**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №11

Вариант №11

Выполнил студент группы РИС-20-2б

Савельев Виталий Антонович

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с составными динамическими типами данных и получить практические навыки работы с такими типами данных, как:

1. Однонаправленные списки;
2. Двунаправленные списки;
3. Стеки;
4. Очереди;

**Постановка задачи**

Задача – реализовать на языке C++ программу, которая:

1. Формирует каждый тип данных: однонаправленный и двунаправленный список, стек, очередь с типами информационного поля, который указан в варианте;
2. Выводит на экран полученные данные;
3. Выполняет обработку данных (по отдельности с каждым типом);
4. Выводит обработанные данные;

Исходные данные для **варианта №11**:

|  |  |  |  |
| --- | --- | --- | --- |
| Однонаправленный список | Двунаправленный список | Стек | Очередь |
| Тип информационного поля: int.  Удалить из списка первый элемент с чётным инф. номером. | Тип информационного поля: char\*.  Добавить в список элемент с заданным номером. | Тип информационного поля: int.  Удалить из стека первый элемент с чётным инф. номером. | Тип информационного поля: char\*.  Добавить в очередь элемент с заданным номером. |

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Для последующей работы будут использоваться:

* Заголовочный файл iostream для операций ввода-вывода
* Заголовочный файл list и iterator для организации однонаправленных и двунаправленных списков, используя библиотеку STL.
* Заголовочный файл stack для организации стеков, используя библиотеку STL.
* Заголовочный файл queue для организации очередей, используя библиотеку STL.
* Пространство имён std для более удобного доступа к содержимому библиотеки STL.

Чтобы подключать заголовочные файлы, используется директива #include.

#include <iostream>

#include <list>

#include <iterator>

#include <stack>

#include <queue>

Также, для удобства, были добавлены кодовые слова, которые содержат сообщения для пользователя, которые будут выводиться на экран. Для этого используется директива #define.

Для однонаправленных списков:

#define LENGTH\_ERROR "Error: the list couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter list's count of new list:"

Для двунаправленных списков:

#define LENGTH\_ERROR "Error: the list couldn't be created, incorrect length value."

#define NEW\_INPUT\_MESSAGE "Enter queue's index where you'll add new word:"

#define INPUT\_MESSAGE "Enter list's count of new list:"

Для стеков:

#define LENGTH\_ERROR "Error: the stack couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new stack:"

Для очередей:

#define LENGTH\_ERROR "Error: the queue couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new queue:"

#define NEW\_INPUT\_MESSAGE "Enter queue's index where you'll add new word:"

Для более удобного ввода чисел, была реализована функция ReadNaturalNum, которая возвращает натуральное число (если введено отрицательное число, то функция попросит новый ввод).

Функция возвращает именно натуральные числа потому, что длинны массивов, их индексы являются натуральными числами (1, 2, 3, 4, 5, …).

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

**11.1: Однонаправленные списки:**

Однонаправленный список – наиболее простая динамическая структура, название которой говорит само за себя. Его элементы – это структуры, которые содержат два поля:

1. Значения текущего элемента – информационное поле;
2. Указатель на следующий элемент – адрес следующего объекта;

**Реализация при помощи структур:**

В языке программирования **C++**: такие объекты объявляются следующим образом:

struct List

{

int item;

List\* nextItem;

};

Для того, чтобы проинициализировать новый список, реализована функция NewRandomList, которая (исходя из названия) инициализирует новый список, добавляя в элементы случайные числа при помощи функции rand(). В качестве аргумента функция принимает число, которое отвечает за длину – количество элементов нового списка.

В начале выполнения идёт проверка на введённое число, если оно натуральное (> 0), то начинается формирование нового списка; в противном случае выводится сообщение об ошибке.

List\* NewRandomList(int listLength)

{

if (listLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return 0;

}

else

{

//формирование нового списка

}

}

Далее объявляются 2 списка:

* Список, который накапливается и будет возвращён в конце выполнения функции;
* Рабочий список, в который будет происходить запись всех элементов;

Тут же будет записан первый элемент списка, затем, в цикле, будет происходить запись остальных элементов и переход на следующие элементы.

//формирование нового списка

List

\*firstItem,

\*workList;

workList = new List;

workList->item = rand();

firstItem = workList;

for (int i = 1; i < listLength; i++)

{

List\* h = new List;

workList->nextItem = h;

delete h;

workList = workList->nextItem;

workList->item = rand();

workList->nextItem = NULL;

}

И, наконец, функция возвращает значения полученного списка:

return firstItem;

Для того, чтобы выводить список на экран, была реализована функция PrintList, которая выводит каждый элемент списка (аргумент функции) через пробел. Чтобы не пропустить не один элемент или не перескочить через них, на каждой итерации идёт проверка на наличие элемента (если указатель не равен NULL, то элемент существует и к нему можно обратиться).

void PrintList(List\* list)

{

List\* workItem = list;

while (workItem != NULL)

{

cout << workItem->item << ' ';

workItem = workItem->nextItem;

}

cout << endl;

}

Для реализации удаления первого элемента с чётным номером, была создана функция DeleteFirstOddItem. Функция всё также проходит по каждому элементу, но когда находит чётный элемент (остаток от деления на 2 равен 0), то происходит удаление текущего элемента и сдвиг указателя на следующий.

Поскольку требуется удалить **только первый чётный элемент**, то после удаления, функция выходит из цикла досрочно при помощи ключевого слова break.

void DeleteFirstOddItem(List\* list)

{

List\* p = list;

while (p->nextItem != NULL)

{

if (p->nextItem->item % 2 == 0)

{

List\* q = p->nextItem;

p->nextItem = p->nextItem->nextItem;

delete q;

break;

}

else

p = p->nextItem;

}

}

Реализация необходимых функций готова, можно приступать к написанию основной функции main.

В методе main происходят следующие действия:

1. Вывод сообщения с просьбой ввода количества элементов списка;
2. Ввод натурального числа с клавиатуры;
3. Инициализация нового списка:
4. Вывод списка на экран;
5. Обработка списка;
6. Вывод обработанного списка на экран;

Вывод сообщения и ввод числа с клавиатуры:

cout << INPUT\_MESSAGE << endl;

int itemsCount = ReadNaturalNum();

Инициализация и вывод нового списка:

List\* items = NewRandomList(itemsCount);

PrintList(items);

Обработка списка и его новый вывод:

DeleteFirstOddItem(items);

PrintList(items);

Завершение выполнения программы:

delete[] items;

return 0;

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#define LENGTH\_ERROR "Error: the list couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new list:"

using namespace std;

struct List

{

int item;

List\* nextItem;

};

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

List\* NewRandomList(int listLength)

{

if (listLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return 0;

}

else

{

List

\*firstItem,

\*workList;

workList = new List;

workList->item = rand();

firstItem = workList;

for (int i = 1; i < listLength; i++)

{

List\* h = new List;

workList->nextItem = h;

delete h;

workList = workList->nextItem;

workList->item = rand();

workList->nextItem = NULL;

}

return firstItem;

}

}

void PrintList(List\* list)

{

List\* workItem = list;

while (workItem != NULL)

{

cout << workItem->item << ' ';

workItem = workItem->nextItem;

}

cout << endl;

}

void DeleteFirstOddItem(List\* list)

{

List\* p = list;

while (p->nextItem != NULL)

{

if (p->nextItem->item % 2 == 0)

{

List\* q = p->nextItem;

p->nextItem = p->nextItem->nextItem;

delete q;

break;

}

else

p = p->nextItem;

}

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemsCount = ReadNaturalNum();

List\* items = NewRandomList(itemsCount);

PrintList(items);

DeleteFirstOddItem(items);

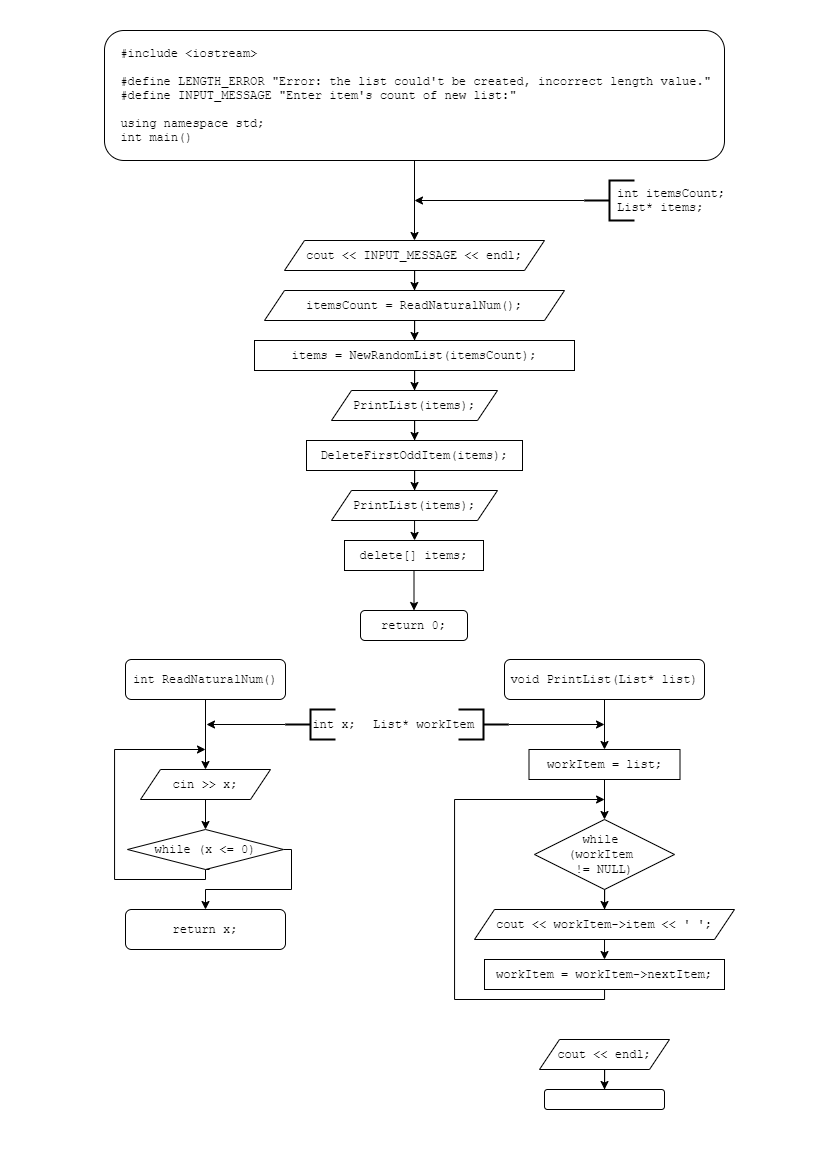
PrintList(items);

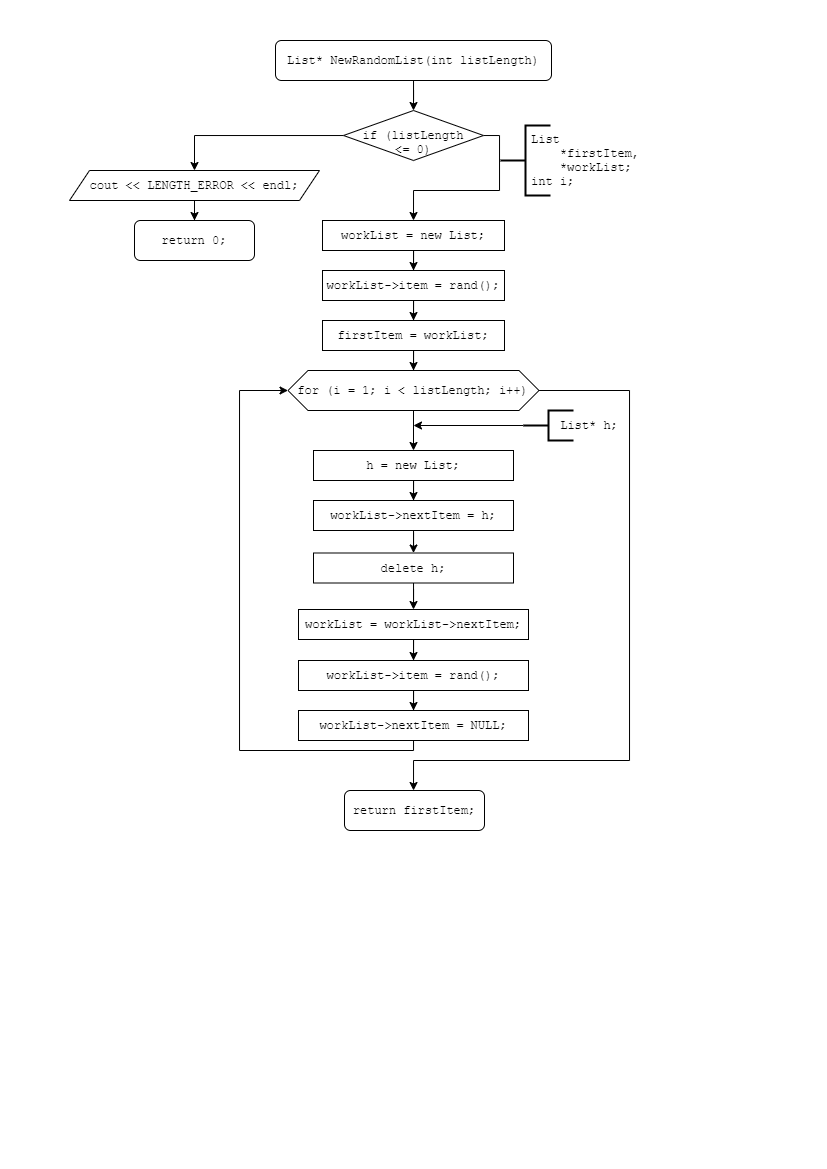
delete[] items;

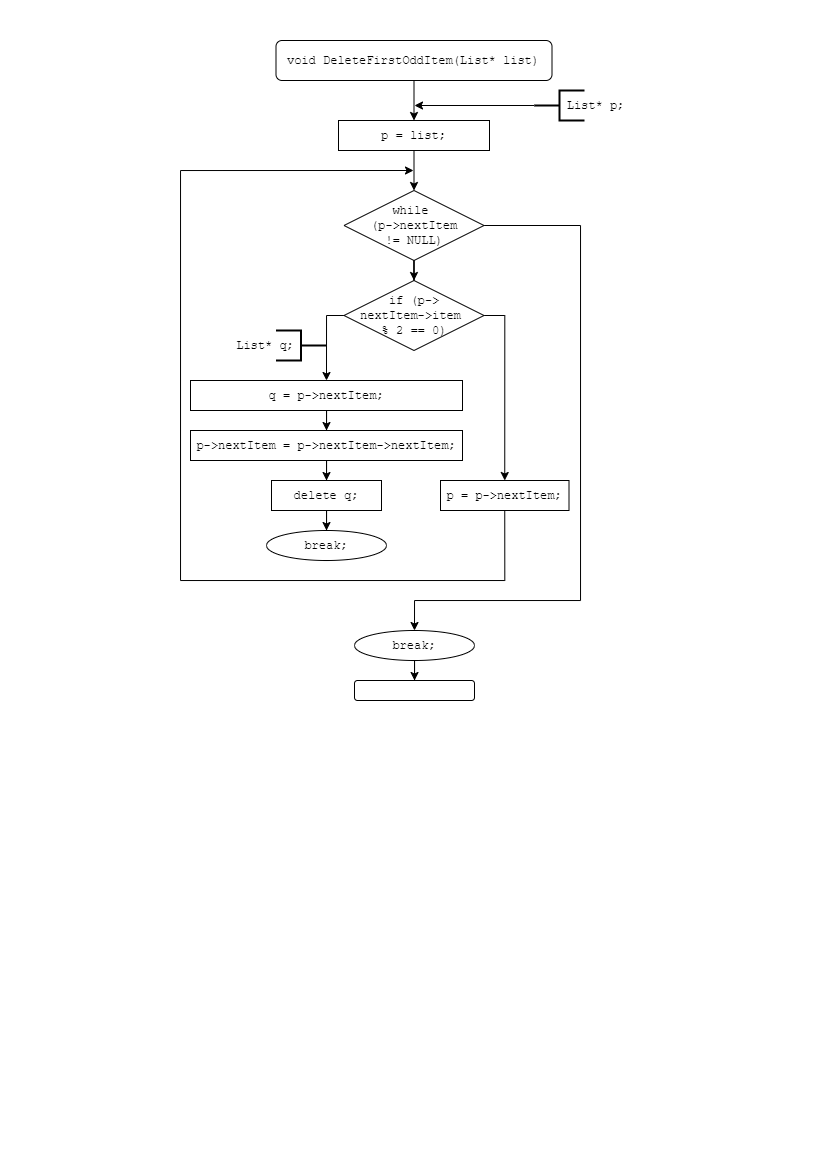
return 0;

}

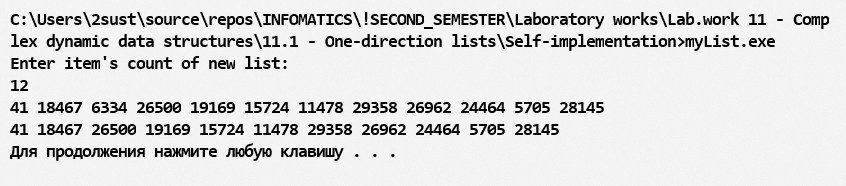
**Блок-схема** алгоритма программы:







**Скриншот** выполненной программы:



**11.2: Двунаправленные списки:**

Двунаправленные списки – это тот же однонаправленный список, но с одним отличием: элемента такого списка хранит не только само информационное поле и указатель на следующий элемент, но и содержит в себе указатель на предыдущий элемент. Поэтому он и называется двунаправленным (2 указателя: один указывает на предыдущий элемент, другой – на следующий элемент).

**Реализация при помощи структур:**

В языке программирования **C++**: такие объекты объявляются следующим образом:

struct List2

{

Char\* item;

List2\* nextItem;

List2\* prevItem;

};

Для удобства, список, с которым будет происходить работа, объявлена как глобальная переменная head.

List2\* head;

Для того, чтобы заполнить новый список элементами, реализованы 2 функции:

* InitializeNewList – общий метод, заполняющий весь список элементами;
* Add – метод, добавляющий 1 элемент в заданную позицию;

Реализация общего метода проста: принимается аргумент – длинна списка, и по циклу заполняется список (для простоты, список автоматически заполняется словами hello). Если длинна массива указана ненатуральным числом, выводится сообщение об ошибке.

void InitializeNewList(int length)

{

if (length <= 0)

cout << LENGTH\_ERROR << endl;

else

for (int i = 1; i <= length; i++)

Add("hello", i);

}

void Add(char\* value, int position) {}

Непосредственно, за добавление элементов в список отвечает метод Add, который в качестве аргументов принимает значение, которое нужно записать в список, и номер элемента, в который нужно записать данные (если под указанным номером уже есть элемент, то происходит вставка).

В начале идёт проверка на существование списка (есть ли в нём какие-нибудь элементы); если элементов нет, то идёт запись в первый элемент списка.

List2 \*node = new List2;

node->item = value;

if (head == NULL)

{

node->nextItem = node;

node->prevItem = node;

head = node;

}

else

{

//вставка или добавление нового элемента

}

Далее происходит непосредственное добавление/вставка нового элемента в список.

* Происходит сдвиг с конца на текущей позиции нового элемента всех последующих элементов вправо.
* В освободившейся ячейке списка записывается новый элемент

//вставка или добавление нового элемента

List2\* p = head;

for(int i = position; i > 1; i--)

p = p->nextItem;

p->prevItem->nextItem = node;

node->prevItem = p->prevItem;

node->nextItem = p;

p->prevItem = node;

Для того, чтобы выводить элементы списка, реализована функция PrintList, которая выводит каждый элемент списка через пробел.

void PrintList()

{

List2\* printList = head;

do

{

cout<< printList->item << ' ';

printList = printList->nextItem;

}

while(printList != head);

cout << endl;

}

Реализация необходимых функций готова: можно приступать к написанию основного метода main.

Вывод сообщения и ввод числа с клавиатуры:

cout << INPUT\_MESSAGE << endl;

int itemsCount = ReadNaturalNum();

Инициализация и вывод нового списка:

InitializeNewList(itemsCount);

PrintList();

Вывод сообщения и ввод позиции для вставки нового элемента

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

Обработка списка и его новый вывод:

Add("KONICHIWA", index);

PrintList();

Завершение выполнения программы:

return 0;

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#define LENGTH\_ERROR "Error: the list couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new list:"

#define NEW\_INPUT\_MESSAGE "Enter list's index where you'll add new word:"

using namespace std;

struct List2

{

char\* item;

List2 \*nextItem;

List2 \*prevItem;

};

List2 \*head;

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

void Add(char\* value, int position)

{

List2 \*node = new List2;

node->item = value;

if (head == NULL)

{

node->nextItem = node;

node->prevItem = node;

head = node;

}

else

{

List2\* p = head;

for(int i = position; i > 1; i--)

p = p->nextItem;

p->prevItem->nextItem = node;

node->prevItem = p->prevItem;

node->nextItem = p;

p->prevItem = node;

}

}

void InitializeNewList(int length)

{

if (length <= 0)

cout << LENGTH\_ERROR << endl;

else

for (int i = 1; i <= length; i++)

Add("hello", i);

}

void PrintList()

{

List2\* printList = head;

do

{

cout<< printList->item << ' ';

printList = printList->nextItem;

}

while(printList != head);

cout << endl;

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemCount = ReadNaturalNum();

InitializeNewList(itemCount);

PrintList();

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

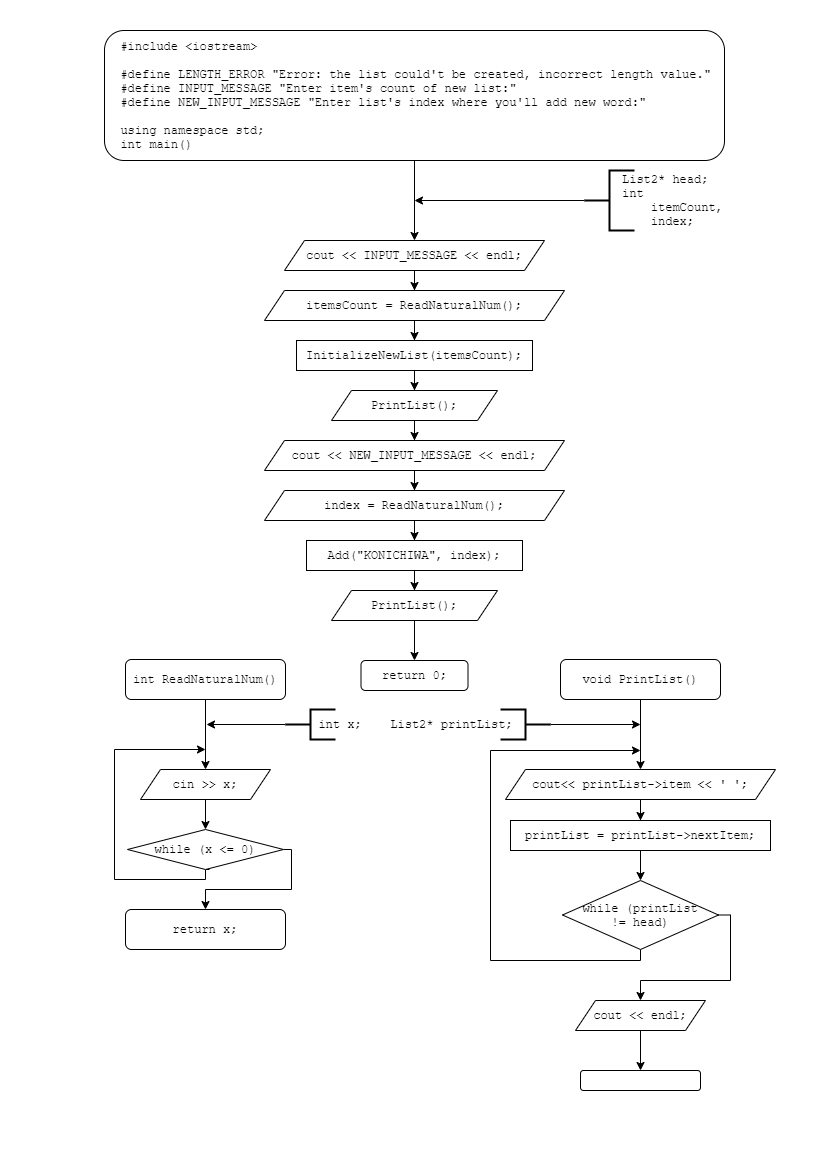
Add("KONICHIWA", index);

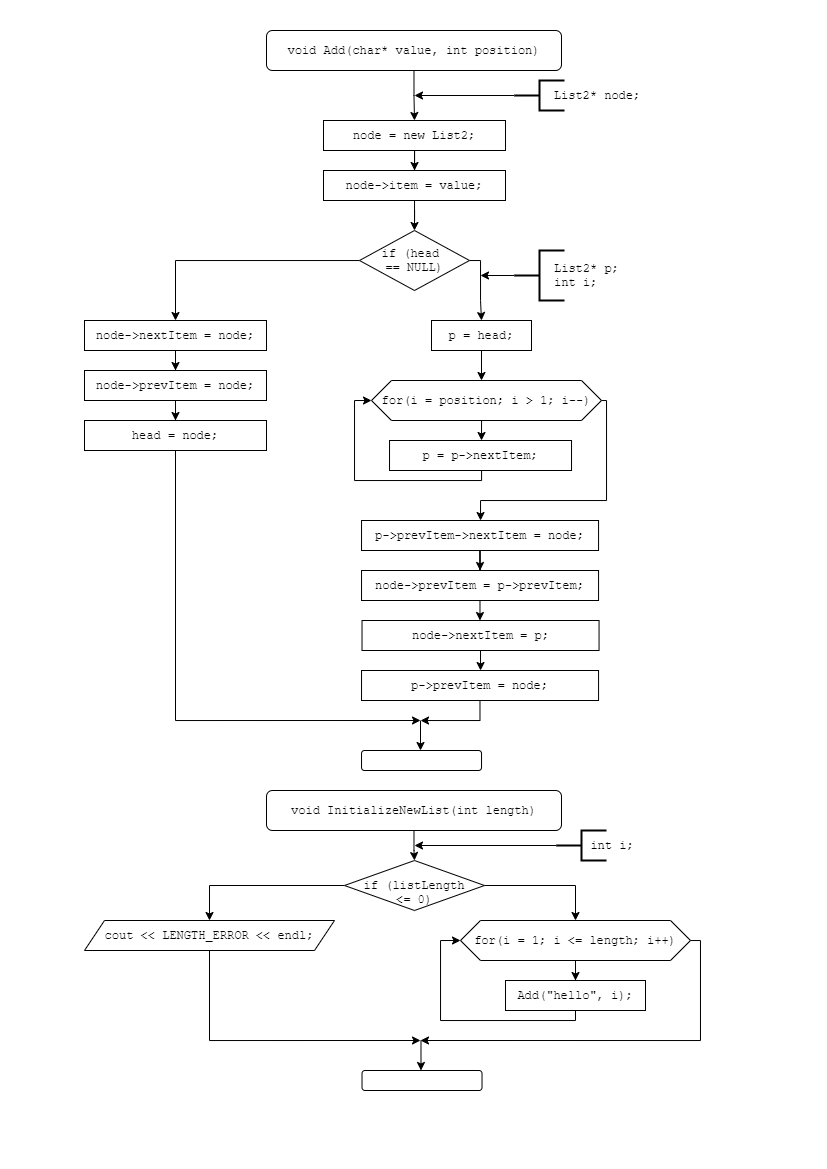
PrintList();

return 0;

}

**Блок-схема** алгоритма программы:





**Реализация при помощи библиотеки STL:**

Весь тот функционал, что был реализован выше содержится в библиотеке STL. Двунаправленные списки объявляются следующим образом:

list<тип\_данных\_инф.поля> имя\_переменной [= значение];

Также для того, чтобы добавить новый элемент, используются функции push\_back (добавление в конец списка) и push\_front (добавление в начало списка).

Для удаления элементов используются функции pop\_back и pop\_front – удаления первого и последнего элемента, соответственно.

Функция, отвечающая за инициализацию нового списка, работает по следующим шагам:

1. Идёт проверка на длину списка, который нужно проинициализировать (если число не натурально (<= 0), то выводится сообщение об ошибке;
2. Через цикл for на каждой итерации добавляется с конца один элемент;
3. Возвращает полученный список;

list<char\*> NewList(int listLength)

{

if (listLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return {};

}

else

{

list<char\*> newList;

for (int i = 0; i < listLength; i++)

newList.push\_back("hello");

return newList;

}

}

Для того, чтобы вывести все элементы списка на экран, необходимо пройти по каждому элементу и вывести значение его информационного поля на экран.

Для того чтобы обращаться к каждом элементу по отдельности в библиотеке STL существуют так называемые итераторы, которые хранят адрес одного поля списка и его порядковый номер, чтобы перемещаться от начала до конца списка.

Поведение итератора можно задать через привычный цикл for, но вместо целочисленной переменной, будет стоять итератор, который будет проходит по каждому элементу списка (от начала и до конца).

void PrintList(list<char\*>& printList)

{

list<char\*>::iterator item;

for (item = printList.begin(); item != printList.end(); item++)

cout << \*item << ' ';

cout << endl;

}

Несколько другая реализация, в отличие от реализации через структуру, реализация функции добавления нового элемента по индексу. Вместо «вставки» алгоритм работает следующим образом:

1. От первого элемента до индекса добавления (не включая данный индекс) элементы переносятся в новый список;
2. В новый список добавляется новый элемент;
3. В новый список переносятся оставшиеся элементы старого списка;
4. Значение старого списка присваивается на новый;

Поскольку, в быту позиции считаются с единицы (1), а в программировании с нуля (0), то функция в начале выполнения будет снижать значения индекса на единицу.

void AddNewItem(list<char\*>& baseList, char\* value, int position)

{

position--;

list<char\*> newList;

int i, length = baseList.size();

for (i = 0; i < position; i++)

{

newList.push\_back(baseList.front());

baseList.pop\_front();

}

newList.push\_back(value);

for (i = position; i < length; i++)

{

newList.push\_back(baseList.front());

baseList.pop\_front();

}

baseList = newList;

}

Реализация метода main в сравнении с реализацией через структуру почти не изменилась: всё тот-же вывод сообщение и получение на ввод тех-же данных.

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemCount = ReadNaturalNum();

list<char\*> items = NewList(itemCount);

PrintList(items);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

AddNewItem(items, "KONICHIWA", index);

PrintList(items);

return 0;

}

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#include <list>

#include <iterator>

#define LENGTH\_ERROR "Error: the list couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new list:"

#define NEW\_INPUT\_MESSAGE "Enter list's index where you'll add new word:"

using namespace std;

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

list<char\*> NewList(int listLength)

{

if (listLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return {};

}

else

{

list<char\*> newList;

for (int i = 0; i < listLength; i++)

newList.push\_back("hello");

return newList;

}

}

void PrintList(list<char\*>& printList)

{

list<char\*>::iterator item;

for (item = printList.begin(); item != printList.end(); item++)

cout << \*item << ' ';

cout << endl;

}

void AddNewItem(list<char\*>& baseList, char\* value, int position)

{

position--;

list<char\*> newList;

int i, length = baseList.size();

for (i = 0; i < position; i++)

{

newList.push\_back(baseList.front());

baseList.pop\_front();

}

newList.push\_back(value);

for (i = position; i < length; i++)

{

newList.push\_back(baseList.front());

baseList.pop\_front();

}

baseList = newList;

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemCount = ReadNaturalNum();

list<char\*> items = NewList(itemCount);

PrintList(items);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

AddNewItem(items, "KONICHIWA", index);

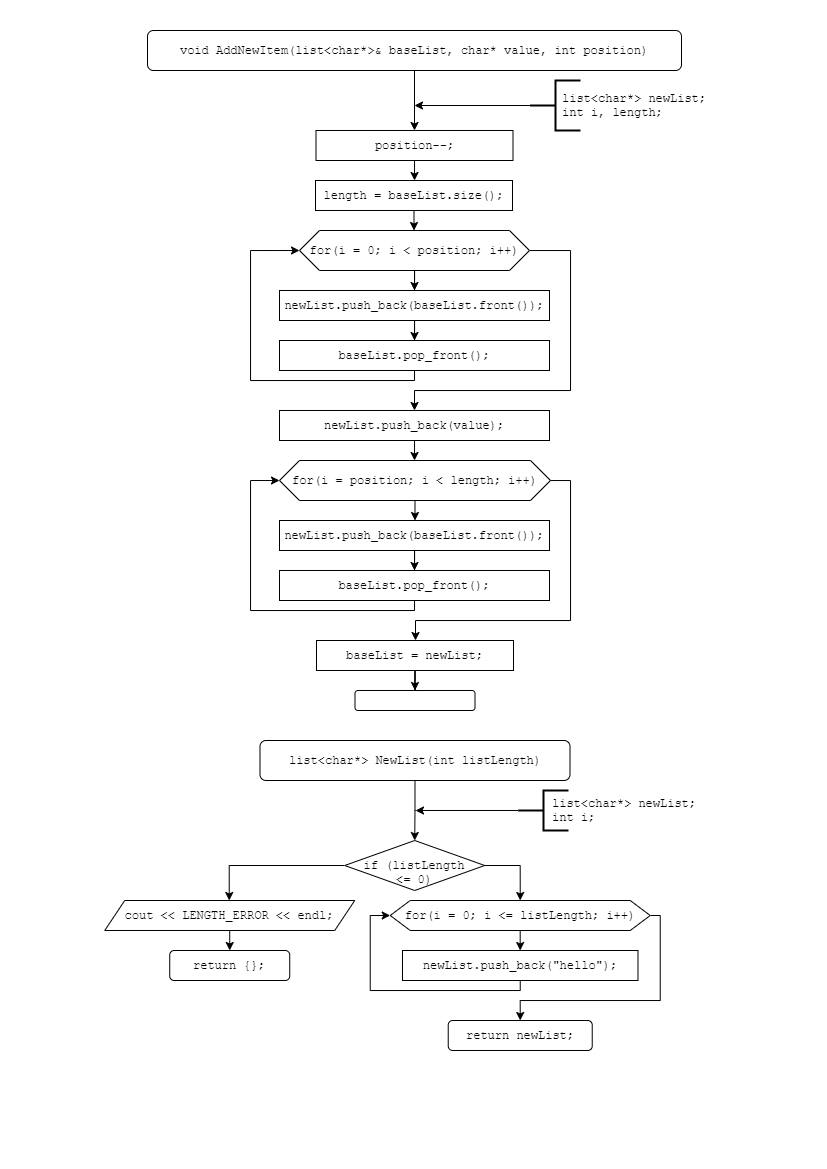
PrintList(items);

return 0;

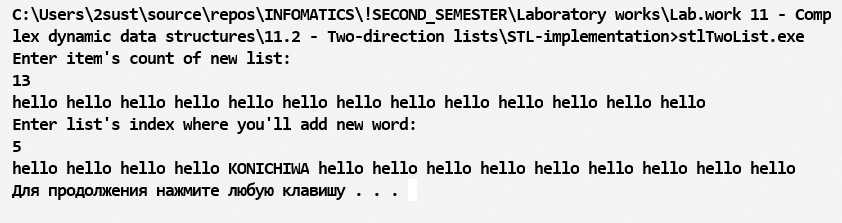
}

**Блок-схема** алгоритма программы:





**Скриншот** выполненной программы:



**11.3: Стеки:**

Стек (англ. Stack – стопка) – динамический тип данных, который представляет собой список элементов, организованных по принципу LIFO: Last In – First Out: последним пришёл – первым вышел. Такое поведение можно сравнить со стопкой тарелок или с патронами в обойме автомата.

**Реализация при помощи структур:**

В C++ элемента стека объявляется следующим образом:

struct Stack

{

int item;

Stack \*nextItem;

};

Для того, чтобы добавлять новые элементы в стек, реализована функция push, которая добавляет один элемент в стек. Если стек, в который необходимо добавить элемент не существует (указатель возвращает NULL), то создаётся новый стек с одним элементом.

В качестве аргументов выступает сам стек, в который необходимо добавить новый элемент и значение, которое будет записано в информационное поле элемента.

void Push(Stack \*\*topStack, int value)

{

Stack \*q;

q = new Stack();

q->item = value;

if (topStack == NULL)

\*topStack = q;

else

{

q->nextItem = \*topStack;

\*topStack = q;

}

}

Теперь, когда появилась возможность добавлять элементы по одному, для удобства реализуем функцию, которая может полностью проинициализировать новый стек определённой длины.

В качестве аргумента принимается сам стек, который нужно проинициализировать и число, которое отвечает за количество элементов стека. В цикле for, на каждой итерации вызывается метод Push, который, как описано выше, добавляет один элемент в стек; каждый новый элемент – случайное число.

void InitiateNewRandomStack(Stack\*\* stack ,int stackLength)

{

for (int i = 0; i < stackLength; i++)

Push(stack, rand());

}

Для того, чтобы удалить первый чётный элемент, реализована функция DeleteFirstOdd, который в качестве аргумента принимает стек, из которого и будет удаляться первый чётный элемент.

Функция проходит по каждому элементу стека, проверяя значение информационного поля каждого элемента на делимость на 2.

void DeleteFirstOdd(Stack \*\*stack)

{

Stack \*q = \*stack;

Stack \*prev = NULL;

while (q != NULL)

{

if (q->item % 2 == 0)

{

//удаление первого чётного элемента

}

prev = q;

q = q->nextItem;

}

}

После того, как чётное число было найдено, происходит проверка на позицию найденного элемента (если элемент первый в списке, то есть является первым элементом аргумента, то просто удаляется этот элемента, иначе проводим перемещения элементов). Сразу после успешного удаления, функция выходит из цикла при помощи ключевого слова break.

//удаление первого чётного элемента

if (q == \*stack)

{

\*stack = q->nextItem;

free(q);

q->item = NULL;

q->nextItem = NULL;

}

else

{

prev->nextItem = q->nextItem;

free(q);

q->item = NULL;

q->nextItem = NULL;

}

break;

Также для того, чтобы выводить элементы стека на экран, необходимо реализовать функцию вывода стека. Назовём её PrintStack, которая в качестве аргумента принимает стек, который нужно вывести на экран.

Функция проходит по каждому элементу стека и организовывает структурированный вывод через пробелы с помощью ключевого слова ‘\t’.

void PrintStack(Stack \*top)

{

Stack \*q = top;

while (q)

{

cout << q->item << '\t';

q = q->nextItem;

}

cout << endl;

}

Реализация необходимых функций, решающих все условия задачи готова, можно приступить к реализации главной функции main.

Вывод сообщения в вводе натурального числа, и его принятие на ввод с клавиатуры:

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

Инициализация нового стека и его вывод на экран:

Stack \*stack = NULL;

InitiateNewRandomStack(&stack, length);

PrintStack(stack);

Удаление первого чётного элемента и последующий вывод обработанного стека:

DeleteFirstOdd(&stack);

PrintStack(stack);

Завершение выполнения программы:

return 0;

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#define LENGTH\_ERROR "Error: the stack couldn't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new stack:"

using namespace std;

struct Stack

{

int item;

Stack \*nextItem;

};

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

void Push(Stack \*\*topStack, int value)

{

Stack \*q;

q = new Stack();

q->item = value;

if (topStack == NULL)

\*topStack = q;

else

{

q->nextItem = \*topStack;

\*topStack = q;

}

}

void DeleteFirstOdd(Stack \*\*stack)

{

Stack \*q = \*stack;

Stack \*prev = NULL;

while (q != NULL)

{

if (q->item % 2 == 0)

{

if (q == \*stack)

{

\*stack = q->nextItem;

free(q);

q->item = NULL;

q->nextItem = NULL;

}

else

{

prev->nextItem = q->nextItem;

free(q);

q->item = NULL;

q->nextItem = NULL;

}

break;

}

prev = q;

q = q->nextItem;

}

}

void PrintStack(Stack \*top)

{

Stack \*q = top;

while (q)

{

cout << q->item << '\t';

q = q->nextItem;

}

cout << endl;

}

void InitiateNewRandomStack(Stack\*\* stack ,int stackLength)

{

for (int i = 0; i < stackLength; i++)

Push(stack, rand());

}

int main()

{

Stack \*stack = NULL;

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

InitiateNewRandomStack(&stack, length);

PrintStack(stack);

DeleteFirstOdd(&stack);

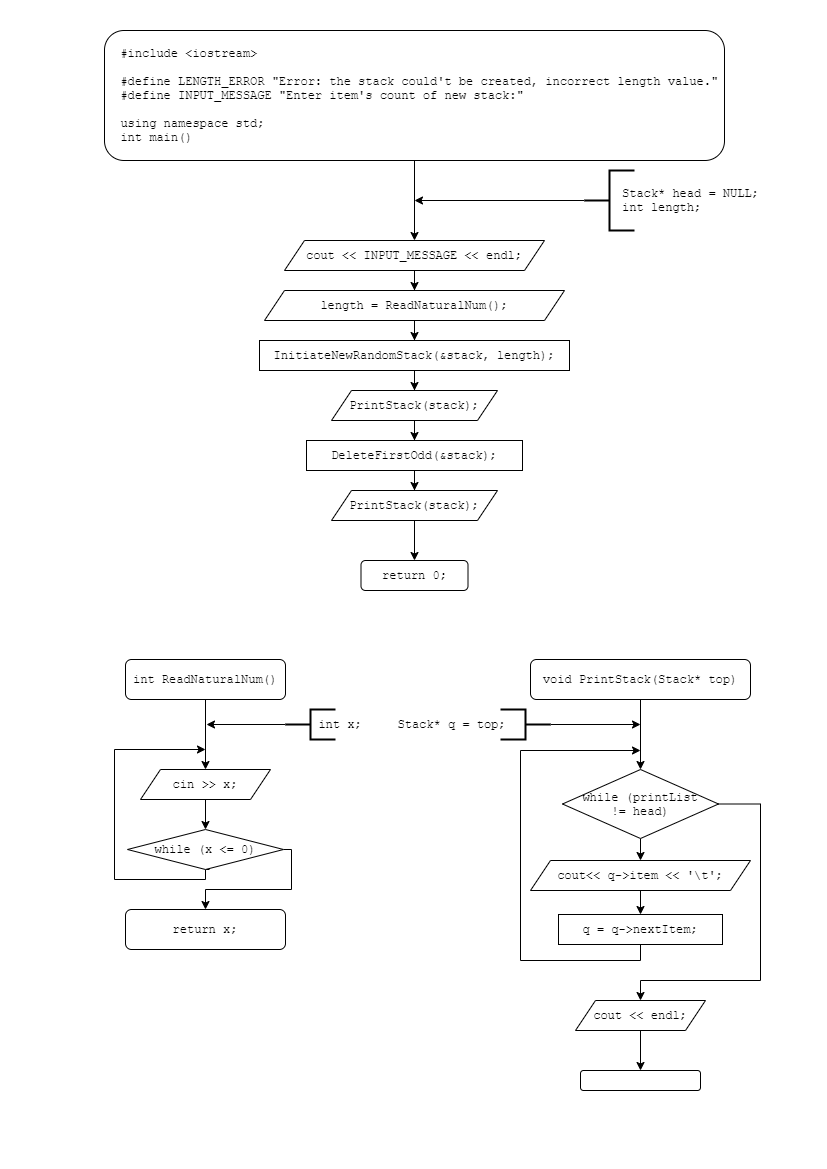
PrintStack(stack);

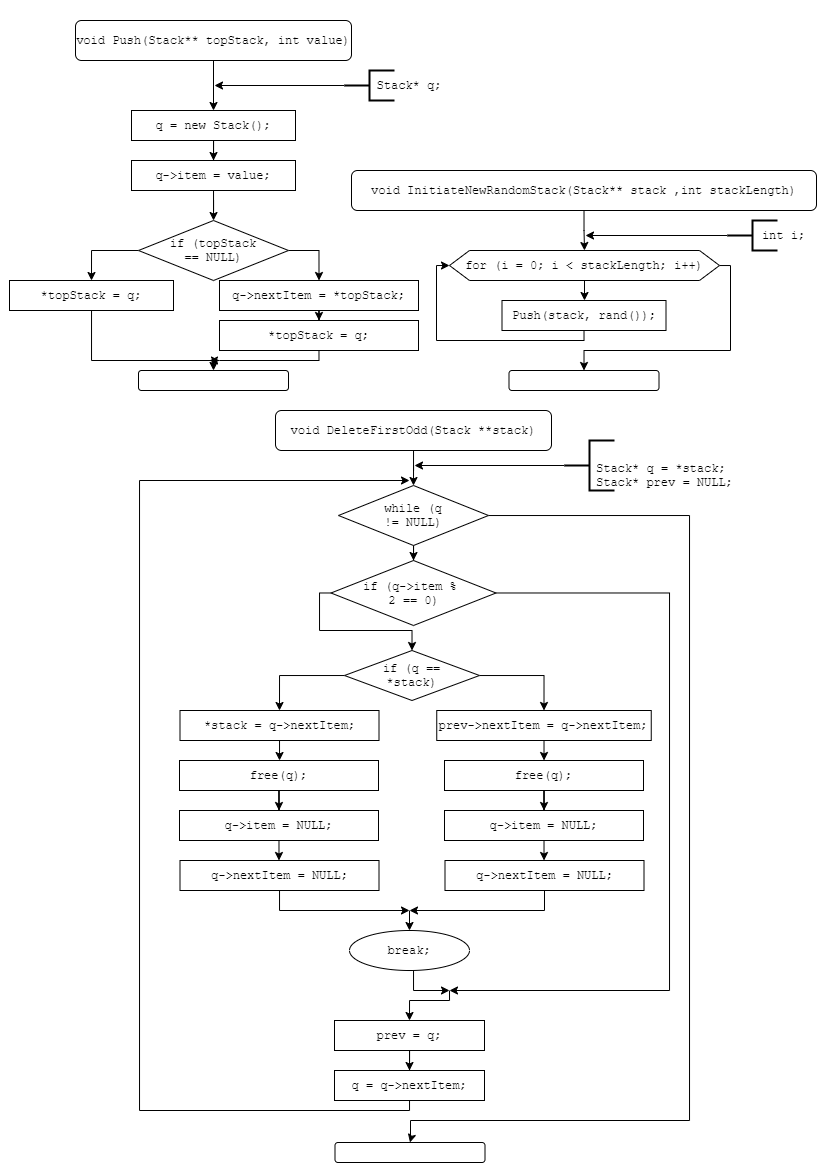
system("pause");

return 0;

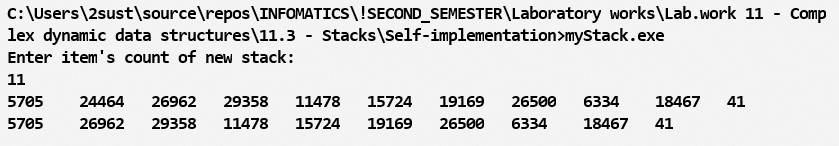
}

**Блок-схема** алгоритма программы:

****

****

**Скриншот** работы выполненной программы:



**Реализация при помощи библиотеки STL:**

Весь функционал, что был реализован выше, содержится в библиотеке STL. Стеки объявляются следующим образом:

stack<тип\_данных\_инф.поля> имя\_переменной [= значение];

Также для того, чтобы добавить новый элемент, используется метод push, а для удаления – метод pop.

Соответственно, функция, отвечающая за инициализацию нового стека, остаётся неизменной в сравнении с реализацией через структуру. Всё тот-же цикл for и число, отвечающие за количество элементов в стеке.

stack<int> NewRandomStack(int stackLength)

{

stack<int> newStack;

for (int i = 0; i < stackLength; i++)

newStack.push(rand());

return newStack;

}

По такому же принципу, как и ранее, работает функция вывода элементов стека на экран, но вместо переприсваивания указателей, первый элемент удаляется через метод pop.

void PrintStack(stack<int> stack)

{

int length = stack.size();

for (int i = 0; i < length; i++)

{

cout << stack.top() << '\t';

stack.pop();

}

cout << endl;

}

Функция удаления первого чётного числа в сравнении с реализацией через структуру заметно отличается. Алгоритм, следующий:

1. Объявляется вспомогательный (промежуточной стек) – если данные записать напрямую, то получится стек, в котором элементы поменялись местами (первый стал последним, последний – первым, второй – предпоследним и т.д.).
2. Идёт проход по каждому элементу с проверкой на два условия
   1. Является ли элемент чётным;
   2. Был ли до этого другой чётный элемент удалён;
3. Если условие не выполняется, то элемент базового стека записывается в промежуточный, иначе же элемент на записывается, а флажок, символизирующий об удалении чётного элемента, получает значение true (да, элемент удалён);
4. Если элемент удалён, то длина нового стека уменьшается на единицу (т. к. в промежуточном стеке на 1 элемент меньше, чем в базовом);
5. В результате данных действий, базовый стек стал пустым, поэтому запись будет производиться из промежуточного стека обратно в базовый;

void DeleteFirstOdd(stack<int>& baseStack)

{

bool deletedOdd = false;

int length = baseStack.size();

stack<int> newStack;

for (int i = 0; i < length; i++)

{

if ((baseStack.top() % 2 == 0) && !deletedOdd)

deletedOdd = true;

else

newStack.push(baseStack.top());

baseStack.pop();

}

if (deletedOdd)

length--;

for (int i = 0; i < length; i++)

{

baseStack.push(newStack.top());

newStack.pop();

}

}

Реализация метода main в сравнении с реализацией через структуру почти не изменилась: всё тот-же вывод сообщение и получение на ввод тех-же данных.

int main()

{

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

stack<int> stack = NewRandomStack(length);

PrintStack(stack);

DeleteFirstOdd(stack);

PrintStack(stack);

return 0;

}

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#include <stack>

#define INPUT\_MESSAGE "Enter item's count of new stack:"

using namespace std;

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

stack<int> NewRandomStack(int stackLength)

{

stack<int> newStack;

for (int i = 0; i < stackLength; i++)

newStack.push(rand());

return newStack;

}

void PrintStack(stack<int> stack)

{

int length = stack.size();

for (int i = 0; i < length; i++)

{

cout << stack.top() << ' ';

stack.pop();

}

cout << endl;

}

void DeleteFirstOdd(stack<int>& baseStack)

{

bool deletedOdd = false;

int length = baseStack.size();

stack<int> newStack;

for (int i = 0; i < length; i++)

{

if ((baseStack.top() % 2 == 0) && !deletedOdd)

deletedOdd = true;

else

newStack.push(baseStack.top());

baseStack.pop();

}

if (deletedOdd)

length--;

for (int i = 0; i < length; i++)

{

baseStack.push(newStack.top());

newStack.pop();

}

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

stack<int> stack = NewRandomStack(length);

PrintStack(stack);

DeleteFirstOdd(stack);

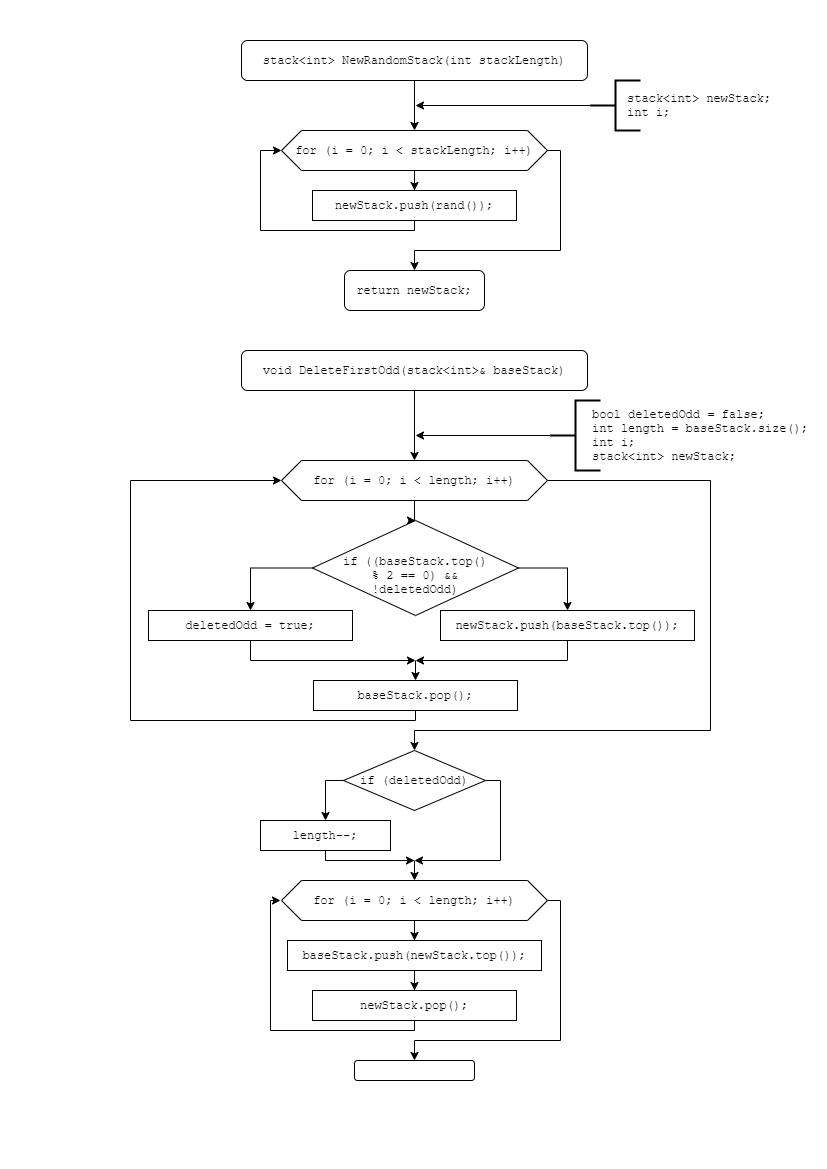
PrintStack(stack);

return 0;

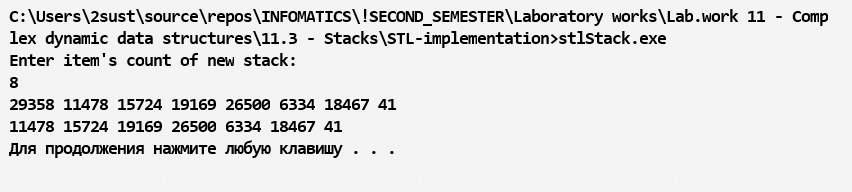
}

**Блок-схема** алгоритма программы:





**Скриншот** работы выполненной программы:



**11.4: Очереди:**

Очередь (англ. queue) – это динамическая структура данных, которая работает по принципу FIFO (First In – First out): первый пришёл – первый ушёл. Очередь поддерживает те же операции, что и стек, но принцип работы отличается от стека (у стека – LIFO).

Поведение очереди можно сравнить с реальной очередью в магазине: кто раньше пришёл к кассе, того раньше обслужат. Точно также устроена автомобильная пробка на шоссе.

**Реализация при помощи структур:**

В языке программирования **C++**: такие объекты объявляются следующим образом:

struct Queue

{

char\* item;

List\* nextItem;

};

Для того, чтобы добавлять новые элементы в очередь, реализована функция Push, которая добавляет один элемент в конец списка.

void Push(Queue\*& lastItem, char\* word)

{

Queue\* workQueue = new Queue;

workQueue->item = word;

workQueue->nextItem = lastItem;

lastItem = workQueue;

delete workQueue;

}

Для удобства работы с очередью объявлена глобальная переменная \_queueLength, которая хранит количество элементов очереди.

int \_queueLength = 0;

Для того, чтобы удалять элементы из очереди, реализована функция Pop.

void Pop(Queue\*& firstItem, Queue\*& lastItem)

{

Queue\* workQueue = lastItem;

int length = 0;

while (workQueue != NULL)

{

length++;

workQueue = workQueue->nextItem;

}

if (length == 1)

{

char\* t = lastItem->item;

delete workQueue;

firstItem = NULL;

lastItem = NULL;

}

else

{

while (workQueue->nextItem->nextItem != NULL)

workQueue = workQueue->nextItem;

Queue\* tempQueue = workQueue;

workQueue = workQueue->nextItem;

tempQueue->nextItem = NULL;

firstItem = tempQueue;

delete workQueue;

}

}

Для того, чтобы проинициализировать новых очередь, реализована функция NewQueue, которая заполняет все элементы словом hello. Но перед этим идёт проверка на длину новой очереди (если число < 1, то выводится сообщение об ошибке и возвращается NULL).

Queue\* NewQueue(int queueLength, Queue\*& firstItem, Queue\*& lastItem)

{

if (queueLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return NULL;

}

else

{

//заполнение очереди

}

}

Заполнение происходит следующим образом: если количество элементов очереди равняется 1, то он просто записывается как последний элемент, если больше, то помимо этого действия, через цикл for в начало каждый раз будут записываться новые элементы в начало очереди.

//заполнение очереди

Queue\* p = new Queue;

p->item = "hello";

p->nextItem = NULL;

firstItem = p;

lastItem = p;

for (int i = 1; i < queueLength; i++)

{

Queue\* h = new Queue;

h->item = "hello";

h->nextItem = p;

lastItem = h;

p = lastItem;

}

Функции вывода элементов очереди на экран работает по тому же принципу, что и вывод на экран описанных выше динамических структур данных, реализованных через структуру. В цикле while идёт проверка на существование элемента и его дальнейший вывод через пробелы.

void PrintQueue(Queue\* lastItem)

{

Queue\* workQueue = lastItem;

while (workQueue != NULL)

{

cout << workQueue->item << " ";

workQueue = workQueue->nextItem;

}

cout << endl;

}

Также, по заданию, необходимо реализовать метод, добавляющий/вставляющий новое слово в очередь по его номеру. Функция работает по следующему принципу:

1. Создаётся новая очередь;
2. С начала и до позиции добавления нового элемента (не включая его) элементы старой очереди без изменений переносятся в новую;
3. Добавляется новое слово (для того, чтобы их отличать добавим японское слово konichiwa (привет);
4. В новую очередь переносятся оставшиеся элементы старой очереди;
5. Старая очередь меняется на новую;

void AddNewItem(Queue\*& firstItem, Queue\*& lastItem, int index)

{

index--;

Queue\* newQueue = new Queue;

newQueue->item = "";

newQueue->nextItem = NULL;

int pos = -1;

for (int i = 0; i < index; i++)

{

Push(newQueue, firstItem->item);

pos++;

}

Push(newQueue, "KONICHIWA");

for (int i = pos + 1; i < \_queueLength; i++)

{

Push(newQueue, firstItem->item);

}

lastItem = newQueue;

}

Реализация необходимых функций готова, приступаем к реализации основного метода main.

Базовая реализация устроена точно также, как и в программах выше:

* Сообщение о вводе данных;
* Приём данных с клавиатуры;
* Инициализация новой очереди;
* Вывод очереди на экран;
* Обработка очереди (добавление нового по номеру);
* Вывод на экран новой обработанной очереди;

int main()

{

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

\_queueLength = length;

Queue\* firstItem, \*lastItem;

NewQueue(length, firstItem, lastItem);

PrintQueue(lastItem);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

AddNewItem(firstItem, lastItem, index);

PrintQueue(lastItem);

return 0;

}

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#define LENGTH\_ERROR "Error: the queue could't be created, incorrect length value."

#define INPUT\_MESSAGE "Enter item's count of new queue:"

#define NEW\_INPUT\_MESSAGE "Enter queue's index where you'll add new word:"

using namespace std;

struct Queue

{

char\* item;

Queue\* nextItem;

};

int \_queueLength = 0;

void Push(Queue\*& lastItem, char\* word)

{

Queue\* workQueue = new Queue;

workQueue->item = word;

workQueue->nextItem = lastItem;

lastItem = workQueue;

}

void Pop(Queue\*& firstItem, Queue\*& lastItem)

{

Queue\* workQueue = lastItem;

int length = 0;

while (workQueue != NULL)

{

length++;

workQueue = workQueue->nextItem;

}

if (length == 1)

{

char\* t = lastItem->item;

delete workQueue;

firstItem = NULL;

lastItem = NULL;

}

else

{

while (workQueue->nextItem->nextItem != NULL)

workQueue = workQueue->nextItem;

Queue\* tempQueue = workQueue;

workQueue = workQueue->nextItem;

tempQueue->nextItem = NULL;

firstItem = tempQueue;

delete workQueue;

}

}

Queue\* NewQueue(int queueLength, Queue\*& firstItem, Queue\*& lastItem)

{

if (queueLength <= 0)

{

cout << LENGTH\_ERROR << endl;

return NULL;

}

else

{

Queue\* p = new Queue;

p->item = "hello";

p->nextItem = NULL;

firstItem = p;

lastItem = p;

for (int i = 1; i < queueLength; i++)

{

Queue\* h = new Queue;

h->item = "hello";

h->nextItem = p;

lastItem = h;

p = lastItem;

}

}

return firstItem;

}

void PrintQueue(Queue\* lastItem)

{

Queue\* workQueue = lastItem;

while (workQueue != NULL)

{

cout << workQueue->item << " ";

workQueue = workQueue->nextItem;

}

cout << endl;

}

void AddNewItem(Queue\*& firstItem, Queue\*& lastItem, int index)

{

index--;

Queue\* newQueue = new Queue;

newQueue->item = "";

newQueue->nextItem = NULL;

int pos = -1;

for (int i = 0; i < index; i++)

{

Push(newQueue, firstItem->item);

pos++;

}

Push(newQueue, "KONICHIWA");

for (int i = pos + 1; i < \_queueLength; i++)

{

Push(newQueue, firstItem->item);

}

lastItem = newQueue;

}

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int length = ReadNaturalNum();

\_queueLength = length;

Queue\* firstItem, \*lastItem;

NewQueue(length, firstItem, lastItem);

PrintQueue(lastItem);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

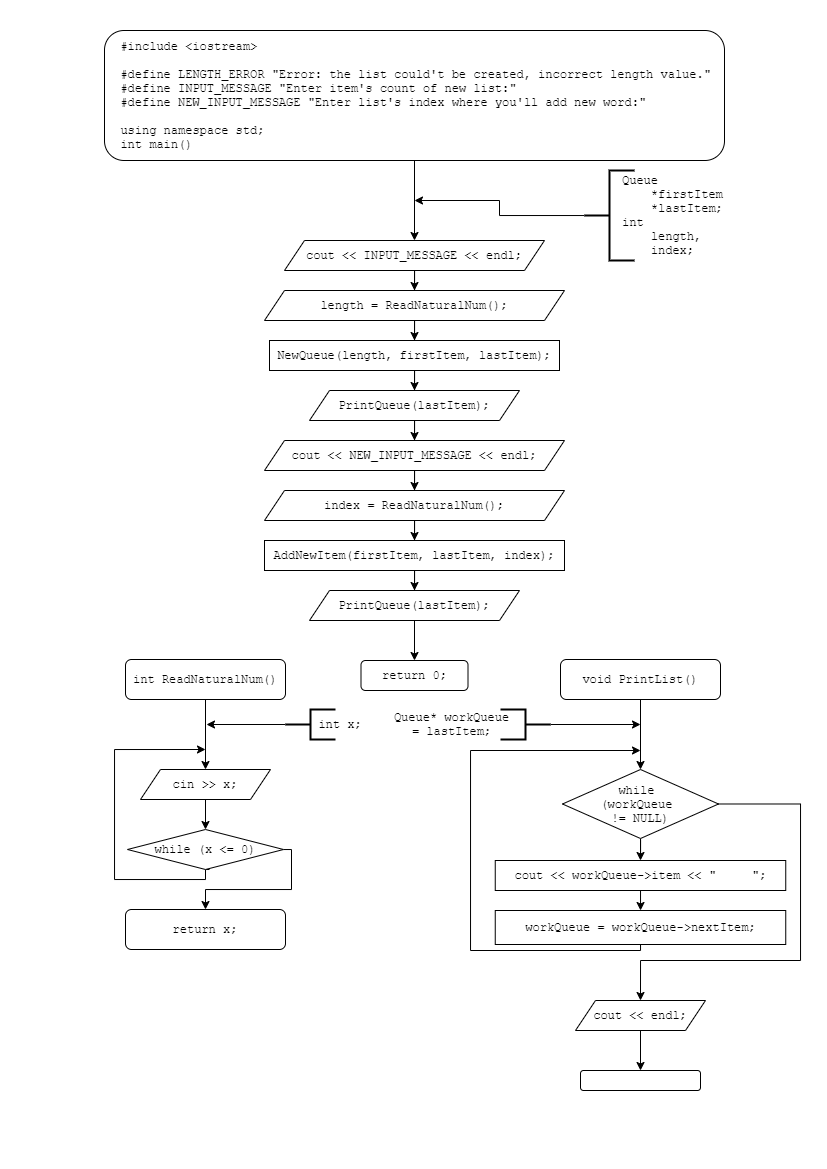
AddNewItem(firstItem, lastItem, index);

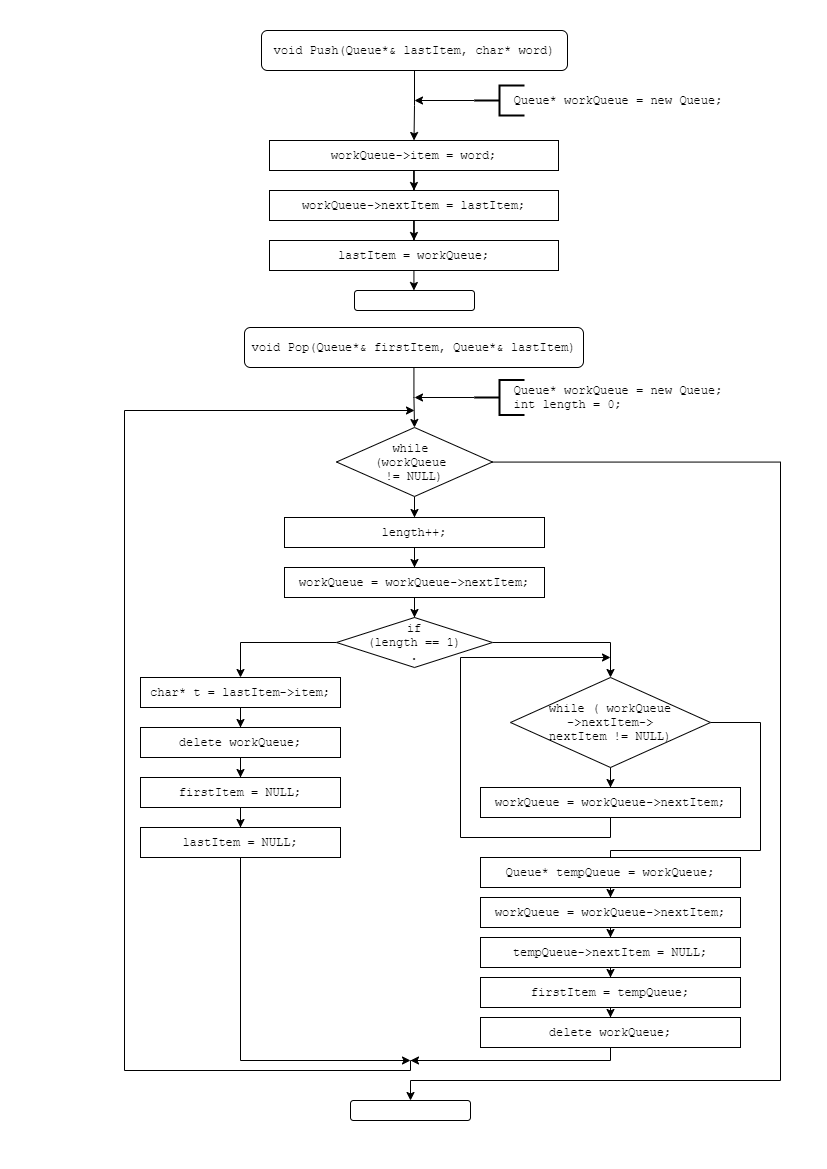
PrintQueue(lastItem);

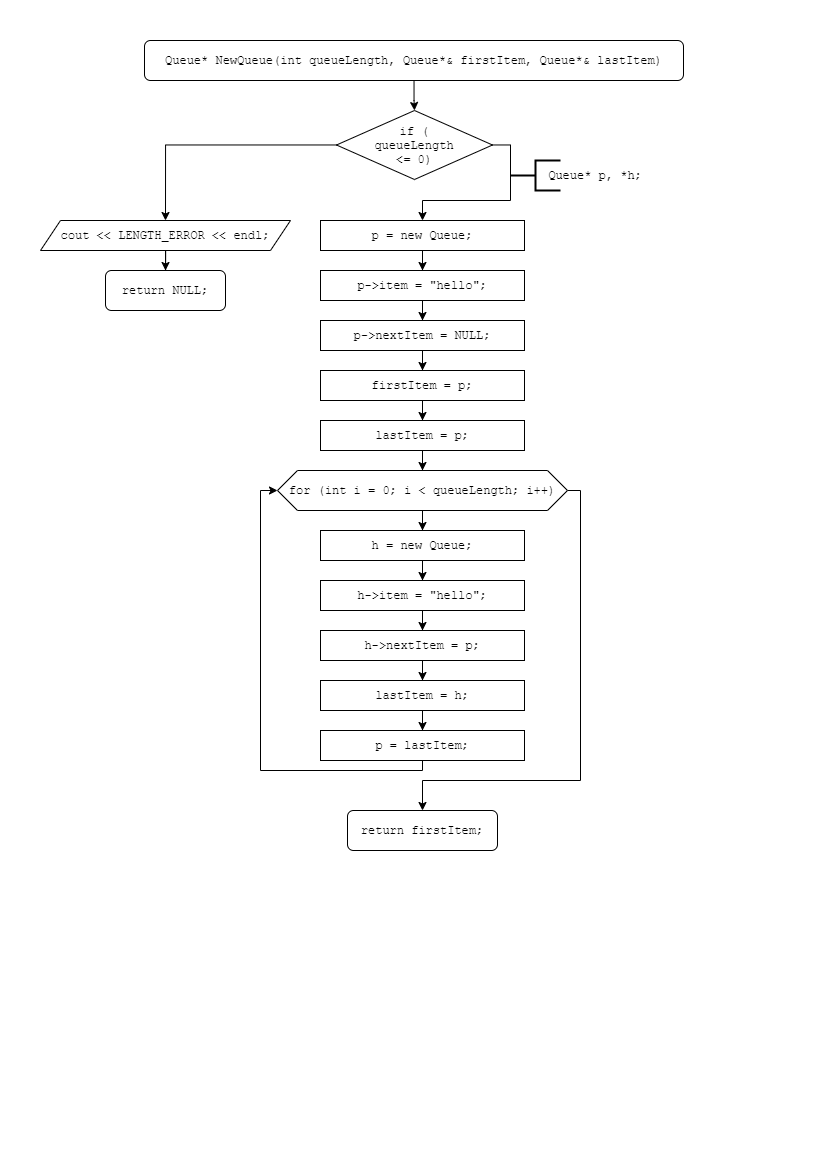
return 0;

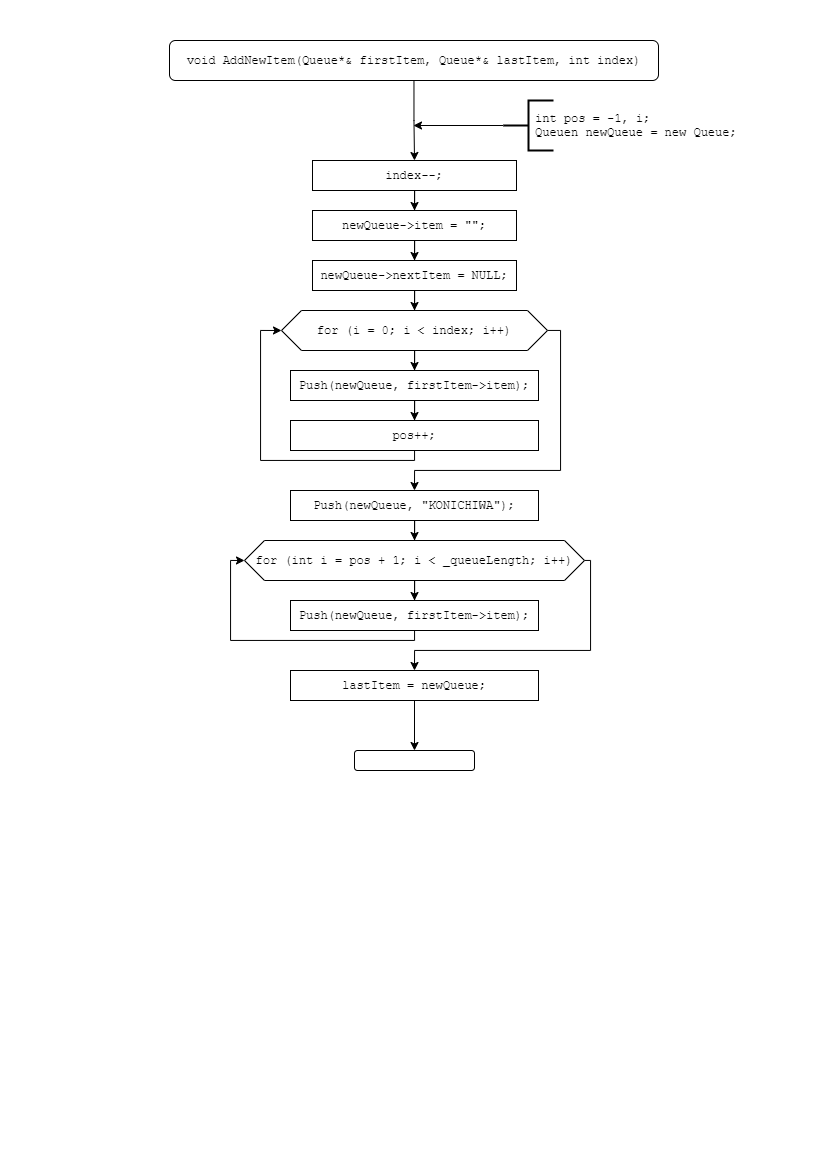
}

**Блок-схема** алгоритма программы:

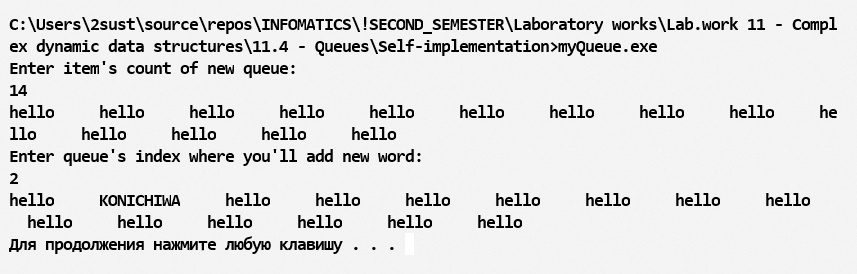








**Скриншот** выполненной программы:



**Реализация при помощи библиотеки STL:**

Функция, которая инициализирует новую очередь:

queue<char\*> NewQueue(int queueLength)

{

queue<char\*> newQueue;

for (int i = 0; i < queueLength; i++)

newQueue.push("hello");

return newQueue;

}

Функция, которая выводит элементы очереди на экран:

void PrintQueue(queue<char\*> queue)

{

while (!queue.empty())

{

cout << queue.front() << ' ';

queue.pop();

}

cout << endl;

}

Функция, которая добавляет новое слово в очередь:

void AddNewItem(queue<char\*>& baseQueue, int index)

{

queue<char\*> newQueue;

int i = 0;

while (!baseQueue.empty())

{

i++;

if (i != index)

{

newQueue.push(baseQueue.front());

baseQueue.pop();

}

else

{

newQueue.push("KONICHIWA");

}

}

baseQueue = newQueue;

}

Главная функция:

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemCount = ReadNaturalNum();

queue<char\*> queue = NewQueue(itemCount);

PrintQueue(queue);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

AddNewItem(queue, index);

PrintQueue(queue);

return 0;

}

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

#include <queue>

#define INPUT\_MESSAGE "Enter item's count of new queue:"

#define NEW\_INPUT\_MESSAGE "Enter queue's index where you'll add new word:"

using namespace std;

int ReadNaturalNum()

{

int x;

do cin >> x; while (x <= 0);

return x;

}

queue<char\*> NewQueue(int queueLength)

{

queue<char\*> newQueue;

for (int i = 0; i < queueLength; i++)

newQueue.push("hello");

return newQueue;

}

void PrintQueue(queue<char\*> queue)

{

while (!queue.empty())

{

cout << queue.front() << ' ';

queue.pop();

}

cout << endl;

}

void AddNewItem(queue<char\*>& baseQueue, int index)

{

queue<char\*> newQueue;

int i = 0;

while (!baseQueue.empty())

{

i++;

if (i != index)

{

newQueue.push(baseQueue.front());

baseQueue.pop();

}

else

{

newQueue.push("KONICHIWA");

}

}

baseQueue = newQueue;

}

int main()

{

cout << INPUT\_MESSAGE << endl;

int itemCount = ReadNaturalNum();

queue<char\*> queue = NewQueue(itemCount);

PrintQueue(queue);

cout << NEW\_INPUT\_MESSAGE << endl;

int index = ReadNaturalNum();

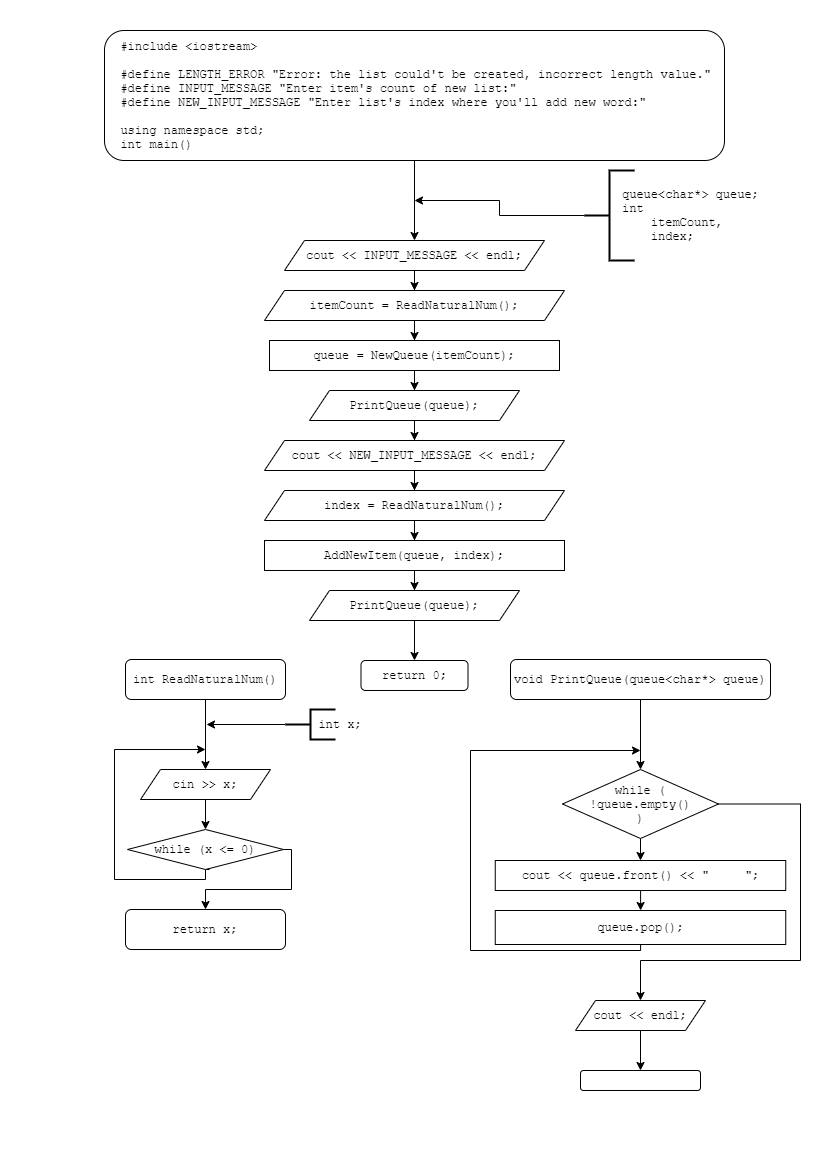
AddNewItem(queue, index);

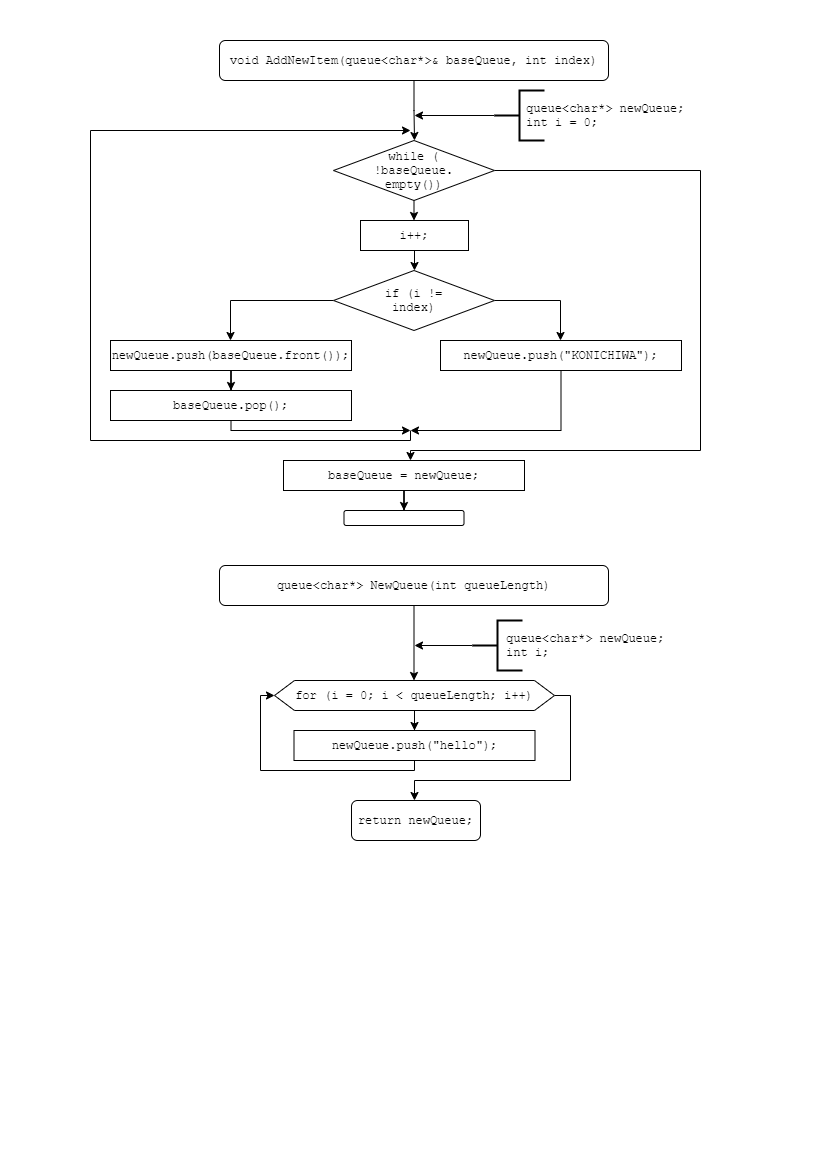
PrintQueue(queue);

return 0;

}

**Блок-схема** алгоритма программы:





**Скриншот** выполненной программы:

