**ПРАКТИЧНЕ ЗАВДАННЯ № 11**

**Завдання №1 :**

Создайте манипулятор для вывода чисел в научной нотации с символом Е в верхнем регистре.

**Код програми:**

#include <iostream>

using namespace std;

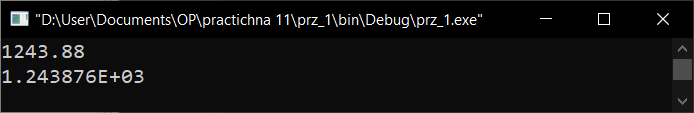
int main() {

cout << 1243.875764345232 << endl << scientific << uppercase << 1243.875764345232;

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №2 :**

Напишите программу для копирования текстового файла. В процессе копирования преобразуйте каждый символ табуляции в соответствующее число пробелов.

**Код програми:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

void change(ifstream &in) {

string str, output;

while(!in.eof()) {

getline(in, str);

cout << "original: " << str << endl;

for (int i = 0; i < str.length(); ++i) {

if (str[i] == '\t')

str[i] = ' ';

}

output.assign(str);

}

in.close();

ofstream out("copy.txt");

out << output;

out.close();

ifstream in\_changed("copy.txt");

while(in\_changed.is\_open()) {

while(getline(in\_changed, output))

cout << "changed: " << output;

in\_changed.close();

}

}

int main() {

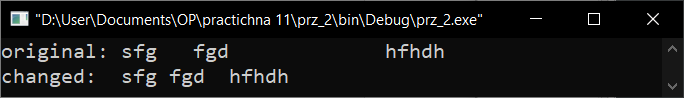
ifstream in("original.txt");

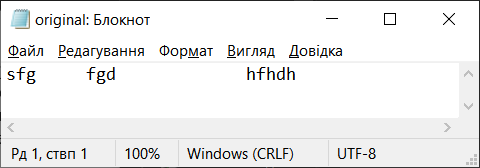
change(in);

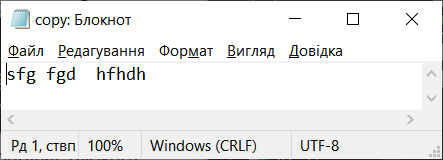
return 0;

}

**Результати роботи програми (скриншоти)**







**Завдання №3 :**

Напишите программу для поиска в текстовом файле слова, заданного в командной строке. После выполнения программы на экране должно появиться число, обозначающее, сколько раз данное слово найдено в файле. Для простоты считайте следующее: все, что с обеих сторон окружено пробелами, есть слово.

**Код програми:**

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

int main(int argc, char \*argv[]) {

char str[100] = "\0"; int k = 0;

if (argc != 2)

return 1;

ifstream in("text.txt");

while (!in.eof()) {

for(int i = 0; i < 100; i++)

in.get(str[i]);

}

cout << str << endl;

char \*del = strtok (str, " ");

while (del != NULL) {

del = strtok (NULL, argv[1]);

k++;

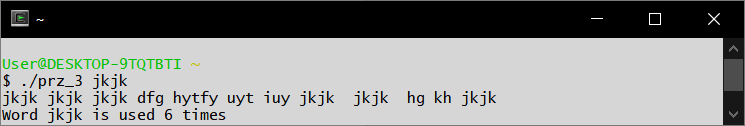
}

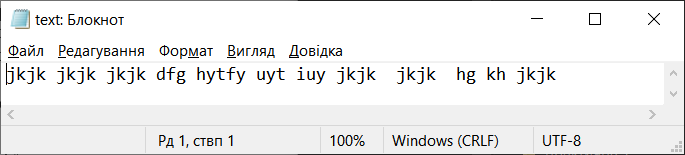
cout << "Word " << argv[1] << " is used " << k << " times";

return 0;

}

**Результати роботи програми (скриншоти)**





**Завдання №4 :**

Напишите инструкцию, которая устанавливает указатель записи на 81-й байт в файле, связанном с потоком out.

**Код програми:**

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream file\_out("OK.txt");

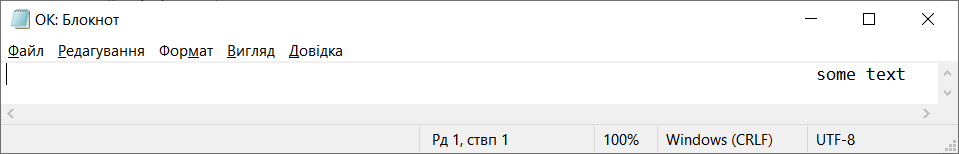
file\_out.seekp(81, ios::beg);

file\_out << "some text";

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №5 :**

Напишите программу создания базового класса Num. В этом классе должно храниться целое и определяться виртуальная функция shownum(). Создайте два производных класса outhex и outoct, которые наследуют класс Num. Функция shownum() должна быть переопределена в производных классах так, чтобы осуществлять вывод на экран значений, в шестнадцатеричной и восьмеричной системах счисления соответственно.

**Код програми:**

#include <iostream>

using namespace std;

class Num {

public:

int value;

Num (int a) {

value = a;

}

virtual void shownum() {

cout << "Value in dec: " << value << endl;

}

};

class Outhex : Num {

public:

Outhex(int a) : Num(a) { }

void shownum() {

cout << "Value in hex: " << hex << value << endl;

}

};

class Outoct : Num {

public:

Outoct(int a) : Num(a) { }

void shownum() {

cout << "Value in oct: " << oct << value << endl;

}

};

int main() {

Num obj(25);

Outhex obj1(25);

Outoct obj2(25);

obj.shownum();

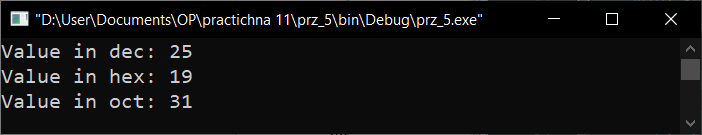
obj1.shownum();

obj2.shownum();

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №6 :**

Напишите программу, в которой базовый класс Dist используется для хранения в переменной типа double расстояния между двумя точками. В классе Dist создайте виртуальную функцию trav\_time(), которая выводит на экран время, необходимое для прохождения этого расстояния с учетом того, что расстояние задано в милях, а скорость равна 60 миль в час. В производном классе Metric переопределите функцию trav\_time() так, чтобы она выводила на экран время, необходимое для прохождения этого расстояния, считая теперь, что расстояние задано в километрах, а скорость равна 100 километров в час.

**Код програми:**

#include <iostream>

#include <iomanip>

using namespace std;

class Dist {

public:

double distance;

Dist(double a) {

distance = a;

}

virtual void trive\_time() {

cout << "Distance in miles: " << setprecision(8) << distance <<

", needed time with speed 60 miles/hour: " << distance/60 << endl;

}

};

class Metric : Dist {

public:

Metric(double a) : Dist(a) { }

void trive\_time() {

cout << "Distance in km " << setprecision(8) << distance <<

", needed time with speed 100 km/hour: " << distance/100 << endl;

}

};

int main() {

double a;

cin >> a;

Dist obj1(a);

Metric obj2(a);

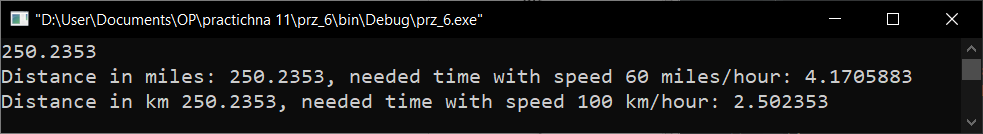
obj1.trive\_time();

obj2.trive\_time();

return 0;

}

**Результати роботи програми (скриншоти)**



**Завдання №7 :**

Проведите эксперимент с двумя программами из Example 11.4 и Example 11.5. Попытайтесь создать объект, используя класс area из Example 11.4, и проанализируйте сообщение об ошибке.

В Example 11.5 попытайтесь удалить переопределение функции func() внутри класса derived2. Убедитесь, что тогда действительно будет использоваться та версия функции func(), переопределение которой находится в классе derived1. Что произойдет в Example 11.5 при удалении переопределения функции func() из класса derived1? Будет ли при этом программа компилироваться и запускаться? Если да, то почему?

**A)**

**Код програми:**

#include <iostream>

using namespace std;

class Area {

double dim1, dim2;

public:

void set\_area(double d1, double d2) {

dim1 = d1;

dim2 = d2;

}

void get\_dimension(double &d1, double &d2) {

d1 = dim1;

d2 = dim2;

}

virtual double get\_area() = 0;

};

class Reactangle : public Area {

public:

double get\_area() {

double d1, d2;

get\_dimension(d1, d2);

return d1\*d2;

}

};

class Triangle : public Area {

public:

double get\_area() {

double d1, d2;

get\_dimension(d1, d2);

return 0.5 \* d1 \* d2;

}

};

int main() {

Area \*p;

//Area a;

Reactangle r;

Triangle t;

//a.set\_area(34. 23.2);

r.set\_area(3.3, 4.5);

t.set\_area(4.0, 5.0);

p = &r;

cout << "Reactangle has area: " << p->get\_area() << endl;

p = &t;

cout << "Triangle has area: " << p->get\_area() << endl;

return 0;

}

/\*Функція *get\_area()* в класі *Area* є *pure function* і не виконує ніяких дій, тому створити об’єк класу *Area* неможливо\*/

**Результати роботи програми (скриншоти)**

****

**B)**

**Код програми:**

#include <iostream >

using namespace std;

class Base {

public:

virtual void func() {

cout << "Using base version of func()\n";

}

};

class Derived1 : public Base { };

class Derived2 : public Derived1 {

public:

void func() {

cout << "Using Derived2 verssion of func\n";

}

};

int main() {

Base \*p;

Base obj;

Derived1 objd1;

Derived2 objd2;

p = &obj;

p->func();

p = &objd1;

p->func();

p = &objd2;

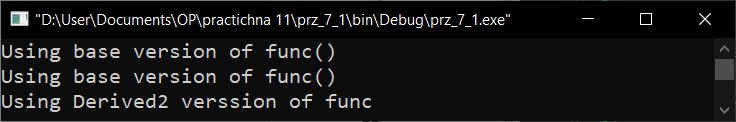
p->func();

return 0;

}

/\*Програма компілюється та запускається. Взяття адресу об’єкта потім допомагає визначити версію функції, яка буде використовуватись при виклику. Оскільки в класі *Base* функція *func()* визначена, а клас *Derived1* дочірний від *Base,* то при відсутності перезавантаженої версії в дочіному буде використовуватись функція з базового класу. \*/

**Результати роботи програми (скриншоти)**



**Завдання №8 :**

Программа из Example 11.6 при компиляции при помощи команды g++ -Wall ex116.cpp выдает предупреждения. Объясните причину появления этих предупреждений и для их устранения внесите необходимые правки в код данного примера. Совет 1: Функция **main()** в программе со списками (см. Example 11.6 ) только иллюстрирует работу классов. Для изучения динамического полиморфизма попробуйте использовать в этой программе следующую функцию **main()**:

**Код програми:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#define SIZE 10

using namespace std;

class List {

public:

List \*head, \*tail, \*next;

int num;

List () {

head = tail = next = NULL;

}

virtual void store(int i) = 0;

virtual int retrieve() = 0;

virtual ~List() {

List \*temp;

while (head) {

head = temp;

temp = head->next;

delete head;

}

}

virtual void show() = 0;

virtual void create() = 0;

};

class Queue : public List {

public:

void store(int i) {

List \*item = new Queue;

if(!item)

exit(1);

item->num = i;

if(tail)

tail->next = item;

tail = item;

item->next = NULL;

if(!head)

head = tail;

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

void create() {

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

store(i);

}

void show() {

cout << "Queue" << endl;

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

cout << retrieve() << ' ';

cout << endl;

}

};

class Stack: public List {

public:

void store(int i) {

List \*item = new Stack;

if(!item)

exit(1);

item->num = i;

if(head)

item->next = head;

head = item;

if(!tail)

tail = head;

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

void create() {

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

store(i);

}

void show() {

cout << "Stack" << endl;

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

cout << retrieve() << ' ';

cout << endl;

}

};

class Sorted : public List {

public:

void store(int i) {

List \*temp = head;

List \*temp2 = new Sorted;

if(head == NULL) {

head = new Sorted;

head->num = i;

head->next = NULL;

} else {

if(i < temp->num) {

temp2 = new Sorted;

temp2->num = i;

temp2->next = head;

head = temp2;

} else {

for( ; temp->next != NULL && !(i < temp->next->num && i >= temp->num); temp=temp->next)

continue;

if(temp->next == NULL) {

temp->next = new Sorted;

temp = temp->next;

temp->num = i;

temp->next = NULL;

} else {

temp2 = new Sorted;

temp2->num = i;

temp2->next = temp->next;

temp->next = temp2;

}

}

}

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

void create() {

int a;

for (int i = 0; i < SIZE; ++i) {

cin >> a;

store(a);

}

}

void show() {

cout << "Sorted" << endl;

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

cout << retrieve() << ' ';

cout << endl;

}

};

int main() {

srand(time(NULL));

List \*ptr;

Queue q\_obj;

Stack s\_obj;

Sorted obj;

ptr = &s\_obj;

ptr->create();

ptr->show();

ptr = &q\_obj;

ptr->create();

ptr->show();

ptr = &obj;

ptr->create();

ptr->show();

return 0;

}

/\*Попередження про відсутність віртуального деструктора.

Оскільки в нас створюється вказівник \*ptr типу базового класу і використовується для доступу до функцій дочірних класів, то при його видаленні компілятор підшукує перезавантажені версії деструкторів, але деструктор класу List був не віртуальним, то він викликався лише для класу List, але не для дочірних класів.

Вирішення – зробити деструктор класу List віртуальним:

*virtual ~List() {*

*List \*temp;*

*while (head) {*

*head = temp;*

*temp = head->next;*

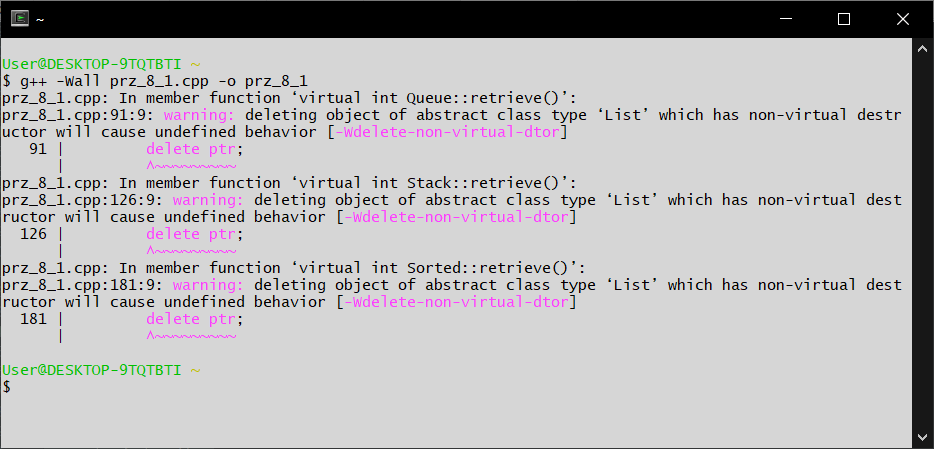
*delete head;*

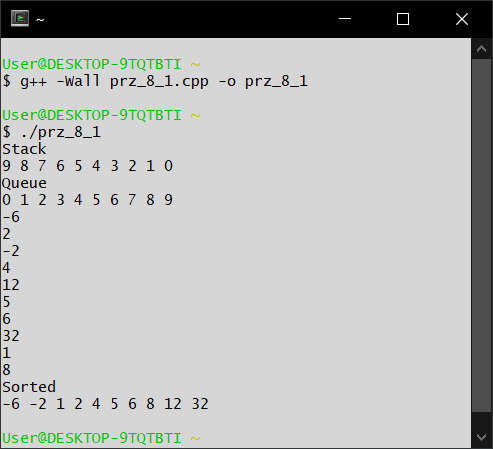
*}*

}

\*/

**Результати роботи програми (скриншоти)**





**Завдання №9 :**

Расширьте пример со списком, (см. Example 11.6 ) так, чтобы в нем перегружались операторы + и -. Используйте оператор + для внесения элемента в список, а оператор - для выборки элемента из списка.

**Код програми:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#define SIZE 10

using namespace std;

class List {

public:

List \*head, \*tail, \*next;

int num;

List () {

head = tail = next = NULL;

}

virtual void store(int i) = 0;

virtual int retrieve() = 0;

};

class Queue : public List {

public:

void store(int i) {

List \*item = new Queue;

if(!item)

exit(1);

item->num = i;

if(tail)

tail->next = item;

tail = item;

item->next = NULL;

if(!head)

head = tail;

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

Queue operator + (int i) {

store(i);

return \*this;

}

int operator-(int unused) {

return retrieve();

}

};

class Stack: public List {

public:

void store(int i) {

List \*item = new Stack;

if(!item)

exit(1);

item->num = i;

if(head)

item->next = head;

head = item;

if(!tail)

tail = head;

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

Stack operator+(int i) {

store(i);

return \*this;

}

int operator-(int unused) {

return retrieve();

}

};

class Sorted : public List {

public:

void store(int i) {

List \*temp = head;

List \*temp2 = new Sorted;

if(head == NULL) {

head = new Sorted;

head->num = i;

head->next = NULL;

return;

}

if(i < temp->num) {

temp2 = new Sorted;

temp2->num = i;

temp2->next = head;

head = temp2;

}

else {

for( ; temp->next != NULL && !(i < temp->next->num && i >= temp->num); temp=temp->next)

continue;

if(temp->next == NULL) {

temp->next = new Sorted;

temp = temp->next;

temp->num = i;

temp->next = NULL;

} else {

temp2 = new Sorted;

temp2->num = i;

temp2->next = temp->next;

temp->next = temp2;

}

}

}

int retrieve() {

List \*ptr;

if(!head)

return 0;

int i = head->num;

ptr = head;

head = head->next;

delete ptr;

return i;

}

Sorted operator + (int i) {

store(i);

return \*this;

}

int operator-(int unused) {

return retrieve();

}

};

int main() {

srand(time(NULL));

Queue q\_obj; Stack s\_obj; Sorted obj; int a;

cout << "Queue" << endl;

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

q\_obj + i;

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

cout << q\_obj - 0 << ' ';

cout << endl;

cout << "Stack" << endl;

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

s\_obj + i;

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

cout << s\_obj - 0 << ' ';

cout << endl;

cout << "Sorted list" << endl;

for(int i = 0; i < SIZE; ++i) {

cin >> a;

obj + a;

}

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

cout << obj - 0 << ' ';

cout << endl;

return 0;

}

**Результати роботи програми (скриншоти):**

