**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе работе №6 на тему:   
**«Разработка программ моделирования работы стека и очереди»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-114,Константинов А.Д.

«\_\_» \_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_/А.Д. Константинов/

Принял: к.ф.-м.н., доцент, И.А. Моисеев

« » \_\_\_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание:**

[Цель работы: 3](#_Toc114692552)

[Листинг кода: 3](#_Toc114692553)

[Результат работы программы: 8](#_Toc114692554)

[Вывод: 8](#_Toc114692555)

[Приложение: 22](#_Toc114692556)

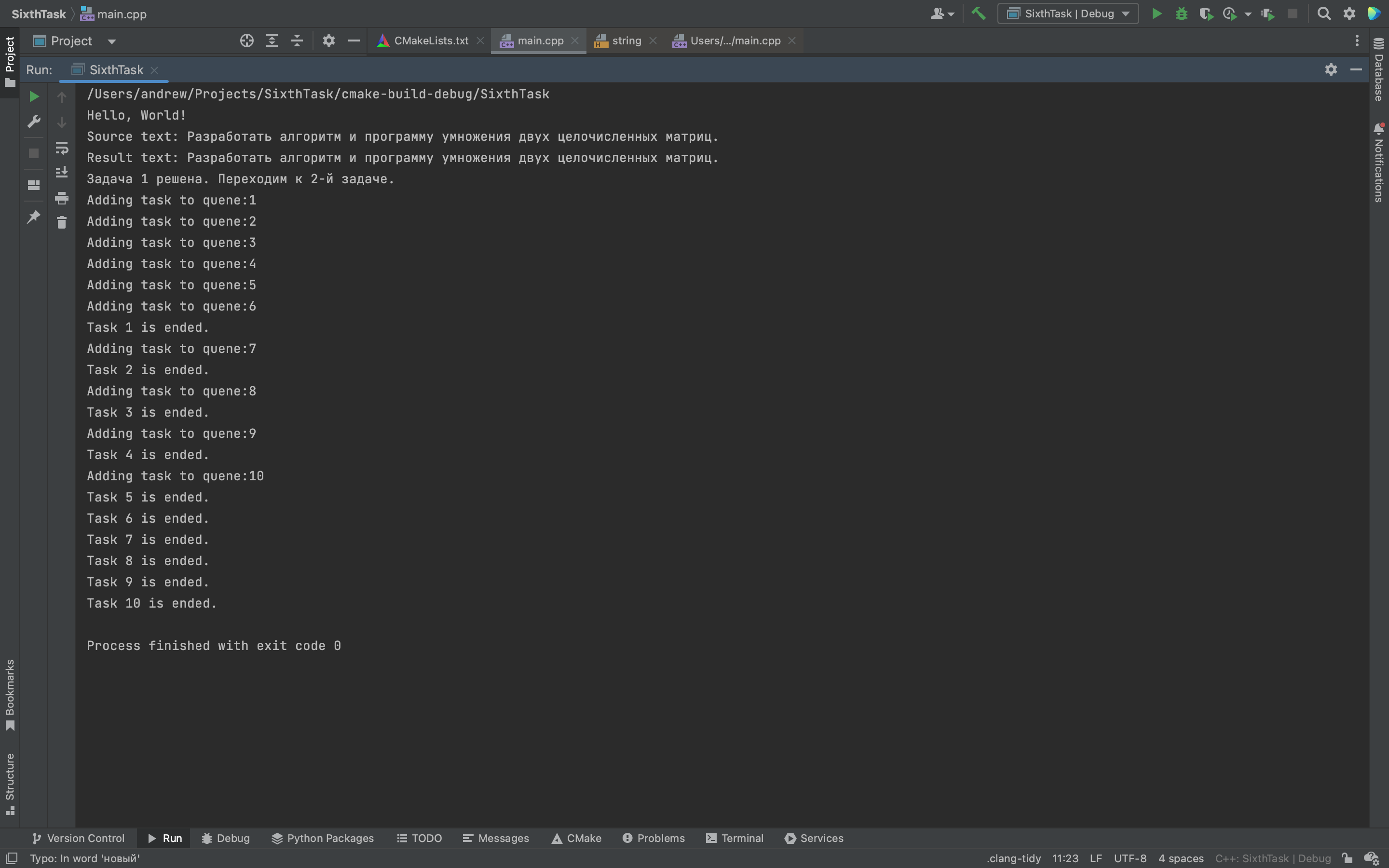
Цель работы:

# Разработка программ моделирования работы стека и очереди

# Листинг кода:

#include <iostream>  
//Stack == LIFO Last Input First Out  
struct Stack{  
 Stack\* next = nullptr;  
 std::string value = "";  
};  
void StackPush(Stack \*\* stack,std::string data)  
{  
 //Добавляем новый элемент в качестве головного  
 //Создаем новый элемент:  
 Stack\* newNode = new Stack;  
 newNode->value = data;  
 newNode->next = nullptr;  
 auto t = \*stack;  
 if(t->value != "")  
 newNode->next = t;  
 \*stack = newNode;  
}  
bool StackEmpty(Stack\* stack)  
{  
 if(stack == nullptr) return true;  
 return false;  
}  
void StackPop(Stack\*\* stack)  
{  
 //удаляем верхний элемент  
 if(StackEmpty(\*stack) == false) {  
 auto addr = (\*stack)->next;  
 \*stack = addr;  
 }  
}  
std::string StackPeek(Stack\* stack)  
{  
 return stack->value;  
}  
void Task1(){  
 auto sourceText = "Разработать алгоритм и программу умножения двух целочисленных матриц.";  
 std::cout << "Source text: " << sourceText << std::endl;  
 //Parse and adding words to stack  
  
 //In  
 Stack \* stack = new Stack;  
 std::string temp = "";  
 for (int i = strlen(sourceText)-1; i >= 0; --i) {  
 if(sourceText[i] == ' ')  
 {  
 std::reverse(temp.begin(), temp.end());  
 StackPush(&stack,temp);  
 temp = "";  
 }  
 temp+=sourceText[i];  
 }  
 std::reverse(temp.begin(), temp.end());  
 StackPush(&stack,temp);  
  
 //Out  
 std::cout << "Result text: ";  
 while(!StackEmpty(stack))  
 {  
 std::cout << StackPeek(stack);  
 StackPop(&stack);  
 }  
  
 std::cout << std::endl << "Задача 1 решена. Переходим к 2-й задаче." << std::endl;  
}  
  
  
  
//Quene == FIFO First In First Out  
struct Node{  
 Node\* next = nullptr;  
 std::string value = "";  
};  
struct Quene{  
 int size = 0;  
 Node\* first;  
 Node\* last;  
};  
void QueneCreate(Quene\* quene){  
 Node\* newNode = new Node; //Создаем ноду в памяти  
 newNode->next = nullptr;//У созданной ноды нет следующего элемента, так что next = nullptr  
 quene->first = newNode; // и сохраняем ссылку на  
 quene->last = newNode; // и сохраняем ссылку на  
 quene->size = 0; // Устанавливаем размер очереди  
}  
//  
void QuenePush(Quene\* quene, std::string data){  
 if(quene->size < 0) QueneCreate(quene);  
 Node\* newNode = new Node; //Создаем новый элемент  
 newNode->value = data; // Сохраняем в нем наши данные  
 newNode->next = nullptr; //Следующего элемента нет  
 quene->last->next = newNode; //В очереди у последнего элемента ссылка на следующий теперь - ссылка на новый жлемент, который толкьо что создали  
 quene->last = newNode; //А ссылкой на новый послежний элемент является ссылка на создланный элемент, поскольку он теперь последний  
 quene->size++; //Добавили элемент, значит размер тоже увеличился  
}  
//Удаляем верхний элемент  
void QuenePop(Quene\* quene){  
 if(quene->size == 0) {  
 quene->first->value ="";  
 return;  
 }  
 auto addr = quene->first;  
 quene->first = quene->first->next;  
 delete(addr);  
 quene->size--;  
}  
std::string QuenePeek(Quene\* quene)  
{  
 return quene->first->value;  
}  
bool QueneEmpty(Quene\* quene){  
 if(quene->first == quene->last) {  
 if(quene->first->value!= "") return false;  
 return true;  
 }  
 return false;  
}  
void Task2(){  
 Quene\* quene = new Quene;  
 QueneCreate(quene);  
 int currentTask = 0;  
 bool t = QueneEmpty(quene);  
 while(!QueneEmpty(quene) || currentTask < 10){  
 if(currentTask < 10)  
 {  
 std::cout << "Adding task to quene:" << (currentTask+1) << std::endl;  
 QuenePush(quene, std::to\_string(currentTask+1));  
 currentTask++;  
 }  
 if(currentTask > 4)  
 {  
 if(QuenePeek(quene) != "")  
 std::cout << "Task " << QuenePeek(quene) << " is ended." << std::endl;  
 QuenePop(quene);  
 }  
 }  
};  
int main() {  
 std::cout << "Hello, World!" << std::endl;  
 Task1();  
 Task2();  
 return 0;  
}

# Результат работы программы:



# Вывод:

В ходе выполнения практической работы была написана программа, работающая со стеком и очередью