МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания **№2**

Тема: Однонаправленный динамический список

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Иолович Е.А

Фамилия И.О.

группа ИНБО-03-22

Москва 2023

СОДЕРЖАНИЕ

1 ОТЧЕТ ПО ЗАДАНИЮ

0 Ответы на вопросы

1) Существует три уровня представления данных: уровень пользователя (предметная область), логический и физический. Каждый объект предметной области характеризуется своими атрибутами, каждый атрибут имеет имя и значение.

2) Тип данных определяет:

– внутреннее представление данных в памяти компьютера;

– множество значений, которые могут принимать величины этого типа;

– операции и функции, которые можно применять к величинам этого типа.

3) Структура данных – это способы хранить и организовывать данные, чтобы эффективней решать различные задачи. Данные можно представить по-разному. В зависимости от того, что это за данные и что вы собираетесь с ними делать, одно представление подойдёт лучше других. Так же это можно назвать контейнером, который хранит данные в определенном макете. Этот «макет» позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других.

4) Структуры хранения данных могут быть статическими или динамическими. Статические структуры создаются при компиляции программы в области данных и их размер изменить нельзя во время работы программы. Они освобождают память, выделенную им, только по завершении программы. Динамические структуры создаются во время работы программы в динамической памяти, их размер может быть изменен во время работы программы. Память этой структуры хранения может и ее отдельных элементов быть освобождена во время работы программы.

5) Линейные структуры – это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером.

6) Структура данных задачи могут представлять линейный список значений – т.е. список значений, следующих друг за другом. Для таких данных можно использовать векторную структуру данных. Вектор представляет линейную структуру, состоящую из нескольких последовательно расположенных ячеек. Ячейка предназначена только для хранения значения и связи со следующей за ней ячейкой она не хранит.

7) Стек – структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Работает по принципу **FILO** (first in — last out; первый пришел — последний ушел)

8) Очередь – это структура данных, представляющая собой последовательность элементов, образованная в порядке их поступления. Очередь подчиняется принципу FIFO — Firts In First Out («первый пришел — первый вышел»).

9) Список – это линейная цепочка элементов с последовательным доступом и вставкой / удалением элементов в любом месте списка. Стек – это доступ по схеме LIFO: "последним пришёл, первым ушёл", добавление и удаление элементов происходит с одного конца, а не в любом месте. Список – один из способов реализации стека в программе. Но это далеко не единственный и не самый эффективный способ.

10) Для вывода введённой последовательности наоборот, лучше использовать линейную структуру «Дек», так как в ней доступ к элементам осуществляется как с начала, так и с конца.

11) Сложность вставки в массив – O(n)

Сложность вставки в структуру – O(n)

12) Сложность удаления элемента из массива – O(n)

Сложность удаления элемента из структуры – O(n)

13) Трюк Вирта заключается в том, чтобы указатель на удаляемый элемент перенаправить на элемент, следующий за удаляемым.

14) Односвязный список состоит из узлов. Узел содержит значение и указатель на следующий узел.

15)

```
void outList(Tnode *L)
{
Tnode *node=L;
cout << "List";
while(node!=0 )
{
cout<< node->a<<" ";
node= node->next;
}
        cout << endl;
}
```

16)

```
void last_to_first(Node*& L)
{
Node* First = L;
Node* Last = nullptr;
while (First->next)
{
Last = First;
First = First->next;
}
Last->next = 0;
First->next = L;
L = First;
}
```

17) Лишнее действие является условный оператор с оператором если/иначе. Тогда код станет рабочим при условии, если вставлять не в пустой список. Новый узел будет вставляться в конец списка.

1.1 Постановка задачи

Реализовать программу решения задачи варианта по использованию линейного однонаправленного списка.

1. Информационная часть узла определена вариантом

2. Разработать функцию для создания исходного списка, используя функцию вставки нового узла перед первым узлом.

3. Разработать функцию вывода списка.

4. Разработать функции дополнительного задания варианта. При необходимости можно добавлять функции, декомпозируя задачу.

5. В основной программе выполните только тестирование каждой функции. Меню можно не создавать. Тесты обязательны.

6. Составить отчет по выполненному заданию (формат отчета после вариантов)

Личный вариант – 11.

Дан линейный однонаправленный список L, информационная часть которого содержит однозначные и двузначные числа.

1) Разработать функцию, которая создает массив A из 10 указателей на элемент списка и включает в список элемента массива с индексом i, числа списка L, которые начинаются с цифры равной i. Включение в конец списка. Однозначные числа включаются в список массива с индексом 0.

2) Разработать функцию, которая удаляет список L.

3) Разработать функцию, которая создает список L, включая в него списки массива A последовательно от списка с индексом 0 до списка с индексом 9.

1.2 Определение списка операций над списком, которые выявлены в процессе исследования задач дополнительного задания

1) void createArray(Node*& L) – функция создания массива с индексом i, числа списка L, которые начинаются с цифры равной i.

2) void createList(Node*& L, int n) – функция создания списка.

3) void outList(Node* L) – функция вывода списка.

4) void Delete(Node*& L) – функция удаления массива.

5) void NewList(Node *& L, int n) – функция создания нового списка для выполнения 3 пункта в задании.

1.3 Определение структуры узла однонаправленного списка

```
struct Node
{
int data; - однозначное/двузначное число.
Node* next; - указатель на последующий узел
};
```

1.4 Изобразить (рисунок) для каждой операции полученного списка процесс выполнения операции на существующем однонаправленном списке

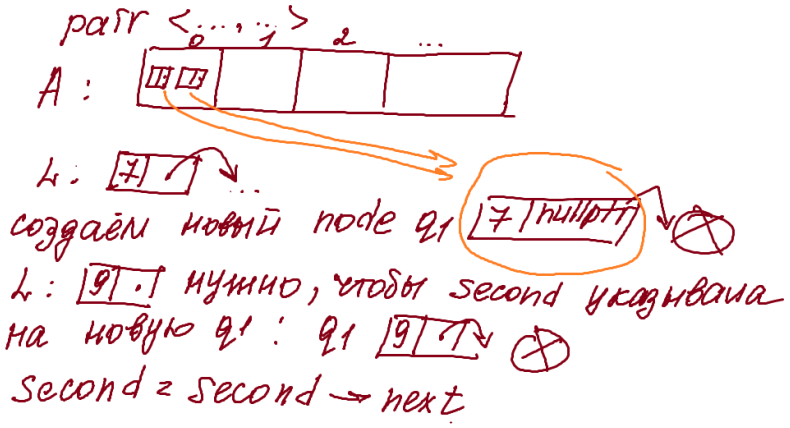

Рисунок 1 – Функция void createArray

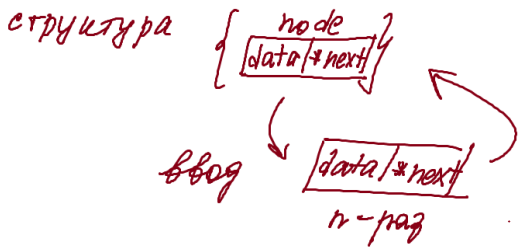

Рисунок 2 – Функция void createList

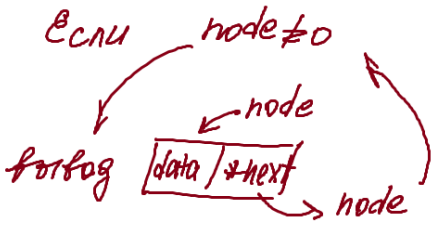

Рисунок 3 – Функция void outList

Рисунок 4 – Функция void Delete

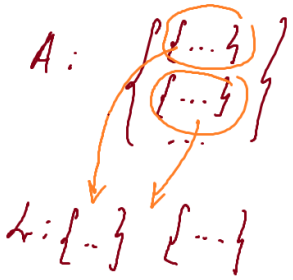

Рисунок 5 – Функция void NewList

1.5 Алгоритм операции создания списка(List)

1) Инициализация q, q1 = 0 – указателей на узлы

2) Ввод информации для каждого узла

3) Добавление в список узла

1.7 Код программы

```
#include <iostream>
using namespace std;

struct Node
{
	int data;
	Node* next; //след. узел.
};
Node* L;
pair <Node*, Node*> A[10];
Node* LL;

void createArray(Node*& L)
{
	for (int i = 0; i < 10; i++)
	{
		A[i] = { nullptr, nullptr };
	}

	Node* node = L;
	Node* q1; //temp.
	while (node != 0)
	{
		int Index = node->data / 10;
```

```
			q1 = new Node;
			q1->data = node->data;
			q1->next = nullptr;
			if (A[Index].first == nullptr)
			{

				A[Index] = { q1,q1 };
			}
			else
			{

				A[Index].second->next = q1;
				A[Index].second = A[Index].second->next;
			}
			node = node->next;

		}
}

void createList(Node*& L, int n)
{
		Node* q, * q1 = NULL;
		for (int i = 1; i <= n; i++)
		{
			q = new Node;
			cin >> q->data;
			q->next = NULL;
			if (L == NULL)
			{                                                    //список пуст
				L = q; q1 = L;
			}
			else
			{
				q1->next = q; q1 = q;
			}
		}
}

void outList(Node* L) {
		Node* node = L;
		cout << "List L: " << endl;
		while (node != 0)
		{                                                            //пока не
найдем NULL в последнем узле
			cout << node->data << " ";
			node = node->next;
		}
		cout << endl;
}



void Delete(Node*& L)
{
		L = NULL;
		cout << endl;
		cout << "Список удален!" << endl;
		cout << endl;
}
```

```
void NewList(Node*& L, int n)
{
	Node* q, * q1 = NULL;
	for (int i = 0; i < n; i++)
	{
		int fl = 0;
		Node* node = A[i].first;
		if (node != NULL)
		{
			fl = 1;
			q = new Node;
			q->data = node->data;
			q->next = NULL;
			if (L == NULL)
			{
				L = q; q1 = L;
			}
			else
			{
				q1->next = q; q1 = q;
			}
		}

		if (fl == 1)
		{
			while ((node->next != nullptr))
			{
				node = node->next;
				if (node != NULL)
				{
					q = new Node;
					q->data = node->data;
					q->next = NULL;
					if (L == NULL)
					{
						L = q; q1 = L;
					}
					else
					{
						q1->next = q; q1 = q;
					}
				}
			}
		}
	}
}

int main()
{
	setlocale(LC_ALL, "Russian");
	int n;
	cout << "Введите колличество узлов: ";
	cin >> n;
	cout << endl;
	cout << "Введите значения для узлов: ";
	createList(L, n);
	cout << endl;
	outList(L);
	cout << endl;
```

```
	createArray(L);

	cout << "List A: " << endl;

	for (int i = 0; i < 10; i++)
	{
		Node* node = A[i].first;
		cout << "№" << i << " ";
		while (node != 0)
		{
			cout << node->data << " ";
			node = node->next;
		}
		cout << endl;
	}
	Delete(L);
	NewList(L, 10);
	outList(L);
}
```

1.8 Результаты работы программы

Сначала пользователь вводит с клавиатуры количество желаемых узлов, а далее data – значения, которые будут храниться в этих узлах(node). Далее идет создание списка L, в котором будут находиться node, связанные указателями между собой. Далее создается список A, куда вносят значения из списка L, в соответствии с требованием задания. Происходит удаление списка L, и внесение значений из списка A в новый список L.

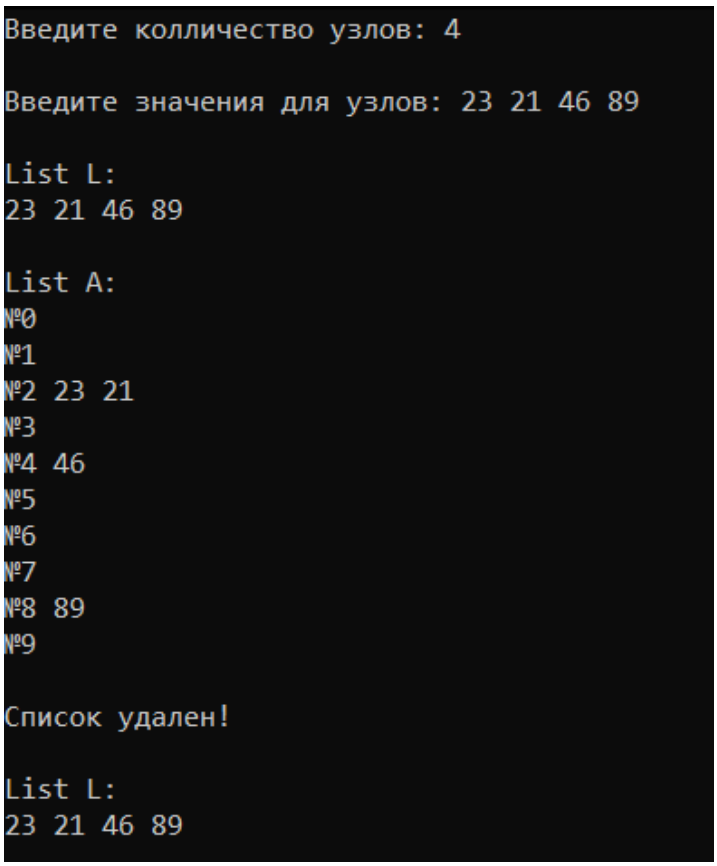

Рисунок 6 – Результат работы программы

2 ВЫВОДЫ

При работе над программой были получены и усвоены знания об однонаправленных списках.

Однонаправленные списки во многом похожи на массивы, но между ними есть существенная разница: при использовании массива элементы хранятся в памяти непрерывно, то есть рядом друг с другом, а при использовании однонаправленного динамического списка стоит учитывать, что элементы могут размещаться, где угодно в памяти. Основная идея списков — использовать ссылки: каждый элемент знает, где находится в памяти следующий за ним элемент. Этот вариант удобен тем, что при расширении не нужно перемещать массив полностью, достаточно лишь найти свободное место и сказать его предыдущему.

Списки работают медленнее с операциями доступа к данным, чем массивы, так как для этого необходимо дойти до конечного элемента, нужно пройти все элементы, но списки куда быстрее, когда нужно часто добавлять или удалять элементы в динамической структуре данных.

3 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных».
2. Habr: сайт. – URL: https://habr.com/ru/all/ (дата обращения: 06.03.2023).