**Міністерство освіти і науки України**

**Державний ВНЗ**

**“НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ”**

**ІНСТИТУТ ЕЛЕКТРОЕНЕРГЕТИКИ**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

*Кафедра автоматизації та комп’ютерних систем*

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

**курсового проекту**

з дисципліни *”Програмування”*

Виконавець,

студента гр. 123-17-1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ З.И.Божок

(підпис)

Керівник, с.в. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Надточий

(підпис)

**Дніпро**

**2018**

**ЗАВДАННЯ**

**на курсовий проект**

з дисципліни *”Програмування”*

студентки групи *123-17-1 Земляного Валерія Костянтиновича*

|  |  |  |
| --- | --- | --- |
| Розділ | Зміст завдання | Термін виконання |
| ***Визначення та використання***  ***динамічного списку*** | *Дати визначення та описати типи і*  *використання динамічного списку* | *29.09.2018р.* |
| ***Програмна організація динамічного списку*** | *Описати програмну організацію*  *динамічного списку* | *22.10.2018 р.* |
| ***Розробка програми*** | *Розробити програму з використанням*  *динамічного списку згідно із завданням.*  *Описати процес розробки програми* | *17.11.2018 р.* |
| ***Опис програми*** | *Описати програму* | *21.11.2018 р.* |

Завдання видав с.в. \_\_\_\_\_\_\_\_\_\_ В.В. Надточий

(підпис)

Завдання прийняла до виконання \_\_\_\_\_\_\_\_\_\_\_\_ З.И.Божок

(підпис)

Дата видачі завдання *01.09.2018р.*

Термін подання курсового проекту до захисту 21.11.2018р.

***ЗМІСТ***

[**Визначення та використання динамічного списку** 3](#_Toc531299809)

[**Опис організації динамічного списку** 5](#_Toc531299810)

[**Розробка програми** 6](#_Toc531299811)

[**Опис програми** 7](#_Toc531299812)

[**DoubleList.h** **Error! Bookmark not defined.**](#_Toc531299813)

[**main.cpp** **Error! Bookmark not defined.**](#_Toc531299814)

[**Висновки** 11](#_Toc531299815)

# **Визначення та використання динамічного списку**

**Динамічний список** - це деяка кількість компонентів (вузлів), що містять безпосередньо інформаційну частину (число, рядок або більш складні типи даних), а також посилання на наступний компонент (односпрямований список) та на наступний (двоспрямований список).

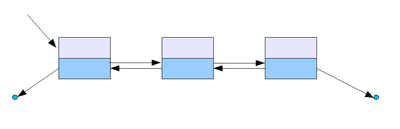
При розробці програм, що оперують з великою кількістю вхідних даних, виникає питання про зберігання цієї інформації під час виконання програми. Такий тип даних, як масив (що являє собою дані, послідовно розташовані у пам’яті) вирішує проблеми зберігання даних, однак, він не позбавлений недоліків. Головним з них, безумовно, є його фіксований розмір. Ця властивість не піддається зміні навіть у динамічно створених масивах, що досить часто змушує програмістів, використовуючих виключно їх, виділяти додаткову пам'ять аби її вистачило у майбутньому. Та навіть додатково виділена пам'ять обмежена, і ніхто не може дати гарантії, що її буде достатньо. З іншого ж боку, додаткової пам'яті може бути виділено забагато, а використано – набагато менше, ніж потрібно, що веде до даремного використання ресурсів комп’ютера.

Проблему фіксованого розміру вирішує інший тип даних, - динамічний список. До нього додаються і видаляються дані під час виконання програми. Розмір списку не фіксований – кількість інформації, яку можна зберігати в цьому типі даних, залежить виключно від розміру доступної пам'яті комп’ютера. Однак, за цю перевагу доводиться розплачуватися недоліком - якщо у випадку з масивом (який зберігає дані послідовно) ми маємо змогу отримати доступ до будь-якого елементу швидко і незалежно від розміру масиву, то в списку елементи (дані) розташовані в пам’яті майже завжди хаотично, а тому й час, потрібний на отримання доступу к елементу, залежить від розміру списку та позиції самого елементу у ньому.

Динамічний список використовується тоді, коли кількість елементів, які треба зберігати, заздалегідь невідома, або коли вставка та видалення елементів важливіша за доступ до конкретного елементу колекції. Прикладом використання динамічного списку є список робітників. У цьому випадку кількість робітників динамічно змінюється та заздалегідь не відома, а видалення та додавання – операції, які зазвичай виконуються над списком.

# **Опис організації динамічного списку**

В основі зв’язного списку лежить поняття вузла, або елемента (Node).   
Вузол – це контейнер, який дозволяє зберігати дані й отримувати наступний та/або попередній вузол.



Вузли динамічного списку

За допомогою головного вузла можна отримати доступ до будь-якого іншого вузла. Для цього потрібно перейти до наступного вузла, зберегти його як поточний, потім перейти до наступного за ним, і продовжувати цей процес допоки не дійдемо до потрібного вузла.

Останній вузол списку зазвичай в якості наступного має посилання на NULL (або nullptr). Таким чином, при лінійному обході списку буде відомо, коли потрібно зупинитися, а коли - продовжувати.

Вставка в кінець динамічного списку – процес додавання нового вузла наступним за останнім. Для цього потрібно лінійно пройти список, зупинитися на останньому вузлі, створити новий вузол, та зв’язати кінцевий вузол з тільки що створеним.

Видалення з кінця дещо нагадує операцію вставки в кінець, але, наприклад, в односпрямованому списку ми зупиняємося на передостанньому вузлі та видаляємо наступний за ним, тоді як в двоспрямованому можна дійти до останнього, зберегти попередній за ним, видалити кінець, та повідомити передостанньому, що наступного за ним вузла вже не існує.

## **Розробка програми**

Необхідно написати програму, що демонструє роботу з динамічним

списком.

У програмі слід застосувати функції, за допомогою яких можна створити список та відповідно до нього виконувати задані дії (згідно з варіантом):

– додавати елемент до списку (наприкінці списку);

– видаляти елемент зі списку (з кінця списку);

– видалити список;

– виводити весь список на екран.

Також необхідно виводити меню для вибору заданих операцій або для

завершення програми.

Проект програми повинен бути багатофайловим, тобто прототипи

функцій повинні бути у заголовному файлі, а визначення функцій – у файлі-джерелі.

При виконанні проекту студент одержує практичні навики з:

– використання і написання функцій;

– використання структур, покажчиків, масивів;

– проектування програмного забезпечення.

Завдання

|  |  |  |  |
| --- | --- | --- | --- |
| № вар. | Тип  динамічно-  го списку | Дані, що  зберігатимуть-  ся у списку | Дії, які необхідно виконувати із  динамічним списком |
| 13 | Двоспрямо-  ваний | Температура повтря | * додавати дані про температуру повітря; * визначити день з найменшою або найбільшою температурою; * вивести температуру і день її вимірювання за спаданням; * знайти дні з однаковою температурою; |

# **Опис програми**

**main.cpp**

#include<iostream>

#include<functional>

#include<random>

#include<time.h>

using namespace std;

#include "DoubleList.h"

#include <list>

struct Temperature

{

Temperature() : m\_day(0), m\_temp(0)

{}

Temperature(const int & day, const double &temp) :m\_day(day), m\_temp(temp)

{}

int m\_day;

double m\_temp;

friend ostream& operator<<(ostream& os, const Temperature& temp)

{

os << "Day : " << temp.m\_day << " Temp : " << temp.m\_temp;

return os;

}

};

void Help()

{

std::cout << "h - help" << std::endl

<< "a - add new temperature" << std::endl

<< "m - get max temperature" << std::endl

<< "s - get sorted by temperature descending " << std::endl

<< "q - get days with equal temp" << std::endl<<

"e - exit" << std::endl;

}

int main()

{

srand(time(NULL));

typedef Temperature temp\_t;

DoubleList<temp\_t> dl;

char ch = 0;

Help();

while (ch != 'q')

{

cout << "Enter your choice : ";

std::cin >> ch;

temp\_t temp;

switch (ch)

{

case 'h':

Help();

break;

case 'a':

cout << "Enter day : ";

cin >> temp.m\_day;

cout << "Enter temp : ";

cin >> temp.m\_temp;

dl.push\_back(temp);

dl.Sort(

[](const temp\_t& a, const temp\_t& b)->bool {return a.m\_temp > b.m\_temp; }

);

break;

case 'm':

std::cout << dl.GetTailVal() << std::endl;

break;

case 's':

std::cout << dl << std::endl;

break;

case 'q':

dl.ShowSame(

[](const temp\_t& a, const temp\_t& b)->bool {return a.m\_temp == b.m\_temp; }

);

break;

case 'e':

return 0;

break;

}

}

#ifdef \_DEBUG

system("pause");

#endif

return 0;

}

**DoubleList.h**

#pragma once

#ifndef DOUBLELIST\_H

#define DOUBLELIST\_H

#include<iostream>

#include<exception>

#include<functional>

template<class T>

class DoubleList

{

struct Node;

public:

DoubleList() : m\_head(nullptr), m\_tail(nullptr), m\_size(0)

{}

~DoubleList();

void push\_back(const T & val);

T pop\_back();

size\_t size() const { return m\_size; }

DoubleList(const DoubleList<T> &list);

void Sort(std::function<bool(const T&, const T&)> comparator);

void Swap(Node \*n1, Node \*n2);

void ShowSame(std::function<bool(const T&, const T&)> comparator)

{

if (!m\_size)

throw std::runtime\_error("Emtpy list");

Node \* showTmp = m\_head;

size\_t count = 0;

for (Node \* it = m\_head; it != m\_tail->m\_next; it = it->m\_next)

{

if (it->m\_next && comparator(it->m\_val,it->m\_next->m\_val))

{

count++;

}

else

{

if (count != 0)

for (showTmp; showTmp != it->m\_next; showTmp = showTmp->m\_next)

{

std::cout << showTmp->m\_val<<std::endl;

}

showTmp = it->m\_next;

count = 0;

}

}

}

friend ostream& operator<<(ostream& os, const DoubleList<T>& dt)

{

for (DoubleList<T>::Node \*it = dt.m\_head; it != dt.m\_tail->m\_next; it = it->m\_next)

{

os << it->m\_val << std::endl;

}

return os;

}

T GetTailVal()

{

return m\_tail->m\_val;

}

private:

struct Node

{

Node() : m\_prev(nullptr), m\_next(nullptr)

{}

Node(Node \* prev, Node \* next, const T & val) : m\_prev(prev), m\_next(next), m\_val(val)

{}

Node \* m\_prev;

Node \* m\_next;

T m\_val;

};

Node \* m\_head;

Node \* m\_tail;

size\_t m\_size;

};

template<class T>

void DoubleList<T>::push\_back(const T & val)

{

if (m\_head == nullptr)

{

m\_head = new Node(nullptr, nullptr, val);

m\_tail = m\_head;

}

else

{

DoubleList<T>::Node \* tmp = m\_tail;

m\_tail = new DoubleList<T>::Node(tmp, nullptr, val);

tmp->m\_next = m\_tail;

}

m\_size++;

}

template<class T>

T DoubleList<T>::pop\_back()

{

if (m\_tail == nullptr || m\_size == 0)

{

throw std::runtime\_error("Popping back from empty list");

}

T ret = m\_tail->m\_val;

Node \* tmp = m\_tail;

m\_tail = m\_tail->m\_prev;

delete tmp;

m\_size--;

return ret;

}

template<class T>

DoubleList<T>::~DoubleList()

{

while (m\_size)

this->pop\_back();

}

template<class T>

DoubleList<T>::DoubleList(const DoubleList<T> &list) : m\_head(nullptr), m\_tail(nullptr), m\_size(0)

{

if (!list.size())

throw std::runtime\_error("Empty list");

DoubleList<T>::Node \*itSrc = list.m\_head;

while (itSrc)

{

push\_back(itSrc->m\_val);

itSrc = itSrc->m\_next;

}

}

template<class T>

void DoubleList<T>::Swap(DoubleList<T>::Node \*n1, DoubleList<T>::Node \*n2)

{

if (n1 == n2 || n1 == nullptr || n2 == nullptr)

return;

if (n1->m\_prev == n2)

{

DoubleList<T>::Node \*tmp = n1;

n1 = n2;

n2 = tmp;

}

if (m\_head == n1)

{

m\_head = n2;

}

else if (m\_head == n2)

{

m\_head = n1;

}

if (m\_tail == n1)

{

m\_tail = n2;

}

else if (m\_tail == n2)

{

m\_tail = n1;

}

if (n1->m\_next == n2)

{

if (n1->m\_prev)

n1->m\_prev->m\_next = n2;

if (n2->m\_next)

n2->m\_next->m\_prev = n1;

n1->m\_next = n2->m\_next;

n2->m\_prev = n1->m\_prev;

n2->m\_next = n1;

n1->m\_prev = n2;

}

else

{

DoubleList<T>::Node \* tmpNext = n2->m\_next;

DoubleList<T>::Node \* tmpPrev = n2->m\_prev;

n2->m\_next->m\_prev = n1;

n2->m\_prev->m\_next = n1;

n2->m\_next = n1->m\_next;

n2->m\_prev = n1->m\_prev;

n1->m\_next->m\_prev = n2;

n1->m\_prev->m\_next = n2;

n1->m\_next = tmpNext;

n1->m\_prev = tmpPrev;

}

}

template<class T>

void DoubleList<T>::Sort(std::function<bool(const T&, const T&)> comparator)

{

for(int i=0; i<m\_size; i++)

{

for (DoubleList<T>::Node \*jt = m\_head; jt != m\_tail; jt = jt->m\_next)

{

if (comparator(jt->m\_val, jt->m\_next->m\_val))

{

Swap(jt, jt->m\_next);

jt = jt->m\_prev;

}

}

}

return;

}

#endif

# **Висновки**

Дізналася про таку структуру даних, як динамічний список, його організацію в ЕОМ, переваги та недоліки його викорисання та навчилася реалізовувати власний односпрямований список за допомогою покажчиків і допоміжних структур даних. Отримала корисні навички в процесі проектування програмного забезпечення, а саме – проектування окремих модулів програми та поєднання їх в одне ціле. Розширила свої знання з ООП при створенні власних класів для опису об’єктів реального світу.