriority(1, queuesCount);  uniform\_int\_distribution<int> uniLetters(4, 9);  uniform\_int\_distribution<int> uniLetter(97, 122);  uniform\_int\_distribution<int> uniDuration(1, 5);  string tName = "";  for (unsigned short i = 0; i < uniLetters(rng); i++)  tName += (char)uniLetter(rng);  Task t;  t.priority = uniPriority(rng);  t.durationTime = uniDuration(rng);  t.name = tName;  t.taskTime = 0;  t.restDuration = t.durationTime;  return t;  }  Task setTaskManually(int queuesCount) {  Task t;  cout << "Название задания: ";  string tName;  cin >> tName;  cout << "Приоритет задания (от 1 до " << queuesCount << "): ";  unsigned short tPriority = intInput();  while (tPriority < 1 || tPriority > queuesCount) {  cout << "Приоритет задания Должен быть в диапазоне от 1 до " << queuesCount << "!\n";  cout << "Повторите ввод: ";  tPriority = intInput();  }  cout << "Длительность задания: ";  unsigned short tDuration = intInput();  cout << "Момент поступления задания: ";  unsigned short tTaskTime = intInput();  t.name = tName;  t.priority = tPriority;  t.durationTime = tDuration;  t.taskTime = tTaskTime;  t.restDuration = t.durationTime;  return t;  } |
| void automaticallyTaskAdding(int count...) {  va\_list vlst;  va\_start(vlst, count);  TasksQueue\*\* tqlst = new TasksQueue \* [count];  for (unsigned short i = 0; i < count; i++)  tqlst[i] = va\_arg(vlst, TasksQueue\*);  int tasksCount;  Task newTask;  cout << "Введите количество задач для автоматической генерации: ";  tasksCount = intInput();  int i = 1;  while (tasksCount-- > 0) {  newTask = taskGenerator(count);  newTask.taskTime = i++;  (\*(tqlst[newTask.priority - 1])).push(newTask);  }  va\_end(vlst);  }  void manuallyTaskAdding(int count...) {  va\_list vlst;  va\_start(vlst, count);  TasksQueue\*\* tqlst = new TasksQueue \* [count];  for (unsigned short i = 0; i < count; i++)  tqlst[i] = va\_arg(vlst, TasksQueue\*);  Task newTask;  bool flag = true;  while (flag) {  cout << "\n1 - Добавить задачу\n";  cout << "2 - Закончить ручное добавление задач\n";  switch (intInput()) {  case 1:  newTask = setTaskManually(count);  if ((\*(tqlst[newTask.priority - 1])).isEmpty())  (\*(tqlst[newTask.priority - 1])).push(newTask);  else {  while (newTask.taskTime < (\*(tqlst[newTask.priority - 1])).head\_peek\_taskTime()) {  cout << "Нельзя задать значение поступления задачи\nболее ранним чем последнее в очереди!\nСоздайте задание еще раз!\n";  newTask = setTaskManually(count);  }  (\*(tqlst[newTask.priority - 1])).push(newTask);  }  break;  case 2:  flag = false;  break;  default:  cout << "Неверно выбран пункт меню. Повторите попытку.\n";  break;  }  }  va\_end(vlst);  } |
| int main()  {  setlocale(LC\_ALL, "RUS");  atomic<TasksQueue> f0;  atomic<TasksQueue> f1;  atomic<TasksQueue> f2;  TasksQueue lateF0;  TasksQueue lateF1;  TasksQueue lateF2;  TasksQueue tempQueue;  unsigned short timer = 1;  bool zeroIncomingTasks = false;  bool flag = true;  while (flag) {  cout << "\n1 - Базовый тестовый набор (как в пособии)\n";  cout << "2 - Тестовый набор\n";  cout << "3 - Сгенерировать задачи автоматически\n";  cout << "4 - Добавить задачи вручную\n";  cout << "5- Показать содержание исходных задач\n";  cout << "6 - Запуск\n";  switch (intInput()) {  case 1:  lateF0.push(Task(3, 1, 2, "SAOD"));  lateF0.push(Task(3, 2, 1, "SAOD"));  lateF0.push(Task(1, 3, 2, "Economy"));  lateF0.push(Task(2, 4, 3, "C++OOP"));  lateF0.push(Task(3, 5, 3, "SAOD"));  break;  case 2:  lateF0.push(Task(1, 4, 4, "Economy"));  lateF1.push(Task(2, 2, 5, "CompGraph"));  lateF1.push(Task(2, 3, 4, "C++OOP"));  lateF2.push(Task(3, 1, 1, "SAOD"));  lateF0.push(Task(1, 8, 2, "late Economy"));  lateF1.push(Task(2, 5, 5, "late C++OOP"));  lateF2.push(Task(3, 15, 3, "late SAOD"));  break;  case 3:  automaticallyTaskAdding(3, ref(lateF0), ref(lateF1), ref(lateF2));  break;  case 4:  manuallyTaskAdding(3, ref(lateF0), ref(lateF1), ref(lateF2));  break;  case 5:  cout << "\n-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n"  << "Стартовые значения:\n";  lateF0.show(1);  lateF1.show(2);  lateF2.show(3);  cout << "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-\n";  break;  case 6:  flag = false;  break;  default:  cout << "Неверно выбран пункт меню. Повторите попытку.\n";  break;  }  }  thread processorThread(processorLoop, 3, ref(timer), ref(zeroIncomingTasks), ref(f0), ref(f1), ref(f2));  while (!lateF0.isEmpty() || !lateF1.isEmpty() || !lateF2.isEmpty()) {  if (!lateF2.isEmpty()) {  while (lateF2.peek().taskTime <= timer) {  tempQueue = f2.load();  tempQueue.push(lateF2.pop());  f2.exchange(tempQueue);  if (lateF2.isEmpty())  break;  }  }  if (!lateF1.isEmpty()) {  while (lateF1.peek().taskTime <= timer) {  tempQueue = f1.load();  tempQueue.push(lateF1.pop());  f1.exchange(tempQueue);  if (lateF1.isEmpty())  break;  }  }  if (!lateF0.isEmpty()) {  while (lateF0.peek().taskTime <= timer) {  tempQueue = f0.load();  tempQueue.push(lateF0.pop());  f0.exchange(tempQueue);  if (lateF0.isEmpty())  break;  }  }  this\_thread::sleep\_for(chrono::milliseconds(400));  }  zeroIncomingTasks = true;  processorThread.join();  } |

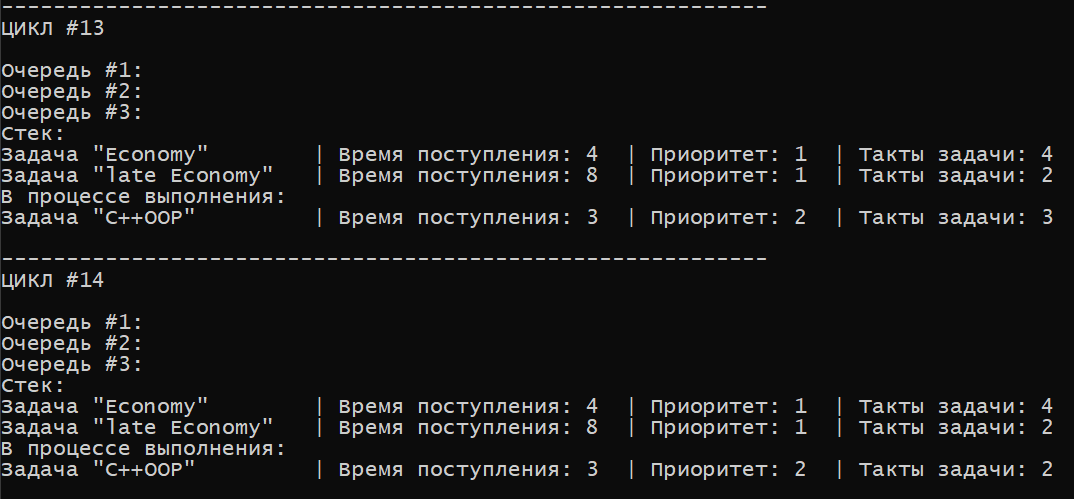
1. **Контрольные примеры:**

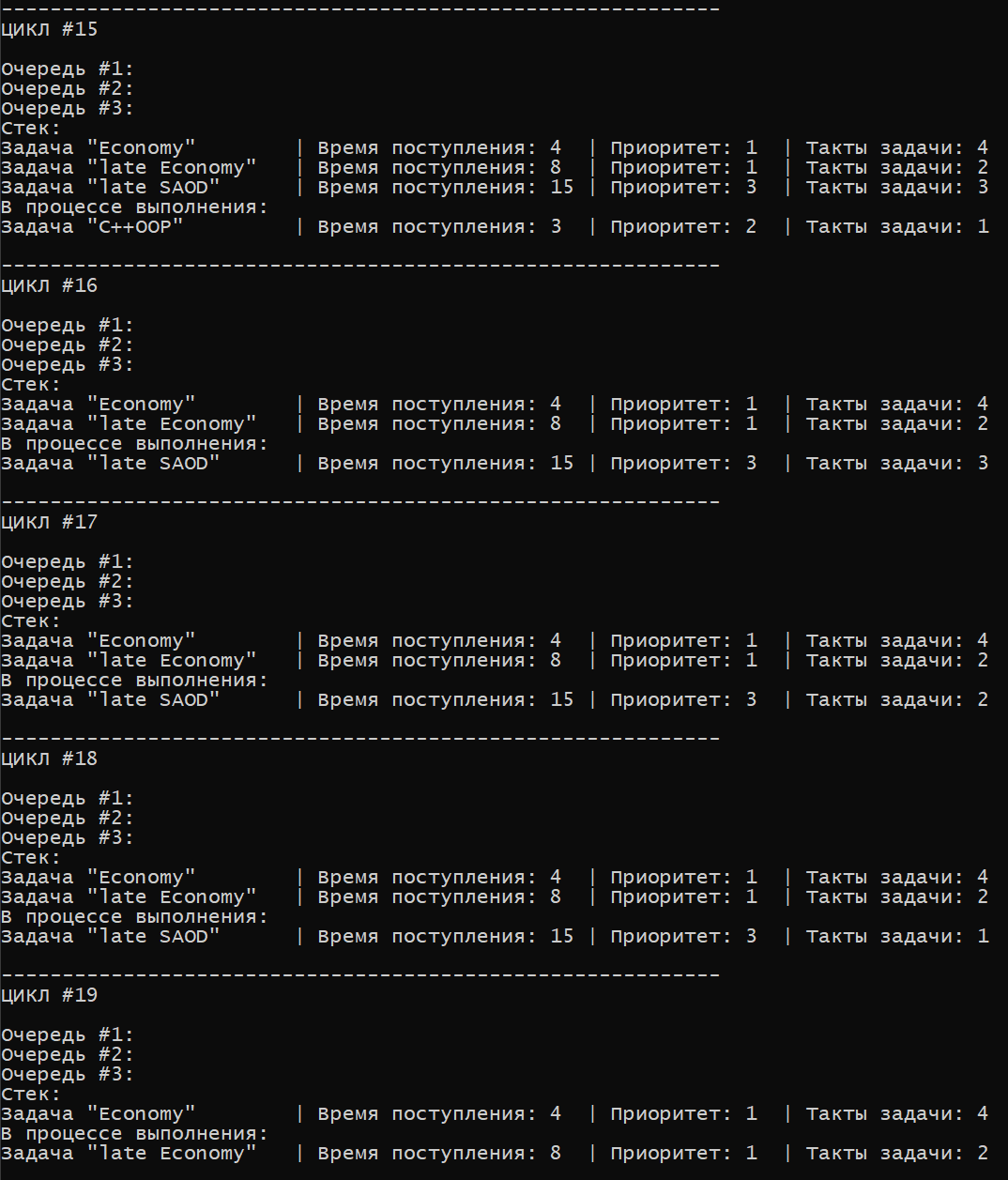


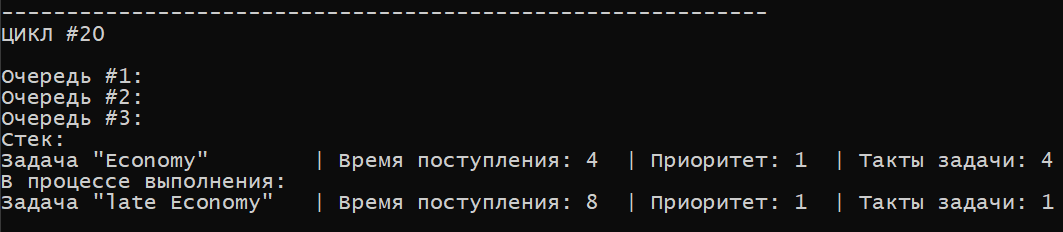


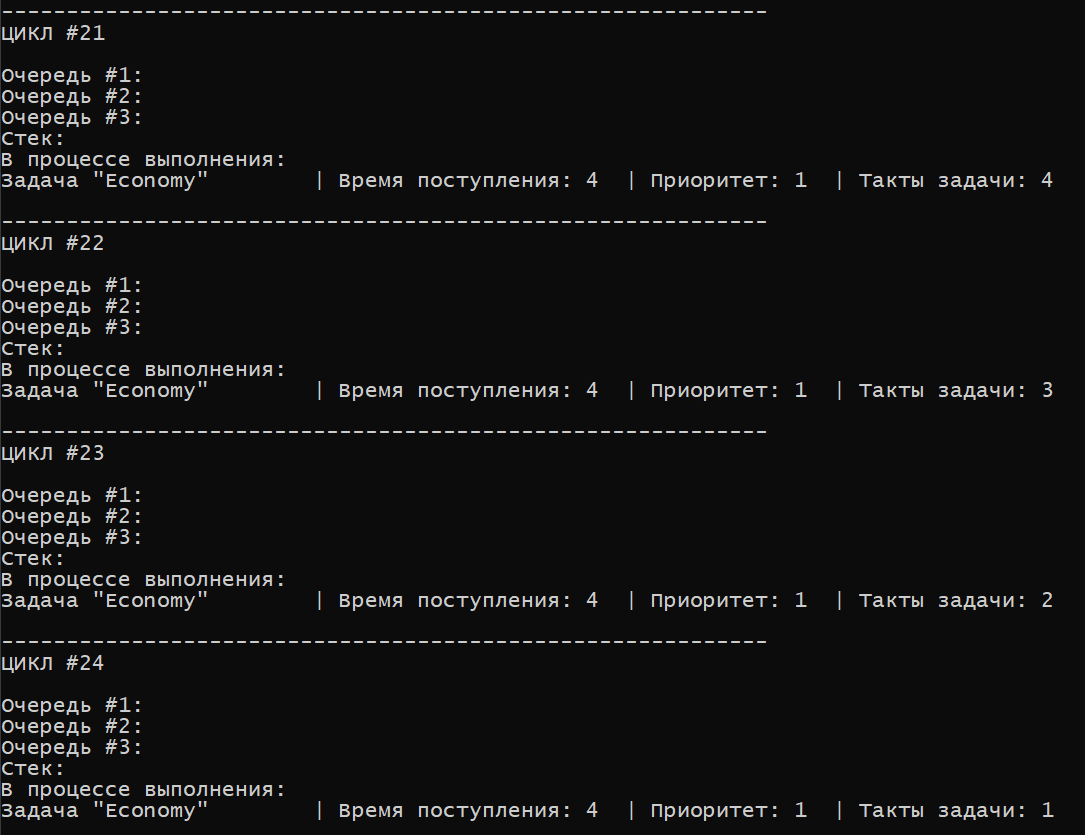












1. **Вывод:**

Изучены структуры данных «стек» и «очередь». Получен опыт их реализации.