Министерство науки и высшего образования РФ ФГБОУ ВПО

«Пермский государственный национальный исследовательский университет»

Кафедра прикладной математики и информатики

**Индивидуальное задание №2**

**«Игра Шарики»**

***по дисциплине «Алгоритмизация и программирование 2»***

|  |  |
| --- | --- |
|  | **Выполнил:** |
|  | Студент 1-го курса механико-математического факультета  \_\_\_\_\_\_\_\_\_\_\_ Шмырина В.О.,  группа ПМИ-1,2-2021 |
|  | **Проверил:** |
|  | Старший преподаватель кафедры прикладной математики и информатики ПГНИУ,  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шеина Т. Ю. |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_ 2022 г. |

Пермь 2022

# **Постановка задачи**

В одной компьютерной игре игрок выставляет в линию шарики разных цветов. Когда образуется непрерывная цепочка из трех и более шариков одного цвета, она удаляется из линии. Все шарики при этом сдвигаются друг к другу, и ситуация может повториться. Напишите программу, которая по данной ситуации определяет, сколько шариков будет сейчас уничтожено.

*Формат входных данных:*

Даны количество шариков в цепочке (не более 10^5 ) и цвета шариков (от 0 до 9, каждому цвету соответствует свое целое число).

*Формат выходных данных:*

Требуется вывести количество шариков, которое будет уничтожено.

*Пример входных данных:*

5

1 3 3 3 2

*Пример выходных данных:*

3

# Определение идеи алгоритма,

# выбор методов решения и структур данных

Для решения задачи использовалась среда разработки Microsoft Visual Studio C++.

Сначала создаем указатели для работы со списками. Вводим входные данные с и строим список с помощью функции AddToSpis(), добавляя элементы поочередно. Начинаем работать с самим списком.

Для написания программы самым выгодным решением было использовать функцию, которая будет вызывать сама себя до тех пор, пока будет считывать одинаковые значения рядом стоящих шариков (>= 3), при каждом удалении повторений функция также вызывает сама себя, чтобы убедится, не появилось ли новых цепочек.

В коде использовать двусвязные списки нет никакого смысла, т.к. это занимает лишнюю память, можно обойтись двумя указателями, одно на значение перед цепью повторений, второе – на хвост цепочки, это удобно в дальнейшем использовать для удаления. В первый проход (и в каждый новый проход по функции) указатели будут стоять на первом и втором элементе соответственно.

Вызываем функцию proxod(). Обязательно проверяем список на пустоту в самом начале, т.к. все элементы могут быть удалены, а дальнейший ход вызовет ошибку в программе. В функции запускается цикл, суть которого – это движение, пока не обнаружится цепочка повторений >= 3. Каждый раз, когда значения шариков будут совпадать, счетчик k будет увеличиваться. Из-за невозможности считать последний элемент (т.к. указатели tek и prev могут быть одинаковы по значению, это сложнее организовать), число удаляемых элементов всегда будет меньше на 1. Это не влияет на решение, т.к. в последствии устраняется. Учтено также то, что цепочка может быть длинной, поэтому проверяем является ли следующий элемент по значению равным предыдущему, если нет, то выходим из цикла и начинаем процесс удаления.

Удаление элементов организовано с помощью условия, где логическая переменная помогает сразу понять, находится ли цепочка в самом начале, либо в самом конце списка, удаляются все элементы списка, либо не все. Подразделение случаев упрощает дальнейшее удаление. Если же ни под один случай не подходит, программа это также учитывает.

В зависимости от случаев удаления, вызываем функцию Del() для освобождения памяти и переназначения ссылок для элементов. С помощью switch() снова подразделяем случаи удаления (продолжаем делать это из условия) и используем соответствующий, последовательно удаляя повторяющиеся шарики, а после переназначая ссылки оставшимся “сбоку” от удаленной цепочки элементам.

Затем функция вызывает себя еще раз (самопроверка), и если повторных случаев не появилось, программа свою работу заканчивает.

Далее мы выводим значение kol.

Сумма в счетчике и есть количество шариков, которые удалены по условию задачи.

**Тестирование программы**

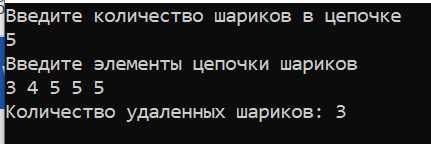
**Тест №1**

Входные данные

5

3 4 5 5 5

Результат



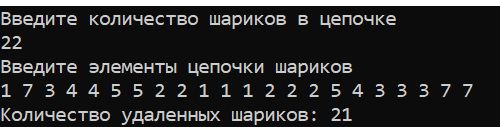
**Тест №2**

Входные данные

22

1 7 3 4 4 5 5 2 2 1 1 1 2 2 2 5 4 3 3 3 7 7

Результат



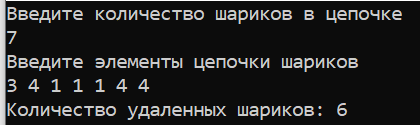
**Тест №3**

Входные данные

7

3 4 1 1 1 4 4

Результат

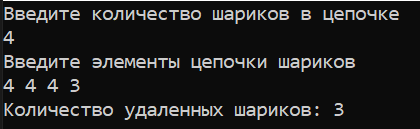


**Тест №4**

Входные данные

4

4 4 4 3



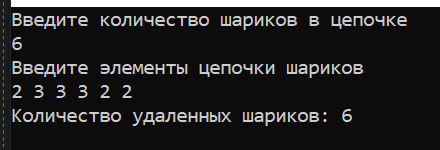
**Тест №5**

Входные данные

6

2 3 3 3 2 2

Результат



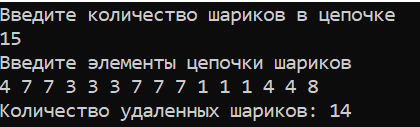
Тест №6

Входные данные

15

4 7 7 3 3 3 7 7 7 1 1 1 4 4 8

Результат



**Текст программы**

Исходный файл программы shariki*.cpp* прилагается к работе в электронном виде, ниже приводим его полностью.

#include "iostream"

#include "iomanip"

#include <time.h>

#include "fstream"

using namespace std;

int kol = 0; //подсчет удаленных шариков

struct Node

{

int data;

Node\* Next;

};

typedef Node\* PNode;

void AddToSpis(PNode& MyNode, PNode& Head, int data)

{

PNode NextNode = MyNode;

if (MyNode == NULL)

{

MyNode = new Node;

MyNode->data = data;

MyNode->Next = NULL;

Head = MyNode;

}

else

{

NextNode = new Node;

MyNode->Next = NextNode;

NextNode->data = data;

NextNode->Next = NULL;

MyNode = NextNode;

}

}

void Del(int k, PNode& Head, PNode tek, PNode prev) //удаление повтор вершин

{

PNode p, p1;

switch (k)

{

case 1:

p = Head;

p1 = p->Next;

while (p != tek)

{

delete p;

p = p1;

p1 = p1->Next;

}

Head = tek->Next;

break;

case 2:

p = prev->Next;

p1 = p->Next;

while (p != tek)

{

delete p;

p = p1;

p1 = p1->Next;

}

prev->Next = tek->Next;

break;

case 3:

p = Head;

p1 = p->Next;

while (p->Next != NULL)

{

delete p;

p = p1;

p1 = p1->Next;

}

Head = NULL;

tek = prev = NULL;

break;

case 4:

p = prev->Next;

p1 = p->Next;

while (p->Next != NULL)

{

delete p;

p = p1;

p1 = p1->Next;

}

prev->Next = NULL;

break;

case 5:

p = prev->Next->Next;

p1 = p->Next;

while (p->Next != NULL)

{

delete p;

p = p1;

p1 = p1->Next;

}

prev->Next->Next = NULL;

break;

case 6:

p = prev->Next;

p1 = p->Next;

while (p != tek)

{

delete p;

p = p1;

p1 = p1->Next;

}

prev->Next = tek->Next;

break;

default:

break;

}

}

void proxod(PNode& Head, int n, PNode tek, PNode prev, int k) //проход пока повторки не < 3, tek - текущая, prev - предыдущая вершина

{

bool Pi = false; //переменная проверки условия, где указатель prev как head является удаляемым звеном

int i = 0;

if (Head == NULL || Head->Next == NULL)

{

return;

}

tek = Head->Next;

prev = Head;

if (tek->data == prev->data)

{

k += 1;

Pi = true;

}

while (tek->Next != NULL && (k >= 3 || i + 1 < n))

{

if (tek->Next != NULL && tek->data == tek->Next->data)

{

k++;

}

else

{

if (tek->Next != NULL && k >= 2 && tek->Next->data != tek->data)

{

break;

}

else

k = 0;

}

tek = tek->Next;

if (tek->Next != NULL && tek->data != prev->Next->data)

{

prev = prev->Next;

}

i++;

}

if (k >= 2)

{

k++;

kol += k;

if (Pi)

{

if (kol != n)

{

if (Head == prev) //для условий где цепь повторов начинается с головы

Del(1, Head, tek, prev);

else

Del(2, Head, tek, prev);

}

else

{

Del(3, Head, tek, prev);

return;

}

}

else

{

if (tek->Next == NULL)

{

if (prev->Next->data == tek->data)

Del(4, Head, tek, prev);

else

Del(5, Head, tek, prev);

}

else

{

Del(6, Head, tek, prev);

}

}

proxod(Head, n, tek, prev, 0);

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

PNode Head, MyNode, PrevNode;

Head = MyNode = PrevNode = NULL;

int n, data; //количество шариков в списке

cout << "Введите количество шариков в цепочке" << endl;

cin >> n;

cout << "Введите элементы цепочки шариков" << endl;

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

{

cin >> data;

AddToSpis(MyNode, Head, data);

}

int k = 0; //количество повторяющихся вершин данной итерации

proxod(Head, n, MyNode, PrevNode, k);

cout << "Количество удаленных шариков: ";

cout << kol << endl;

}