Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

**ТЕСТИРОВАНИЕ**

По предмету “Основы алгоритмизации и программирования”

На тему: “Однонаправленные списки без заглавного звена”

Выполнил:

Студент 1 курса 9 группы

Павлович Ян Андреевич

Преподаватель: Белодед Н.И.

2024, Минск

Содержание

1. Цель
2. Анализ методов отображения действительных адресов элементов однонаправленного списка без заглавного звена в C++
   1. Метод вывода адресов с использованием указателя
   2. Использование стандартной функции ‘&’
   3. Отличие использования &(\*t).next от (\*t).next
3. Работоспособность программы
4. Заключение

**Цель**

Целью данного тестирования является рассмотрение методов отображения действительных адресов элементов однонаправленных списков без заглавного звена в языке программирования C++. Основное внимание уделено показу адресов, которые позволяют отслеживать переходы от одного элемента к другому в списке. В результате тестирования представлены различные способы вывода адресов элементов списка, а также анализ их преимуществ и недостатков с целью обеспечения лучшего понимания и эффективного использования при работе с динамическими структурами данных в C++.

Тестируемая программа - это алгоритм построения однонаправленного списка без заглавного звена с помощью схем "до и после".

Код тестируемой программы:

**#include<iostream>**

**using namespace std;**

**// Определение структуры узла списка**

**struct node**

**{**

**int value; // Значение элемента списка**

**node \*next; // Указатель на следующий элемент списка**

**};**

**void main ()**

**{**

**int i;**

**node \*phead, \*t; // Указатели на голову списка и текущий элемент списка**

**setlocale(LC\_ALL,"Rus"); // Установка локали для вывода на русском языке**

**phead = new (node); // Создание первого элемента списка и указание головы списка на него**

**t = phead; // Установка текущего элемента списка на голову списка**

**// Заполнение списка элементами**

**(\*t).value = 1;**

**(\*t).next = new (node);**

**t = (\*t).next;**

**(\*t).value = 2;**

**(\*t).next = new (node);**

**t = (\*t).next;**

**(\*t).value = 6;**

**(\*t).next = new (node);**

**t = (\*t).next;**

**(\*t).value = 17;**

**(\*t).next = new (node);**

**(\*t).next = NULL; // Установка указателя последнего элемента списка на NULL для обозначения конца списка**

**// Вывод содержимого информационных полей списка**

**for (t = phead; t != NULL; t = (\*t).next)**

**cout << (\*t).value << " ";**

**cout << "\n";**

**system("PAUSE"); // Приостановка выполнения программы перед ее завершением**

**}**

**Анализ методов отображения действительных адресов элементов однонаправленного списка без заглавного звена в C++**

В ходе разработки программного кода, особенно при работе с динамическими структурами данных, важно иметь возможность отслеживать адреса элементов для управления памятью и решения других задач. Одним из часто используемых методов является отображение действительных адресов элементов списка в C++. В данном тестировании представлен анализ методов реализации такого отображения.

**Метод вывода адресов с использованием указателя**

Простой и наиболее распространенный метод - это использование указателя для вывода адреса каждого элемента списка перед его значением. Этот метод представлен в следующем коде:

**// Вывод адреса и содержимого информационных полей списка**

**for (t = phead; t != NULL; t = (\*t).next)**

**{**

**// Вывод адреса текущего элемента списка и его значения**

**cout << "Адрес: " << t << ", Значение: " << (\*t).value << endl;**

**}**

В этом цикле осуществляется проход по всем элементам списка, начиная с головы (указатель **phead**). Каждый элемент списка представлен переменной **t**. Проверка **t != NULL** гарантирует, что цикл будет продолжаться до тех пор, пока не будет достигнут конец списка (т.е., пока указатель **t** не станет равным **NULL**).

**Использование стандартной функции ‘&’**

В C++ есть стандартная функция **&**, которая позволяет получать адрес переменной. Этот способ несколько компактнее, чем использование явного указателя:

**cout << "Адрес: " << &(\*t) << ", Значение: " << (\*t).value;**

Почему способ с использованием **&** более компактен? В контексте данной задачи, оператор **&** является более кратким способом получения адреса переменной. Вместо явного разыменования указателя и получения его адреса, можно просто применить оператор **&** к переменной. Этот подход сокращает количество операций и упрощает код, делая его более читаемым и понятным.

**if ((\*t).next != NULL)**

**{**

**cout << "Адрес указателя на следующий элемент: " << (\*t).next;**

**}**

При использовании &(\*t) мы действительно получаем адрес переменной, на которую указывает указатель t. Разберем, что происходит в этой строке:

1. (\*t) разыменовывает указатель t, чтобы получить объект, на который он указывает.
2. & берет адрес этого объекта.

Таким образом, &(\*t) возвращает адрес объекта, на который указывает t. В контексте вывода адреса текущего элемента списка это корректно, поскольку мы хотим увидеть адрес именно того объекта, который представляет текущий элемент списка.

Важно отличать это от ситуации, когда мы хотим получить адрес самого указателя, а не объекта, на который он указывает. В нашем случае, когда мы хотим получить адрес указателя на следующий элемент списка, использование &(\*t).next неверно, так как это приведет к получению адреса объекта, на который указывает (\*t).next, а не адреса самого указателя.

**Работоспособность программы**

Итоговые изменения для кода:

**// Проход по всем элементам списка, начиная с головы (phead) и перемещаясь к следующему элементу, пока не достигнут конец списка (t != NULL)**

**for (t = phead; t != NULL; t = (\*t).next) {**

**// Вывод значения текущего элемента списка**

**cout << (\*t).value << " ";**

**// Вывод адреса текущего элемента списка и его значения**

**cout << "Адрес текущего элемента: " << t << ", Значение: " << (\*t).value << endl;**

**// Вывод адреса объекта, на который указывает указатель, и его значения**

**cout << "Адрес: " << &(\*t) << ", Значение: " << (\*t).value << endl;**

**// Проверка, существует ли следующий элемент списка**

**if ((\*t).next != NULL) {**

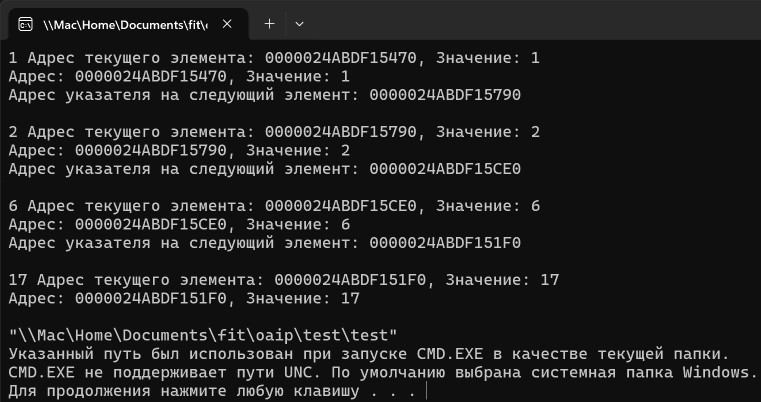
**// Вывод адреса указателя на следующий элемент списка**

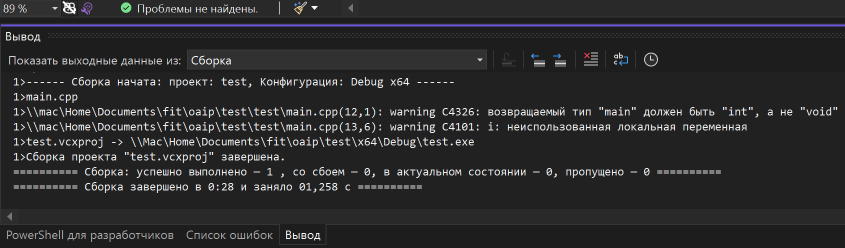
**cout << "Адрес указателя на следующий элемент: " << (\*t).next << endl << endl;**

**}**

**}**

Скришоты работоспособности программного кода:

****

****

**Заключение**

Внесенные изменения в код позволили нам более детально изучить работу с указателями и адресами элементов однонаправленного списка без заглавного звена в языке программирования C++. Используя цикл, мы прошлись по каждому элементу списка, выводя его значение, а также адрес текущего элемента и адрес объекта, на который указывает указатель.

Особое внимание было уделено выводу адреса указателя на следующий элемент списка. Мы обнаружили, что использование оператора & перед разыменованием указателя не всегда является необходимым, поскольку это может привести к получению адреса объекта, а не адреса указателя. Таким образом, правильным способом получения адреса указателя на следующий элемент списка оказалось использование просто (\*t).next, что позволило корректно отображать адреса переходов между элементами списка.

Эти изменения позволили нам улучшить понимание работы с указателями и адресами в языке C++, а также повысили ясность и читаемость кода.