МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з курсу
«Технології програмування в задачах теплофізики»
для студентів спеціальностей 8.05060102 «Теплофізика»
та 8.05060402 «Турбіни»

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 25.06.2015

Харків
НТУ «ХПІ»
2015

Методичні вказівки до лабораторних робіт з курсу «Технології програмування в задачах теплофізики» для студентів спеціальностей 8.05060102 «Теплофізика» та 8.05060402 «Турбіни» / уклад. В.М. Савченко, О.О. Литвиненко, Н.С. Шаповалова. – Харків: НТУ «ХПІ», 2015. – 44 с.

Укладачі: В. М. Савченко
О. О. Литвиненко
Н. С. Шаповалова

Рецензент О. І. Тарасов

Кафедра турбінобудування

ВСТУП

Лабораторні заняття з курсу «Технології програмування в задачах теплофізики» мають за мету навчити студентів основам розробки програмного забезпечення для розв'язання актуальних задач моделювання теплофізичних процесів та проектування енергетичних установок [1–4].

Студенти мають уявляти, що таке методи аналізу, проектування, реалізації та тестування програмних продуктів, навчитися орієнтуватися в існуючих підходах і технологіях та використовувати їх на практиці. Під час лабораторного практикуму студенти вчаться використовувати технології структурного, об'єктно-орієнтованого (ООП) та узагальненого програмування для розв'язання задача в галузі теплофізики [1, 2, 5–11].

Лабораторні роботи розподілено за темами, що пов'язані з послідовністю викладення теоретичного матеріалу з курсу. В процесі виконання лабораторних робіт студенти отримують навички використання уніфікованої мови моделювання (*UML)* та елементів об'єктно-орієнтованого дизайну на прикладах розробки класів реальних елементів теплових схем [1, 5].

Лабораторні роботи виконуються впродовж десятого семестру, програми розробляються мовою програмування *C++* (надалі, *C++*). Кожна робота містить необхідний теоретичний мінімум та приклади програм.

По кожній лабораторній роботі студент виконує індивідуальне завдання на персональному комп'ютері з оформленням звіту та його подальшою здачею викладачу. Індивідуальне завдання обирається згідно з номером прізвища у журналі.

Рекомендується використовувати ресурси Internet [10–11] та сучасні безкоштовні компілятори та середовища розробника на мові *C++*, такі як gcc [12], Code::Blocks [13], Visual Studio Express [14] та ін.

ЗАГАЛЬНА ПОСТАНОВКА ЗАДАЧІ

Принципова теплова схема паротурбінної установки (ПТУ) є сукупністю елементів пароводяного тракту з технологічними зв'язками між ними. Її елементами є: турбіна, конденсатор, конденсатний і живильний насоси, ежектор, підігрівачі низького і високого тиску, мережеві і сальникові підігрівачі, деаератор, котел та ін., які обираються відповідно до технологічних вимог до проектованої ТЕС(рис. 1).

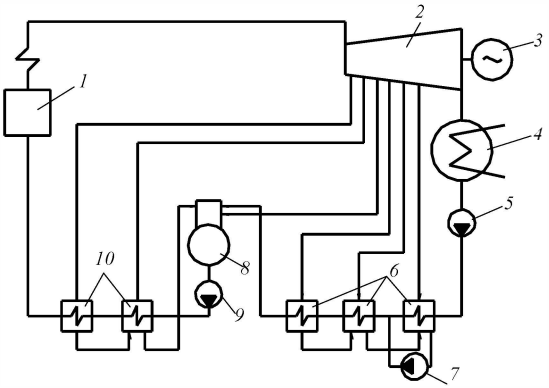


Рисунок 1 – Принципова теплова схема ПТУ:
1 – котел; *2* – турбіна; *3* – генератор; *4* – конденсатор; *5* – конденсатний насос;
6 – підігрівачі низького тиску; *7* – дренажний насос; *8* – деаератор;
9 – живильний насос; *10* – підігрівачі високого тиску

В залежності від типу турбоустановки додають необхідні елементи з різними конструктивними особливостями. Для теплового розрахунку необхідно встановити зв'язки між елементами, знайти теплові баланси кожного елемента та знати окремі конструктивні характеристики. Особливо це стосується підігрівачів високого та низького тиску. Також необхідно враховувати наявність охолоджувачів пари і дренажу та схеми їх включення в лінію основного конденсату. Елементи теплової схеми зручно представити класами об'єктів для подальшого використання та реалізації засобами об'єктно-орієнтованої мови програмування.

З відомих безкоштовних професійних систем теплофізичних розрахунків виділимо програму DNA (Brian Elmegaard, Данія), написану на Фортрані 77/90. Незважаючи на можливість її розширення, вона є складною, в її бібліотеці елементів відсутні поширені елементи теплових схеми ПТУ, тому задача розробки програмного забезпечення подібного до DNA є актуальною.

Лабораторна робота 1

РОЗРОБКА КЛАСІВ ЕЛЕМЕНТІВ ТЕПЛОВИХ СХЕМ

Метою є ознайомлення з основними поняттями об'єктно-орієнтованого програмування, розробка класів елементів теплових схем, отримання навичок роботи з об'єктами введення-виведення *C++*.

Завдання

Варіанти завдання обираються із додатка А.

1. Згідно з варіантом розробити клас для опису в програмі елементу теплової схеми: а) як елемента розрахункової схеми; б) як конструктивного елемента.

2. Розробити консольну програму з використанням форматування та об'єктів введення-виведення *C++*.

3. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

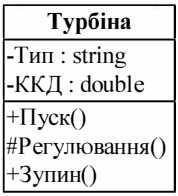
Методичні вказівки до виконання завдання

Об'єкт – це деяка фізична або абстрактна сутність, яка чітко ідентифікується, наприклад, турбінна установка, турбіна, теплова схема, конденсатор, підігрівач, тощо. Кожний об'єкт – це екземпляр деякого класу об'єктів або просто класу. *Клас* – це абстрактне поняття, яке характеризує споріднені об'єкти. Турбіна, конденсатор – приклади класів, тоді як турбіна К300-240 – приклад об'єкту.

Об'єкти характеризуються *атрибутами*. Атрибутами елементів теплових схем є їх характеристики, такі як величина поверхні теплообміну, питомі втрати, коефіцієнт корисної дії, тиск пари, ентальпія та ін. Ті ж самі елементи з точки зору конструкторських розрахунків характеризуються конструктивними параметрами – формою, геометричними розмірами, типами теплоносія тощо. Крім атрибутів, об'єкти характеризуються поведінкою, яку в ООП називають *операціями* або *методами*. Так, турбінна установка перетворює потенційну енергію носія в кінетичну енергію обертання, автомобіль рухається, комп'ютер обчислює. Таким чином, об'єкт складається з атрибутів та методів, приховуючи від інших об'єктів, що взаємодіють з ним і використовують його функціональність, свою реалізацію.

Для представлення класу уніфікованою мовою моделювання використовують прямокутник, розділений на три частини. У верхній частині міститься ім'я класу (іменник, записаний з великої літери), в середній – атрибути, в нижній – методи (рис. 1.1). Позначки «-» (закритий (*private*)), «+» (відкритий

(*public*)), «#» (захищений (*protected*)), які зображено перед іменем атрибуту або назвою методу, означають рівень доступу до відповідних компонентів об'єкту ззовні.



а

```
class Turbine {
private:
    std::string m_type;  // тип
    double m_efficiency; // ККД
public:
    void start(); // пуск
    void stop();  // зупин
protected:
    void tuning();// регулювання
};
```

б

Рисунок 1.1 – Опис класу:
а – діаграма *UML*; *б* – код *C++*

Основними парадигмами об'єктно-орієнтованого програмування є інкапсуляція, успадкування і поліморфізм.

Інкапсуляція – приховування внутрішньої структури об'єкту від зовнішнього світу.

Успадкування – спосіб визначення нового класу на основі того, що вже існує. При цьому успадковуються властивості і методи класу, що існує, та додаються нові, модифікуючи або розширюючи його.

Поліморфізм (*ad hoc* поліморфізм) – це здатність об'єктів різних класів, зв'язаних ієрархією успадкування, реагувати по-різному при зверненні до однієї й тієї ж функції-елемента.

Для розробки об'єктно-орієнтованих програм будемо використовувати мову програмування *C++*[5, 8].

Потоки введення-виведення в C++ реалізовано за допомогою визначених в файлі *iostream* класів *istream* (введення), *ostream* (виведення) та стандартних об'єктів: *cin* (введення), *cout* (виведення), *cerr* (помилки, виведення), *clog* (запис станів у програмі, виведення). Для роботи із потоками використовують операції «Помістити в потік» (<<) та «Взяти з потоку» (>>), наприклад, *cin>>a>>b>>c*; *cout <<"rez*: "<<*a*<<", "<<*b*<<","<<*c*;

Для форматування тексту, що виводиться на екран, використовують маніпулятори (файл *iomanip*) або відповідні властивості об'єктів введення-виведення (табл. 1.1). Для зміни формату відображення за допомогою методів *setf*(), *unsetf*() використовують пари значень «формат, властивість», наприклад, *cout.setf*(*ios*::*scientific*, *ios*::*floatfield*).

Таблиця 1.1 – Маніпулятори та методи потоків

Маніпулятор	Метод
Ширина поля під значення	
setw(n)	*width(n)*
Символ-заповнювач	
setfill('c')	*fill('c')*
Точність (кількість розряді після крапки)	
setprecision(*n*)	*precision*(*n*)
Формат відображення даних	
scientific, *fixed*, *dec*, *oct* , *hex*, *left*, *right*	*setf* (), *unsetf*() *adjustfield* (вирівнювання), *basefield* (система числення), *floatfield* (формат чисел з рухомою крапкою)

Приклад.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() { // таблиця
    int time1[] = { 4,48,2937,123011,221073 };
    int time2[] = { 8,16,293,12301,22107 };
    cout<<"size"<<setw(20)<<"time"<<setw(20)<<"time2\n";
    cout<<setfill('-')<<setw(60)<<"-"<<endl;
    int run = 10;
    cout<<setfill('.');  //символ-заповнювач "пробіл"
    for (int i = 0; i < 5; i++) {
        cout<<setw(20)<<left<<run;          run *= 10;
        cout<<setw(20)<<left<<time1[i];
        cout<<setw(20)<<left<<time2[i]<<"\n";
    }  // різні маніпулятори
    double a = 0.2;double b = 2.0;double c = 1234567890.0;
    cout<<"1. "<<a<<", "<<b<<", "<<c<<endl;
    cout<<"2. "<<fixed<<a<<", "<<b<<", "<<c<<endl;
    cout<<"3. "<<scientific<<a<<", "<<b<<", "<<c<<endl;
    cout<<"4. "<<fixed<<setprecision(3)<<a<<", "<<b<<", "
        <<c <<endl;
    cout<<"5. "<<setprecision(20)<<a<<endl;
    cout<<"6. "<<setw(8)<<setfill('*')<<32<<98<<endl;
    cout<<"7. "<<setw(8)<<34<<setw(8)<<45<<endl;
    cout<<"8. "<<setfill(' ') <<setw(8)<<hex<<54 <<oct<<54 <<
              <<dec<<setw(16)<<right<<54<<endl;
}
```

Порядок виконання роботи

1. Розробити клас, створити його діаграму *UML* та написати відповідний код мовою *C++*.

2. Розробити програму табуляції значень функції з використанням потоків введення-виведення. Оформити вивід на екран у вигляді таблиці.

Питання для самоперевірки

1. Як використовують абстрагування даних у програмуванні?
2. Дайте наступні визначення: «Об'єкт»,«Клас», «Діаграма *UML»*.
3. Основні принципи об'єктно-орієнтованого програмування?
4. Дайте визначення: «Інкапсуляція», «Успадкування», «Поліморфізм».
5. Які переваги та недоліки об'єктно-орієнтованого підходу до розробки програм?

Лабораторна робота 2

РОЗРОБКА ПРОГРАМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ ТЕПЛОФІЗИКИ

Метою є розробка програми для розв'язання задачі теплофізики з використанням простих класів, конструкторів, деструкторів, статичних типів даних та контролем помилок (виняткових ситуацій).

Завдання

Варіанти завдання обираються із додатка Б.

1. Згідно з варіантом розробити програму мовою *C++* з використанням технології ООП.

2. Додати до програми клас для обробки виняткових ситуацій.

3. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

Розглянемо приклад програми для розв'язання квадратного рівняння, розроблену у відповідності до принципів ООП. Нижче наведено визначення об'єкта SquareEq, яке складаються з ключового слова *class,* імені класу та його «тіла» у фігурних дужках Реалізацію класу наведено в додатку В.

```
#ifndef SQAUREEQ_H_
#define SQAUREEQ_H_
class SquareEq {
    double m_a, m_b, m_c; // коефіцієнти
    double m_x1, m_x2;    // розв'язки
```

```
    double discrim();     //дискримінант
public:
    SquareEq();            // конструктор
    SquareEq(double a, double b, double c); // конструктор
    void set_a(double a);  // сеттер
    double get_x1() const; // геттер
    double get_x2() const; // геттер
    void solve();
}; // обов'язковий елемент визначення класу
#endif /* SQAUREEQ_H_ */
```

Конструктор – це функція без типу, ім'я якої співпадає з іменем класу. Конструктор використовують для створення об'єкту. Якщо конструктор класу не визначено, то використовується конструктор за умовчанням. Як правило, конструктор приймає параметри та здійснює ініціалізацію атрибутів класу через список ініціалізації, який розташовується після «:»

```
SqaureEq::SquareEq(double a,double b,double c) : m_a(a),
                    m_b(b) , m_c(c), m_x1(0), m_x2(0){ }.
```

Це дозволяє встановити коректні значення даним-членам класу до його створення. Параметри можуть мати значення за умовчанням (наприклад, *double a*=0), такі параметри повинні бути останніми в списку параметрів. Атрибути класу ініціалізуються згідно з порядком визначення, а не порядком слідування в списку параметрів.

Оператор розширення області дії «::» використовують для позначення приналежності атрибуту або методу до відповідного класу або простору імен (*namespace*).

Деструктор – це функція без типу та параметрів, ім'я якої співпадає з іменем класу, якому передує «~». Деструктор відповідає за звільнення ресурсів, які займав клас.

Для доступу до атрибутів класу іноді використовують спеціальні функції – гетери (*getter)* та сетери (*setter)*. Перші повертають значення, тому мають тип відповідно до типу атрибуту класу. Другі – встановлюють значення, тому приймають параметр та присвоюють його відповідному атрибуту класу. Якщо функція не змінює внутрішній стан об'єкту, то після її імені вказують специфікатор const.

В *C++* для контролю помилок та виключних ситуацій використовують механізм виключень (*exception*). Для його застосування у місці програми, де можливі виключні ситуації, генерують виключення (*throw*). Для перехоплення виключень фрагмент коду розміщують в блоці

```
try{
   /*код,  що може генерувати виключення*/
}
catch (/*перехоплення*/) {
    /*обробка*/
}
```

Зазвичай створюють власні оброблювачі виключень або використовують стандартні, визначені в файлі *stdexcept*.

Порядок виконання роботи

1. Виконати тестовий приклад (додаток В), додати методи, що дозволяють вводити з клавіатури та виводити на екран значення відповідних атрибутів класу.
2. Розробити клас, що призначений для розв'язання задачі теплофізики (додаток Б).
3. Додати у програму власний клас для обробки виключних ситуацій.

Питання для самоперевірки

1. Як визначити клас мовою *C++*?
2. Де може бути визначена (описана) функція класу?
3. Порівняйте виклик члена-функції класу у функціях класу та поза функціями цього класу.
4. Назвіть специфікатори доступу до членів класу. Чи може з'являтися у визначенні класу один і той же специфікатор доступу багато разів?
5. Який рівень доступу за замовчуванням для атрибутів класу?
6. Які функції в програмі мають доступ до закритих та відкритих атрибутів класу?

Лабораторна робота 3

ДИНАМІЧНІ МАСИВИ ДЛЯ ЗБЕРІГАННЯ ЕЛЕМЕНТІВ РОЗРАХУНКОВИХ ТЕПЛОВИХ СХЕМ

Метою є розробка класу, що реалізує сховище для зберігання об'єктів на основі динамічного масиву даних та перевантаження стандартних операторів мови програмування *C++*.

Завдання

Варіанти завдання обираються із додатків А та Б.

1. Розробити клас *Array* для зберігання об'єктів, створених в лабораторних роботах 1–2.

2. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

Для спрощення розглянемо реалізацію класу, який зберігає цілі числа у динамічному масиві. Визначення класу *Array* наведено нижче, а повний приклад у додатку Г. Конструктор класу приймає один параметр – розмір, та виділяє необхідну кількість байт у пам'яті. Деструктор звільняє зайняту пам'ять.

```
#ifndef ARRAY_H_
#define ARRAY_H_
#include <iostream>
#include <iomanip>
class Array {
    friend std::ostream &operator<<(std::ostream &os, const
                                    Array & a);
    friend std::istream&operator>>(std::istream&is,Array& a);
    size_t size_; //розмір масиву
    int *storage_; // динамічний масив
public:
    Array();
    Array(size_t sz);
    Array(const Array & a);
    virtual ~Array();
    int &operator[](const size_t ind); //запис
    int operator[](const size_t ind) const; //читання
    Array &operator=(const Array &a);
    size_t size() const;
    void resize(const size_t new_size);
    int sum() const;
    int operator()() const;
    void print() const;
    void set();
    void init(const size_t sz);
};
#endif /* ARRAY_H_ */
```

Новим у порівнянні з лабораторною роботою 2 є наявність конструктору копії (***Array***(***const*** *Array & a*)), перевантаження операторів «[ ]», «( )», «=», «+=», потокових операторів «>>», «<<» для роботи з масивом. Визначення однієї зі стандартних операцій для роботи з операндами абстрактних типів здійснюють за допомогою функції «*operator* знак( )», де «знак» – позначка операції («+», «–» , «+=» і т.п.).

Якщо об'єкту для зберігання даних потрібно виділити пам'ять у купі (*new* або *malloc*), відповідальність за вивільнення ресурсів у деструкторі (*delete* або *free*) несе клас (і програміст). Треба запам'ятати правило трьох – якщо у класі визначають деструктор, конструктор копії або оператор присвоєння («=»), то у класі визначають всі три означені компоненти. В нашому випадку необхідний деструктор для вивільнення ресурсів, тому додатково визначаємо інші два компоненти. Конструктор копії дозволяє створити об'єкт за допомогою синтаксису *class A b*(); *class A c*(*b*), а оператор «=» – *class A b*(); *class A c=b*. В обох випадках запобігають поверхневому (*shallow*) копіюванню адрес покажчиків, натомість виконують повне копіювання вмісту пам'яті, на який вказує покажчик (глибоке (*deep*) копіювання).

Оператор взяття індексу «[ ]» використовують для доступу до елементу масиву, а функціональний оператор «( )» – для довільної обробки масиву *Array*, наприклад, пошук суми та кількості елементів, що відповідають заданій умові. Також його можна використовувати для індексації елементів масиву, в тому числі багатовимірних.

У випадках перевантаження оператора як члену класу, він приймає перший параметр (покажчик *this,* що вказує на об'єкт, який здійснив виклик функції) неявно, а другий (посилання на об'єкт відповідного класу) – явно, в іншому випадку – два параметри (посилання на об'єкти) передаються явно

```
const MyClass operator+(const MyClass &o){return m_a+o.m_a;};
```

або

```
friend MyClass operator-(const MyClass &o1, const MyClass &o2)
                         {return o1.m_a-o2.m_a;};
```

Таблиця 3.1 – Правила перевантаження операторів

Оператор або клас операторів	Форма
Унарні	Член класу
=, ( ), [ ], ->, ->*	Обов'язково член класу
+=, -=, /=, *=, ^=, &=, |=, %=, >>=, <<=	Член класу
Інші бінарні	Не член класу

Якщо оператор або функцію визначено друзями (`friend`) класу, то вони отримують доступ до його приватних членів. Друзі не є членами класу, тому вони можуть бути визначеними в будь-якій секції, де визначається рівень доступу до членів класу – `public`, `private` або `protected`.

Порядок виконання роботи

1. Виконати приклад програми (додаток Г) та проаналізувати її роботу, зробити висновки.

2. Додати в програму інші арифметичні оператори, реалізувавши їх у відповідності до рекомендацій, наведених у табл. 3.1.

3. Змінити програму, розроблену в лабораторній роботі 2, додавши перевантаження потокових операторів «>>», «<<».

4. Змінити програму п. 1 таким чином, щоб вона могла зберігати об'єкти, визначені в лабораторних роботах 1–2.

Питання для самоперевірки

1. Призначення механізму перевантаження функцій?
2. Чи можна перевантажити конструктор? Якщо так, то для чого?
3. Чи можна перевантажити деструктор?
4. Що таке аргументи по замовчуванню і коли їх використовують?
5. Для чого використовують конструктор копій?
6. Прокоментуйте реалізацію конструктора з параметром.
7. Прокоментуйте реалізацію конструктора копії.
8. Прокоментуйте реалізацію деструктора.

Лабораторна робота 4
УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ. ШАБЛОНИ

Метою є ознайомлення з шаблонними функціями та класами, розробка програми з їх використанням.

Завдання

Варіанти завдання обираються із додатків А та Б.

1. Розробити програму з використанням перевантаження функцій та шаблонних функцій.

2. Розробити шаблонний клас для розв'язання задачі теплофізики.

3. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

C++ надає можливість скоротити кількість функцій, які використовують в програмі, за допомогою механізму перевантаження, коли функції мають однакові імена, але відрізняються кількістю та типами параметрів (сигнатурою). При цьому спрощується написання коду, але потрібно мати реалізації перевантажених функцій зі всіма можливими аргументами. Це ж справедливе

й для класів. Відповідальність за правильний виклик функції покладається на компілятор. Є інший механізм – механізм параметризованих функцій та класів.

Подібно до класу, що є схематичним описом побудови об'єктів, шаблон також є схематичним описом побудови класів та функцій. Використовуючи шаблони, створюють узагальнені специфікації для класів та функцій, які називають параметризованими (*generic*) класами та функціями. За допомогою реалізації узагальнених функцій можна зменшити розмір та складність програми, а найчастіше шаблони використовують в бібліотеках класів.

Визначаючи параметризований клас, створюють каркас(шаблон), який описує усі алгоритми, що використовуються класом. Фактичний тип даних буде вказаний в якості параметру при конкретизації об'єктів цього класу. Компілятор автоматично згенерує відповідний об'єкт на основі вказаного типу. Загальна форма декларування параметризованої функції та класу

```
template<class T>T FuncName(T parm1){//тіло функції};
```

та

```
template <class T> class ClassName{//опис класу};
```

В конструкції *<class T> T* є ім'ям типу шаблону, який в кожному випадку конкретизації буде замінюватися конкретним типом даних. За необхідності визначають більше одного параметризованого типу даних, записуючи їх через кому. Зазвичай *«template T»* еквівалентний *«class T»*. У межах визначення класу шаблонне ім'я можна використовувати у будь-якому місці, наприклад,

```
template <typename T> T sum(T a, T b){
    return (a+b);
}
```

або

```
template <typename T, typename U> T sum(T a, U b){
    return (a+b);
}
```

Шаблонні класи можуть мати додаткові нешаблонні параметри, наприклад, *template <class Type, int size> class Buffer*{};

Спеціалізацію шаблону використовують тоді, коли компілятор не може визначити тип операндів (наприклад, для *char** не визначено оператор «>»)

```
template<class T>
T Max(T a, T b) {
return a>b?a:b;
}
```

```
template<char*>
char* Max(char* a, char* b) {
return strcmp(a,b)<0?a:b;
}
```

Методи шаблонного класу визначаються наступним чином:

```
#include <iostream>
template<class T>
class A {
    T data_;
public:
    A(T data) : data_(data) { }
    void hello();
    void by() {std::cout<<"By, "<<data_<< "!\n"; }
    template<class U> friend std::ostream& operator <<
                            (std::ostream& stream,A<U>&obj);
};
template<typename T>
void A<T>::hello() {std::cout<<"Hello, "<< data_ << "!\n";}
template<typename T>
std::ostream& operator <<(std::ostream& stream, A<T>&obj)
              {return stream << obj.data_ << std::endl; }
int main() {
    A<const char*> a("Ivan");
    a.hello(); a.by(); std::cout << a;
    return 0;
 }
```

Реалізація та визначення шаблонних класів, як правило, знаходяться в тому самому файлі.

Порядок виконання роботи

1. Розробити функцію виведення на екран одновимірного масиву за допомогою перевантаження та шаблонів.
2. Замінити розроблений в лабораторній роботі 3 клас *Array* шаблонним класом *ArrayT*, який приймає 2 параметри – тип елементу та розмір масиву.
3. Розробити програму, в якій формується масив розроблених в лабораторній роботі 2 об'єктів *ArrayT<type, n>* та розв'язується задача для різних наборів вхідних даних, друкуються результати роботи програми на екран у вигляді таблиці.

Питання для самоперевірки

1. Що таке шаблон та шаблонна функція?
2. Скільки разів компілюється шаблонна функція?
3. Чи можливо, щоб шаблонна функція мала аргументи за замовчанням?
4. Назвіть правила визначення шаблонних класів.
5. Чи може шаблонний клас мати аргументи за замовченням?

Лабораторна робота 5

КОНТЕЙНЕРИ ТА АЛГОРИТМИ *STL*

Метою є ознайомлення з принципами використання стандартної бібліотеки шаблонів *C++* (*Standard Template Library*, *STL*) при розробці програм.

Завдання

Варіанти завдання обираються із додатка Д.

1. Розробити програму, в якій дані зберігаються в заданому контейнері, додати обробку даних з використанням стандартних алгоритмів згідно з варіантом.
2. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

Бібліотеку *STL* побудовано на шаблонах класів і функцій, що забезпечує можливість уніфікованої роботи з різними типами даних. В *STL* розрізняють наступні компоненти:

- *контейнери* (*containers*) – це класи, призначені для зберігання сукупностей об'єктів (як вбудованих, так і визначених користувачем типів);
- *ітератори* (*iterators*) – це абстракція покажчика. Призначення ітератора – забезпечення доступу до об'єкта, на який він посилається (розіменування), і перехід від одного елемента контейнера до іншого (ітерація);
- *алгоритми* (*algorithms*) – це функції для маніпулювання об'єктами, що містяться в контейнері;
- *функціональні* об'єкти (функтори, *functors*);

Контейнери. У табл. 5.1 приведені основні класи контейнерів бібліотеки *STL* (*C++*03). Стандарт *C++*11 визначає декілька додаткових контейнерів. Призначення контейнеру – зберігання об'єктів. Незалежно від фактичної організації контейнера,, кожен з них надає строго визначений інтерфейс (ітератор), через який з ним взаємодіють алгоритми. Для кожного контейнеру

обирається найбільш функціональний тип ітератора (табл. 5.2), який може бути ефективно реалізований для даного контейнера.

Ітератори. У залежності від набору підтримуваних операцій розрізняють 5 типів ітераторів (табл. 5.2), які забезпечують доступ до елементів контейнерів. Стандартні ітератори визначено в файлі *iterator*.

З точки зори ітератора елементи, які зберігаються в контейнері, складають неперервну послідовність. Ітератор першого елементу в цій послідовності повертає метод *begin*(), а ітератор елемента, що розташований за останнім – метод *end*(). Крім звичайних ітераторів у *STL* існують зворотні ітератори (*reverse iterator*). Зворотний ітератор відрізняється тим, що обходить послідовність елементів у контейнері у зворотному порядку.

При розміщенні у контейнер використовується копія об'єкта (повинен бути визначений конструктор копії). Для економії пам'яті в контейнері зберігають покажчики на об'єкти. Від правильного вибору контейнеру залежить ефективність програми.

Таблиця 5.1 – Контейнери *STL*

Тип контейнера	Клас контейнера	Директива *#include*
Послідовні контейнери (*sequence containers*)	*vector* – масиви	< *vector* >
	deque – двостороння черга	<*deque*>
	list – списки	< *list*>
Асоціативні контейнери (*associative containers*)	*map*, *multimap* – відсортований контейнер, що містить пари «ключ-дані»	<*map*>
	set, *multiset* – множини	<*set*>
Адаптори (*container adaptors*)	*queue* – черга *FIFO, priority queue* – пріоритетна черга	<*queue*>
	stack – стек *LIFO*	<*stack*>
Псевдоконтейнери	*bitset* – бітові набори	<*bitset*>
	valarray – масив числових даних	<*valarray*>

Алгоритми. Бібліотека *STL* надає алгоритми: 1) операцій, що не змінюють послідовності (*find_if*, *count*, *search*); 2) операцій що змінюють послідовність (*replace*, *remove*, *reverse*); 3) сортування (*sort*, *stable_sort*, *partial_sort*); 4) бінарного пошуку (*lower_bound*, *upper_bound*); 5) роботи з купою (*make_heap*, *push_heap*); 6) злиття (*merge*); 7) пошуку мінімуму та максимуму (*min*, *max*) та ін. [7, 8]

Призначення алгоритмів – маніпулювання елементами контейнера. Будь-який алгоритм розглядає вміст контейнера як послідовність, що задається ітераторами першого і наступного за останнім елементів. Ітератори забезпечують інтерфейс між контейнерами та алгоритмами, завдяки чому досягається гнучкість й універсальність бібліотеки *STL*.

Таблиця 5.2 – Ітератори *STL*

Тип ітератора	Доступ	Розіменування	Ітерація	Порівняння
Виводу (*output*)	***r**ead*	*	++	
Вводу (*input*)	***w**rite*	*, ->	++	==, !=
Прямий (*forward*)	*r*/*w*	*, ->	++	==, !=
Двічі спрямований (*bidirectional*)	*r*/*w*	*, ->	++, --	==, !=
З довільним доступом (*random-access*)	*r*/*w*	*, ->, []	++,--, +, -, +=, -=	==, !=, <,<=, >, >=

Кожний алгоритм використовує ітератори тільки визначеного типу. Так, алгоритму простого пошуку (*find*), який переглядає всі елементи підряд, потрібний ітератор вводу, а алгоритму двійкового пошуку (*binary_search*) необхідно переходити до будь-якого елемента послідовності, тому потрібен ітератор з довільним доступом.

Частину алгоритмів додатково реалізовано як методи контейнерних класів. Такі алгоритми оптимізовані та працюють швидше за аналогічні, визначені в *algorithm*.

Порядок виконання роботи

1. Вивчити роботу прикладу (додаток Е).
2. Вивчити особливості роботи з заданим контейнером, на прикладі роботи із цілими числами. Застосувати до даних, що зберігаються в контейнері, алгоритми *find, count, for_each, generate, transform, replace, remove, unique, sort, binary_search, max, min, max_element, min_element.*
3. Змінити програму п. 1 для роботи з об’єктами, розробленими в лабораторних роботах 1–2.

Питання для самоперевірки

1. Призначення *STL*? Що таке контейнери, алгоритми, ітератори?
2. На які типи поділяються контейнери.
3. Як працюють алгоритми? Які алгоритми ви знаєте?
4. Види ітераторів. Від чого залежить обрання типу ітератора?

Лабораторна робота 6

УСПАДКУВАННЯ

Метою є ознайомлення з принципами успадкування та розробка простої ієрархії класів.

Завдання

1. Розробити ієрархію класів «Підігрівач» для використання в розрахункових теплових схемах. В ієрархію включити звичайний підігрівач, різні типи поверхневих підігрівачів, підігрівачі, що змішують.

2. Створити програму для роботи з розробленою ієрархією класів.

3. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

Під час розв'язання задач теплофізики постає проблема роботи з геометричними формами, що обмежують деякий простір, формуючи конструкцію елементів. Для таких елементів доцільно визначити класи простих геометричних фігур та за їх допомогою формувати більш складні контури, поверхні та об'єми. Однією з супутніх задач є визначення площі поверхні, що випромінює або поглинає теплову енергію.

Для вирішення цієї проблеми доцільно скористатися одним із механізмів ООП, а саме *успадкуванням* – процесом створення нових *похідних* (*derived*) класів із вже існуючих *базових* (*base*) класів. Похідний клас отримує атрибути та поведінку базового класу, але може бути вдосконалений за рахунок додавання власних. Базовий клас при цьому залишається незмінним. Похідний клас є більш специфічним, ніж його базовий клас, та представляє меншу групу об'єктів. Кожен об'єкт похідного класу є також об'єктом відповідного базового класу, проте, зворотне невірно – об'єкт базового класу не є об'єктом класів, породжених ним. Наприклад, прямокутник є чотирикутником, але не кожний чотирикутник є прямокутником. Успадкування сприяє *повторному використанню коду*, що прискорює розробку та зменшує кількість помилок.

Успадкування реалізує відношення *«is-a»* між об'єктами. *C++* підтримує одиночне (від одного класу) та множинне (від декількох класів) успадкування, яке може бути відкритим (*public*), закритим (*private*) й захищеним (*protected*). Відкрите – це успадкування інтерфейсу, а закрите – успадкування реалізації. В будь-якому випадку, доступ до приватних членів базового класу заборонений (рис. 6.1).

Успадкування в *C++*					
public		***protected***		***private***	
base	***derived***	***base***	***derived***	***base***	***derived***
public	*public*	*public*	*protected*	*public*	*private*
protected	*protected*	*protected*	*protected*	*protected*	*private*
private	–	*private*	–	*private*	–

Рисунок 6.1 – Успадкування в *C++*

Похідні класи успадковують дані й методи базового класу. Конструктори не успадковуються, деструктори успадковуються. Порядок виклику конструкторів:

- якщо в конструкторі похідного класу явний виклик конструктора базового класу відсутній, автоматично викликається конструктор базового класу за замовчанням;
- для ієрархії, що складається з декількох рівнів, конструктори базових класів викликаються, починаючи з самого верхнього рівня. Після – конструктори тих елементів класу, які є об'єктами, а потім – конструктор класу;
- параметри до конструктора базового класу передаються через список ініціалізації.

Правила успадкування деструкторів:

- якщо в похідному класі не описано деструктор, то він формується за замовчанням і викликає деструктори всіх базових класів;
- на відміну від конструкторів, деструктори базових класів викликаються автоматично;
- деструктори викликаються в порядку, строго зворотному виклику конструкторів: спочатку деструктор класу, потім – деструктор елементів класу, а потім деструктор базового класу.

Приклад роботи з ієрархією класів геометричних фігур (рис. 6.2):

```
class Shape {};
class Dot: public Shape {};
class Ellipse: public Point {};
class Circle: public Ellipse {};
class Polygon: public Shape {};
class Rectangle: public Polygon {};
class Sqaure: public Rectangle {};
class Triangle: public Polygon {};
```

```
// допоміжні класи
class Point {}; // точка на координатній площині
class Angle {}; // кут
```

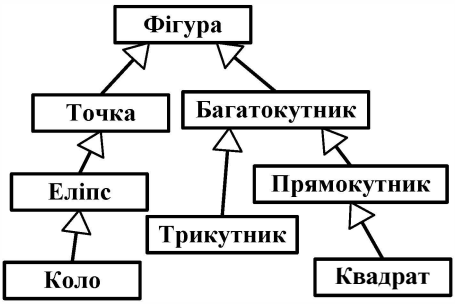


Рисунок 6.2 – Ієрархія класів

Порядок виконання роботи

1. Реалізувати ієрархію класів (рис. 6.2). Для кожного класу визначити методи для введення-виведення даних, розрахунку периметру та площі за формулами. Додати в конструктор та деструктор виведення на екран відповідного повідомлення «*Shape*» та «~*Shape*», «*Dot*» та «~*Dot*» тощо.

2. Розробити ієрархію класів підігрівачів (див. додаток А та [1]), привести повну *UML*-діаграму та код *C++*. Винести в базовий клас всі властивості та поведінку, характерні для всіх похідних класів.

Питання для самоперевірки

1. У чому полягає суть успадкування?
2. Які види успадкування ви знаєте?
3. Чи всі члени базового класу успадковують похідні класи?
4. Який порядок виклику конструкторів та деструкторів при успадкуванні?
5. Як на діаграмах UML позначається успадкування?
6. У яких випадках успадкування не можна застосовувати?
7. Розробіть інші варіанти ієрархії геометричних фігур. Обґрунтуйте ваш вибір.

Лабораторна робота 7

ПОЛІМОРФІЗМ

Метою є ознайомлення з абстрактними класами, віртуальними функціями та принципами використання поліморфної поведінки об'єктів.

Завдання

1. Замінити в ієрархії класів (лабораторна робота №6) базовий клас абстрактним базовим класом.

2. Написати програму з використанням розробленої в п. 1 ієрархії класів.

3. Звіт про роботу повинен містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання.

Методичні вказівки до виконання завдання

Поліморфізм – це здатність об'єктів різних класів, пов'язаних за допомогою *успадкування*, реагувати різним чином при зверненні до одного й того ж метода. Поліморфізм реалізується за допомогою віртуальних функцій, визначених з ключовим словом *virtual*: ***virtual void*** *print*(). Після визначення віртуальною, функція лишається такою на всіх підлеглих рівнях ієрархії.

Якщо у класі визначено *чисту віртуальну* (*pure virtual*) функцію, то такий клас називають абстрактним або інтерфейсом. При визначенні чистої віртуальної функції потрібно праворуч додати «=0», наприклад, *virtual void print*()=0. Всі віртуальні функції повинні бути визначеними у класах-нащадках. Заборонено створювати об'єкт абстрактного класу, але можна використовувати покажчик на такий об'єкт.

Якщо у класі є віртуальні функції, то деструктор також – віртуальний. Для роботи з віртуальним функціями використовується механізм пізнього зв'язування, коли компілятору не відомі типи даних, а для їх визначення під час виконання програми використовується механізм *RTTI* (*RunTime Type Information*).

Наприклад, створимо масив, елементами якого будуть покажчики на *Shape*, потім додамо покажчики на два об'єкти *Circle* та *Rectangle*, після чого двічі викличемо метод *print*( ):

```
#include <iostream>
#include <vector>
struct Shape {
    virtual void print()=0;
    virtual ~Shape() {    } };
```

```
struct Circle: public Shape {
    void print() {
        std::cout << "Circle\n";
    }
};
struct Rectangle: public Shape {
    void print() {
        std::cout << "Rectangle\n";
    }
};
int main() {
    std::vector<Shape*> f;
    f.push_back(new Circle);
    f.push_back(new Rectangle);
    f[0]->print(); // «Circle»
    f[1]->print(); // «Rectangle»
    delete f[0];
    delete f[1];
    return 0;
}
```

За допомогою покажчика на базовий клас можна звернутися до методів похідного класу, використовуючи динамічне приведення типів *dynamic_cast*<тип*>(покажчик), наприклад,

```
Circle *c=new Circle;
Shape *p = c;
dynamic_cast<Circle*>(p)->out();
```

Перший випадок (*f*[0]*->print*()) – приведення типів по ієрархії вгору (*upcast*), другий (*dynamic_cast<Circle*>*(*p*)*->out*();) – вниз (*downcast*).

Порядок виконання роботи

1. Змінити програму, розроблену в лабораторній роботі 6 таким чином, щоб клас *Shape* виконував роль інтерфейсу.

2. Додати в базовий клас масив для зберігання координат точок, що формують контури фігури. Додати віртуальний метод *draw*(), який буде заповнювати цей масив.

3. Додати до базового класу метод обчислення площі через розбивання опуклого багатокутника на трапеції $S = \frac{1}{2}|\sum_{i=1}^{n}[(x_i + x_{i+1})(y_i - y_{i+1})]|$. Обчислити площі похідних фігур обома способами – за формулою площі та за координатами точок.

4. Змінити ієрархію класів підігрівачів (лабораторна робота 6), зробивши базовий клас абстрактним.

Питання для самоперевірки

1. Що означає поліморфізм та як він реалізується?
2. Чи можна покажчику на базовий клас присвоїти значення адреси об'єкта похідного класу?
3. Що необхідно для того, щоб реалізувати поліморфізм в програмі?
4. Для чого використовують абстрактні класи?
5. Чи можна створювати об'єкти абстрактного класу?
6. Назвіть відмінності між віртуальними та чистими віртуальними функціями.
7. Чи всі віртуальні функції в абстрактному базовому класі повинні бути визначені як чисті віртуальні функції?
8. Чи можна створювати функцію, параметром якої є покажчик на абстрактний клас?

Лабораторна робота 8

РОЗРОБКА ПРОГРАМИ ДЛЯ РОЗРАХУНКУ ФРАГМЕНТУ ТЕПЛОВОЇ СХЕМИ

Метою є розробка програми для розрахунку фрагменту теплової схеми та використання файлів для зберігання даних.

Завдання

Варіанти завдання обираються із додатка Ж.

1. Розробити програму для розрахунку фрагменту теплової схеми із використанням композиції, агрегації та успадкування.
2. Звіт про роботу має містити короткий опис роботи, алгоритми, *UML*-діаграми, текст програми і результати її виконання..

Методичні вказівки до виконання завдання

В даній роботі потрібно розробити програму для розрахунку фрагменту теплотехнічної схеми. Схеми, розрахункові співвідношення, методики та приклади розрахунків наведено в [1].

Агрегація (рис. 8.1 *а*) та *композиція* (рис. 8.1 *б*) – це спосіб взаємодії між об'єктами «*has-a*», коли полями даних об'єкту є інші об'єкти. Різниця між композицією і агрегацією полягає в тому, що у разі композиції ціле явно контролює час життя своєї складової частини (частина не існує без цілого), а в разі агрегації ціле хоч і містить свою складову частину, термін їхнього життя не пов'язані (наприклад, складова частина передається через параметри конструктора).

Елементи теплової схеми (додаток Ж) складаються з декількох окремих компонентів, наприклад, підігрівач високого тиску може бути з ОП, з ОД, без ОП та ОД, з ОП та ОД. У кожному випадку параметри схеми визначаються параметрами окремих елементів та умовами роботи (тип носія, тиск, ентальпія тощо). Для таких елементів потрібно правильно вибрати тип зв'язку. Наприклад, можна побудувати ієрархію класів, в якій кожний окремий клас є композитним, тобто, його члени-дані є об'єктами або покажчиками на об'єкти, які визначають внутрішній стан об'єктів ієрархії.

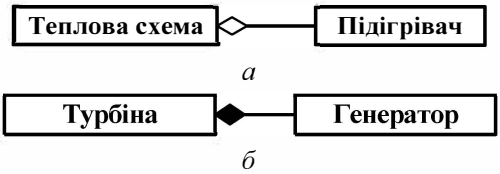


Рисунок 8.1 – Взаємодія між класами по типу «*has-a*»:
а – агрегація; *б* – композиція

Для композиції (рис. 8.1 *б*) код (спрощений) може бути наступним

```
class Generator { };
class Turbine {
    Generator *gen_;
  public:
    Turbine () : gen_(new Generator) {  }
    ~Turbine() { delete gen_;  }
};
int main() {
    Turbine turb;
    return 0;
}
```

Зазначимо, що кожний екземпляр класу, який використовується як поле даних композитного класу, є повністю самостійним об'єктом. Композитний клас керує тільки часом життя такого об'єкту. У прикладі об'єкт класу «Генератор» створюється одночасно із об'єктом класу «Турбіна», а наприкінці одночасно з ним знищується.

Робота з файлами через потоки введення-виведення. Файлові операції в *C++* використовують відповідні класи введення-виведення, визначені в файлі *fstream*. Основні прийоми роботи з файловими потоками ти ж самі, що були розглянуті в лабораторній роботі 1. Для роботи з файлом у режимі читання/запису створюють об'єкт класу *fstream* (або *ofstream* (запис), або

ifstream (читання)). Для керування методом доступу до даних використовують відповідні константи класу *ios_base* (табл. 8.1).

Детальний приклад роботи з файлами наведено в додатку И. З метою запобігання помилкам читання даних, для простих текстових файлі оптимальним є зчитування даних рядками та їх подальший розбір (парсинг), див. приклад в додатку И.

Таблиця 8.1 – Константи класу *ios_base*

Режим	Опис
ios_base::*app*	Додавання даних у кінець файлу
ios_ios::*ate*	Перемістити покажчик в кінець файлу.
ios_base::*in*	Відкрити файл для читання (вводу).
ios_base::*out*	Відкрити файл для запису (виводу)
ios_base::*trunc*	Якщо файл існує, видалити
ios_base::*nocreate*	Якщо файл не існує, то видати помилку.
ios_base::*noreplace*	Якщо файл існує, то видати помилку.
ios_base::*binary*	Відкрити файл у двійковому режимі

Зазначимо, що у випадках, коли атрибути класу – покажчики на дані в купі або стандартні контейнери, потрібно використовувати прості текстові файли або серіалізацію (перетворення об'єкту на послідовність байтів), а не запис даних блоками за допомогою методів *write*(), *read*().

Таблиця 8.2 – Стани потоків

Біт	Функція доступу	Опис стану
eofbit	*eof()*	1, якщо досягнуто кінець файлу
badbit	*bad()*	1, якщо потік пошкоджений
failbit	*fail()*	1, якщо помилка читання/запису даних
goodbit	*good()*	0, якщо з потік у справному стані
	rdstate()	повертає стан потоку
	clear()	скидає всі біти помилок, «очищує» потік, генерує виключення ios_base::failure, якщо неможливо виправити стан потоку.

Робота з файлами – потенційне джерело помилок, тому в програмний код потрібно включати перевірки існування файлів (потоків). На помилки, що виникають в процесі читання/запису, потік реагує відповідним чином через встановленням відповідних бітів (прапорців) (табл.. 8.2).

Порядок виконання роботи

1. Згідно з варіантом розробити діаграму класів елемента розрахункової схеми, теплоносій також реалізувати окремим класом.

2. Розробити та протестувати програму для розрахунку фрагменту теплової схеми. Вихідні дані та методику розрахунків див. [1] та додаток Ж. Вихідні дані та результат мають бути відповідно у файлах *in.dat* та *out.dat*.

Питання для самоперевірки

1. Яким чином реалізується зв'язок «*has-a*»?
2. Який тип зв'язку сильніший агрегація чи композиція?
3. В яких випадках доречно використовувати успадкування, а в яких композицію?
4. Які типи зв'язків між класами ви обрали при розробці індивідуального завдання? Обґрунтуйте свій вибір.
5. Які елементи *STL* ви використали в програмі, чому?
6. Які типи доступу до вмісту файлів ви можете назвати?
7. Призначення класів *fstream, ofstream, ifstream*?

ДОДАТКИ

Додаток А
ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ 1

Розробити клас для типового елементу теплової схеми турбоустановки згідно з варіантом як елементу для подальшого теплового та конструктивного розрахунку [1]:

1) турбіна;

2) конденсатор;

3-4) конденсатний та живильний насоси;

5-9) підігрівачі високого (ПВТ) та низького (ПНТ) тиску без охолоджувача пари (ОП) та з охолоджувачем дренажу (ОД), з ОП, з ОД, з ОП та ОД;

10) деаератор;

11-12) мережеві та сальникові підігрівачі;

13) випарник;

14) котел;

15) генератор.

Додаток Б
ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ 2

Задачі за темою «Тепловіддача при змушеному поперечному обтіканні циліндра і пучка труб» [2, с. 119]:

1. Мідний шинопровід круглого перетину діаметром d = 15 мм охолоджується поперечним потоком повітря. Швидкість і температура потоку повітря, що набігає, відповідно w = 1 м/с і t_p = 20 ºС. Обчислити коефіцієнт тепловіддачі від поверхні шинопроводу до повітря й припустиму силу струму в шинопроводі за умови, що температура його поверхні не повинна перевищувати t_c = 80 ºС. Питомий електричний опір міді ρ= 0,0175 Ом·мм^2/м.

Відповідь: α = 23,8 Вт/(м^2 ·град) I = 825 А

2. Водяний калориметр, що має форму трубки із зовнішнім діаметром d = 15 мм, поміщений у поперечний потік повітря. Повітря має швидкість w = 2 м/с, спрямовану під кутом 90º до осі калориметра, і середню температуру t_p = 20 ºС. При стаціонарному тепловому режимі на зовнішній поверхні калориметра встановлюється постійна середня температура t_c = 80 ºС. Обчислити коефіцієнт тепловіддачі від трубки до повітря й тепловий потік на одиницю довжини калориметра.

Відповідь: $\alpha = 36{,}3$ Вт/(м^2·град) $q_l = 102$ Вт/м^2

3. Циліндрична трубка діаметром $d = 20$ мм охолоджується поперечним потоком води. Швидкість потоку $w = 1$ м/с. Середня температура води $t_p = 10$ ºС, а температура поверхні трубки $t_c = 50$ ºС. Визначити коефіцієнт тепловіддачі від поверхні трубки до охолодної води.

Відповідь: $\alpha = 7050$ Вт/(м^2·град)

4. У теплообміннику шаховий пучок труб обтікає поперечний потік трансформаторного масла. Зовнішній діаметр труб у пучку $d = 20$ мм**.** Поперечний крок $s_1 = 2{,}5d$, поздовжній крок $s_2 = 1{,}5d$**.** Середня швидкість у вузькому перетині пучка й середня температура масла відповідно дорівнюють $w = 0{,}6$ м/с і $t_p = 40$ ºС**.** Визначити коефіцієнт тепловіддачі від поверхні труб до масла для третього ряду пучка труб за умови, що температура поверхні труб $t_c = 90$ ºС. Потік обмиває труби під кутом атаки $\psi = 60$º.

Відповідь: $\alpha_{\psi=60} = 1060$ Вт/(м^2·град)

5. Трубчастий повітропідігрівник передбачається виконати із труб діаметром $d = 38$ мм**,** розташованих у коридорному порядку з поперечним і поздовжнім кроками $s_1 = s_2 = 2{,}5d$. Число труб в одному ряді поперек потоку обране $m = 8$**,** число рядів $n = 5$. Температура повітря, що надходить у підігрівник, $t_{p1} = 20$ ºС і на виході з підігрівника, $t_{p2} = 80$ ºС**.** Температура стінки $t_c = 150$ ºС**.** Якої довжини повинні бути труби, щоб при швидкості повітря у вузькому перетині пучка $w = 10$ м/с, кількість тепла, передана повітрю, становило $Q = 125$ кВт?

Відповідь: $l = 3$ м

Задачі за темою «Тепловіддача при вільному русі рідини»[2, с. 134]:

6. Визначити коефіцієнт тепловіддачі від вертикальної плити висотою $H = 2$ м до навколишнього спокійного повітря, якщо відомо, що температура поверхні плити $t_c = 100$ ºС**,** температура навколишнього повітря вдалині від поверхні $t_p = 20$ ºС.

Відповідь: $\alpha = 7{,}92$ Вт/(м^2 ·град).

7. У контурі для вивчення гідродинаміки і тепловіддачі рідинно-металевих теплоносіїв метал у забірному баку нагрівається за допомогою горизонтального електричного нагрівника, що має форму циліндра діаметром $d = 50$ мм. Обчислити коефіцієнт тепловіддачі від поверхні нагрівника до металу для випадку, коли контур заповнений натрієм з температурою $t_p = 200$ ºС**,** а температура поверхні нагрівача $t_c = 400$ ºС.

Відповідь: $\alpha = 15750$ Вт/(м^2 ·град).

Продовження додатка Б

Задача за темою «Тепловіддача при конденсації пари» [2, с. 142]:

8. На зовнішній поверхні вертикальної труби діаметром d = 20 мм і висотою H = 2 м конденсується суха насичена водяна пара при тиску p = 1 бар. Температура поверхні труби t_c = 94,5 °С. Визначити середній за висотою коефіцієнт тепловіддачі від пари до труби й кількість пари G кг/год, що конденсується на поверхні труби.

Відповідь: α = 7840 Вт/(м^2 ·град), G = 8 кг/год

Задача за темою «Тепловіддача при кипінні рідини» [2, с. 165]:

9. Визначити коефіцієнт тепловіддачі від зовнішньої поверхні трубки випарника до киплячої води, якщо теплове навантаження поверхні нагрівання $q = 2 \cdot 10^5$ Вт/м^2, режим кипіння пузирковий і вода знаходиться під тиском p = 2 бар

Відповідь: α = 18400 Вт/(м^2 ·град)

Задача за темою «Теплопровідність при стаціонарному режимі» [2, с. 7]:

10. Електричний нагрівник виконаний з ніхромового дроту діаметром d = 2 мм і довжиною l = 10 м. Нагрівник обдувається холодним повітрям з температурою t_p = 20 °С. Обчислити тепловий потік з одного погонного метра нагрівника, а також температури на поверхні й осі дроту, якщо сила струму, що проходить через нагрівник, становить 25 А. Питомий електричний опір ніхрому ρ = 1,1 Ом·мм^2/м, коефіцієнт теплопровідності ніхрому λ = 17,5 Вт/(м·град), коефіцієнт тепловіддачі від поверхні нагрівника до повітря α = 46,5 Вт/(м^2·град).

Відповідь: q_l = 218,5 Вт/м, температура на осі дроту t_o = 770 °С, температура на поверхні дроту t_c = 769 °С.

Задачі за темою «Тепловіддача при змушеному поздовжньому обтіканні плоскої пластини» [2, с. 63]:

11. Тонку пластину довжиною l_0 = 2 м і шириною a = 1,5 м обтікає поздовжній потік повітря. Швидкість і температура потоку, що набігає, рівні відповідно w_0 = 3 м/с і t_0 = 20 °С. Температура поверхні пластини t_n = 90 °С. Визначити середній за довжиною пластини коефіцієнт тепловіддачі й кількість теплоти, що віддає пластиною повітрю.

Відповідь: α = 4,87 Вт/(м^2·град), Q = 2050 Вт .

12. Плоска тонка пластина довжиною l_0 = 1 м обтікається поздовжнім потоком повітря. Швидкість і температура потоку, що набігає, рівні відповідно w_0 = 80 м/с і t_0 = 10 °С.

Закінчення додатка Б

Перед пластиною встановлені турбулізуючі ґрати, в результаті чого режим руху в примежовому шарі по всій довжині пластини турбулентний. Визначити середній по поверхні пластини коефіцієнт тепловіддачі й значення місцевого коефіцієнта тепловіддачі на задній кромці, а так само товщину гідродинамічного примежового шару на задній кромці пластини.

Відповідь: α = 202 Вт/(м^2·град), $\alpha_м$ = 157,5 Вт/м^2 град, $\delta_т$ = 16,6 мм .

Задачі за темою «Тепловий розрахунок теплообмінних апаратів» [2, с. 216]:

13. Визначити поверхню нагрівання водяного економайзера, у якому теплоносії рухаються за противоточною схемою, якщо відомі наступні величини: температура газів на вході $t_1^{'}$ = 420 ºС, витрата газів G_1 = 220 т/ч, теплоємність газів c_{p1} = 1,045 кДж/(кг·град), температура води на вході $t_2^{'}$ =105 ºС, витрата води G_2 = 120 т/ч, теплоємність води c_{p2} = 4,22 кДж/(кг·град), кількість переданого тепла Q = 13500 кВт, коефіцієнт теплопередачі від газів до води k = 79 Вт/(м^2·град). Визначити кінцеві температури теплоносіїв і площу поверхні теплообміну.

Відповідь: $t_1^{"}$ = 208,6 ºС , $t_2^{"}$ = 201 ºС, F = 1108 м^2.

14. Визначити поверхню нагрівання водяного економайзера, у якому теплоносії рухаються за прямоточною схемою, якщо відомі наступні величини: температура газів на вході $t_1^{'}$ = 420 ºС, витрата газів G_1 = 220 т/ч, теплоємність газів c_{p1} = 1,045 кДж/(кг·град), температура води на вході $t_2^{'}$ = 105 ºС, витрата води G_2 = 120 т/ч, теплоємність води c_{p2} = 4,22 кДж/(кг·град), кількість переданого тепла Q = 13500 кВт, коефіцієнт теплопередачі від газів до води k = 79 Вт/(м^2·град). Визначити кінцеві температури теплоносіїв і площу поверхні теплообміну.

Відповідь: $t_1^{"}$ = 208,6 ºС , $t_2^{"}$ = 201 ºС, F = 2071 м^2 .

15. Суха насичена водяна пара конденсується при p = 4,75 бар. Вода рухається по трубах і нагрівається від температури $T_2^{'}$ = 293 К до температури $T_2^{"}$ = 343 К, витрата води $G_в$ = 8 т/год. Визначити теплову продуктивність теплообмінника, середній температурний напір і витрату пари.

Відповідь: Q = 464 кВт, Δt = 102,9 К, $G_п$ = 0,22 кг/с .

Додаток В

ПРИКЛАД ПРОГРАМИ З ВИКОРИСТАННЯМ ООП ДЛЯ РОЗВ'ЯЗАННЯ ОБЧИСЛЮВАЛЬНОЇ ЗАДАЧІ

Файл *SqaureEq.cpp*

```
#include <stdexcept>
#include "SqaureEq.h" // див. опис в лабораторній роботі 2
double SquareEq::discrim() {
  double d = m_b * m_b - 4 * m_a * m_c;
  if (d < 0)
    throw std::invalid_argument("Error, d<0");
  return d;
}
SquareEq::SquareEq()
 : m_a(0), m_b(0), m_c(0), m_x1(0), m_x2(0) { }
SquareEq::SquareEq(double a, double b, double c)
 : m_a(a), m_b(b), m_c(c), m_x1(0), m_x2(0) { }
void SquareEq::set_a(double a) {
  if (a == 0)
    throw std::invalid_argument("Error, a=0");
  m_a = a;
}
double SquareEq::get_x1() const {  return m_x1; }
double SquareEq::get_x2() const {  return m_x2; }
void SquareEq::solve() {
  m_x1 = -m_b - discrim() / (2 * m_a);
  m_x2 = -m_b + discrim() / (2 * m_a);
}
```

Файл *main.cpp*

```
#include <iostream>  // cout, cin
#include <stdexcept> // std::invalid_argument
#include "SqaureEq.h"
using namespace std;
int main() {
  try {
    SquareEq eq1(4, 7, 2);     eq1.solve();
    cout<<"x1= "<< eq1.get_x1()<<", x2="<< eq1.get_x2();
    return 0;
  } catch (std::exception &e) {
    cerr << e.what() << std::endl;
 }
}
```

Додаток Г

ПРИКЛАД РЕАЛІЗАЦІЇ КЛАСУ *Array*

```
#include <cstdlib>
#include <ctime>
#include <stdexcept>
#include "Array.h"
Array::Array(): size_(0), storage_(0) {}
Array::Array(const size_t sz) :
    size_(0), storage_(0) {
  if (sz != 0) {
    size_ = sz;
    storage_ = new int[size_]();
  }
}
Array::Array(const Array& a) {
  if (this != &a) {
    delete[] storage_; size_ = a.size();
    storage_ = new int[size_];
    for (size_t i = 0; i < size_; ++i)
      storage_[i] = a[i];
  }
}
Array::~Array() {
  delete[] storage_;
}
int& Array::operator [](const size_t ind) {
  if (ind < 0 || ind > size_)
    throw std::out_of_range("Index out of range");
  return storage_[ind];
}
int Array::operator [](const size_t ind) const {
  if (ind < 0 || ind > size_)
    throw std::out_of_range("Index out of range");
  return storage_[ind];
}
Array& Array::operator =(const Array& a) {
  if (this != &a) {
    if (size_ == a.size_)
      for (size_t i = 0; i < size_; ++i) {
        storage_[i] = a[i];
      }
    else {
      size_ = a.size_;
      delete[] storage_;
      storage_ = new int[size_];
```

```
      for (size_t i = 0; i < size_; ++i) {
        storage_[i] = a[i];
      }
    }
  }
  return *this;
}
Array& Array::operator +=(const Array& a) {
  if (this != &a) {
    for (size_t i = 0; i < size_; ++i) {
      storage_[i] += a[i];
    }
  } else {
    for (size_t i = 0; i < size_; ++i) {
      storage_[i] += storage_[i];
    }
  }
  return *this;
}
size_t Array::size() const {
  return size_;
}
int Array::sum() const {
  int sum = 0;
  for (size_t i = 0; i < size_; ++i) {
    sum += storage_[i];
  }
  return sum;
}
int Array::operator ()() const {
  int sum = 0;
  for (size_t i = 0; i < size_; ++i) {
    sum += storage_[i];
  }
  return sum;
}
void Array::print() const {
  for (size_t i = 0; i < size_; ++i) {
    std::cout << std::setw(6) << storage_[i];
  }
  std::cout << std::endl;
}
void Array::set() {
  std::cout << "Input elem of massiv.\n";
```

Закінчення додатка Г

```
  for (size_t i = 0; i < size_; ++i) {
    std::cout << "Array[" << i << "]=";
    std::cin >> storage_[i];
  }
}
std::ostream &operator<<(std::ostream &os, const Array & a) {
  for (size_t i = 0; i < a.size_; ++i) {
    os << std::setw(6) << a.storage_[i];
  }
  return os;
}
std::istream & operator>>(std::istream & is, Array & a) {
  std::cout << "Input elem of massiv.\n";
  for (size_t i = 0; i < a.size_; ++i) {
    std::cout << "Array[" << i << "]=";
    is >> a.storage_[i];
  }
  return is;
}
void Array::init(const size_t sz) {
  size_ = sz; storage_ = new int[size_]; srand(time(0));
  for (size_t i = 0; i < size_; ++i)
    storage_[i] = rand() % 10;
}
```

Додаток Д

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ 5

Розробити програму з використанням контейнеру STL згідно з варіантом:

1) *vector;*
2) *deque;*
3) *list,*
4) *map;*
5) *multimap,*
6) *set;*
7) *multiset;*

Додаток Е

ПРИКЛАД РОБОТИ З КОМПОНЕНТАМИ *STL*

```
#include <iostream>    // std::cout
#include <vector>      // std::vector
#include <algorithm>   // std::copy
#include <functional>  // std::greater
#include <iterator>    // std::ostream_iterator
using namespace std;
int main() {
  vector<int> v; // вектор нульової довжини
  int i;
  cout << "розмір: " << v.size() << endl; // Розмір v.
  // нові елементи додаються в кінець
  for (i = 0; i < 10; i++)
    v.push_back(i);
  cout << "новий розмір: " << v.size() << endl;
  // Доступ через індекси
  for (i = 0; i < 10; i++)
    cout << v[i] << " ";
  cout << endl;
  // перший та останній елементи вектору.
  cout << "перший елемент: " << v.front() << endl;
  cout << "останній елемент" << v.back() << endl;
  // Доступ через ітератор.
  vector<int>::const_iterator p = v.begin();
  while (p != v.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;
  cout << "всього елементів == 5:" << count(v.begin(),
                                        v.end(), 5) << endl;
  sort(v.begin(), v.begin() + 5, greater<int>());
  ostream_iterator<int> out_it(cout, ", ");
  copy(v.begin(), v.end(), out_it);
  return 0;
}
```

Додаток Ж

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ 8

Розробити програму з використанням методології ООП для розрахунку фрагменту теплової схеми. Необхідні розрахункові співвідношення наведено в [1].

Варіант 1. Мережева підігрівальна установка (рис. Д Ж.1)

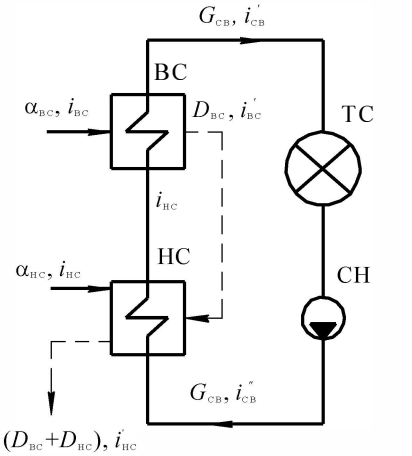


Рисунок Д Ж.1 – Розрахункова схема мережевої підігрівальної установки: ВС – верхній ступінь підігріву; НС – нижній ступінь підігріву; ТС – тепломережа; СН – мережевий насос; $\alpha_{ВС}$, $D_{ВС}$ - витрата пари у ВС; $\alpha_{НС}$, $(D_{ВС} + D_{НС})$ – витрата пари у НС; $i_{ВС}$, $i'_{ВС}$ – ентальпія пари на вході та виході з ВС; $i_{НС}$, $i'_{НС}$ – ентальпія пари на вході та виході з НС; $G_{СВ}$, $i_{СВ}$, $i'_{СВ}$ – витрата та ентальпія мережевої води.

Варіант 2. Турбоживильний насос (рис. Д Ж.2)

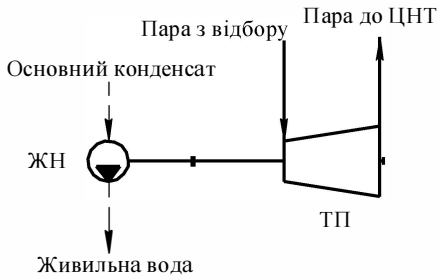


Рисунок Д Ж.2 – Розрахункова схема турбоживильного насосу: ЖН – живильний насос; ТП – турбопривід

Варіант 3. Випарник в тепловій схемі (рис. Д Ж.3)

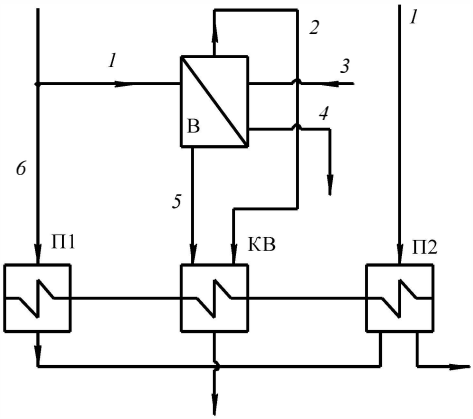


Рисунок Д Ж.3 – Розрахункова схема включення випарника в схему:
1 – підведення гарячої пари; *2* – відведення вторинної пари; *3*–підведення живильної води; *4* – продування; *5* – відведення конденсату пари, що нагріває; П1, П2 – підігрівачі; В – випарник; КВ – конденсатор випарника

Варіант 4. Підігрівач високого тиску з ОП та ОД (рис. Д Ж.4)

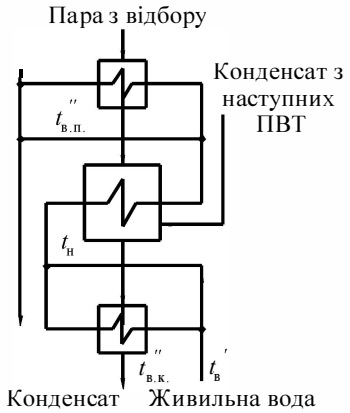


Рисунок Д Ж.4 – Розрахункова схема ПВТ з ОП та ОД

Додаток И

ПРИКЛАД ПРОГРАМИ ДЛЯ РОБОТИ З ФАЙЛАМИ

```
#include <iostream> // cin, cout
#include <iterator> //ostream_iterator
#include <algorithm> // copy
#include <vector>  // vector<int>
#include <string>  // string
#include <fstream>  // of-if-fstream.
#include <sstream>  // strinstream
#include <cstdlib>  // rand
struct Person {
  char name[25];
  unsigned int age;
  friend std::ostream&operator<<(std::ostream&stream, const
                                        Person &p) {
    return stream << p.name << " " << p.age;
  }
};
struct sPerson {
  std::string name;
  unsigned int age;
  friend std::ostream&operator<<(std::ostream&stream, const
                                    sPerson &p) {
    return stream << p.name << " " << p.age;
  }
};

int main() {
  // вихідний масив
  std::vector<int> v(10), w(10);
  // заповнюємо випадковими числами
  std::generate(v.begin(), v.end(), std::rand);
  std::ofstream off("data.dat"); //відкриття на запис
  //  std::fstream f_in_out("data.dat", std::ios_base::out);
  if (!off.is_open()) {
    std::cerr << "Error!. Can't open file\n";
    exit(EXIT_FAILURE);
  }
  // вихідний ітератор даних цілого типу
  // для файлового потоку
  std::ostream_iterator<int> out_it(off, "\n");
```

```
  std::copy(v.begin(), v.end(), out_it);
  off.close();
  //
  std::ifstream iff("data.dat");
  //  std::fstream f_in_out("data.dat", std::ios_base::in);
  if (!iff.is_open()) {
    std::cerr << "Error!. Can't open file\n";
    exit(EXIT_FAILURE);
  }
  std::string line;
  int cnt(0);
  while (getline(iff, line)) {
    std::stringstream stream(line);
    stream >> w[cnt++];
  }
  std::ostream_iterator<int> cout_it(std::cout, "\n");
  std::copy(w.begin(), w.end(), cout_it);
  //створюємо масив елементів типу Person
  Person p[5] = { { "Ivanov", 25 }, { "Petrov", 25 },
                  { "Sidorov", 25 }, { "Ivanova", 25 },
                  {"Sychov", 25 } };
  // записуємо дані одним блоком
  std::ofstream pout("person.dat", std::ios_base::binary);
  pout.write((const char*) &p, sizeof(p));
  pout.close();
  // зчитуємо дані у масив
  Person pp[5];
  std::ifstream pin("person.dat", std::ios_base::binary);
  pin.read((char*) &pp, sizeof(pp));
  pout.close();
  for (int i = 0; i < 5; ++i)
    std::cout << pp[i] << std::endl;
  //створюємо масив елементів типу sPerson
  sPerson sp[5] = { { "Ivanov", 23 }, { "Petrov", 25 },
                    {"Sidorov", 22 }, { "Ivanova", 24 },
                    { "Sychov", 27 } };
  // помилка, не можна копіювати таким чином
  // класи із членами, що є динамічними даними
  // та контейнерами STL
  std::ofstream spout("sperson.dat", std::ios_base::binary);
```

```
  spout.write((const char*) &sp, sizeof(sp));
  spout.close();
  // знову помилка, неможна читати дані у структуру,
  // в якій присутні члени, що є динамічними даними
  // та контейнерами STL
  sPerson spp[5];
  std::ifstream spin("sperson.dat", std::ios_base::binary);
  spin.read((char*) &spp, sizeof(spp));
  spin.close();
  for (int i = 0; i < 5; ++i)
    std::cout << spp[i] << std::endl;
  // для таких структур, як sPerson, потрібно
  // використовувати серіалізацію або прості текстові файли
  std::ofstream s_out("s_person.dat");
  for (int i = 0; i < 5; ++i)
    s_out << sp[i] << std::endl;
  std::ifstream s_in("s_person.dat");
  for (int i = 0; i < 5; ++i) {
    // зчитування даних до першого пробільного символу
    s_in >> spp[i].name >> spp[i].age;
    std::cout << spp[i] << "\n";
  }
  return 0;
}
```

Список літератури

1. Литвиненко О. О. Методичні вказівки до виконання випускної роботи бакалавра «Розрахунок принципових теплових схем турбоустановок малої потужності» для студентів напряму 6.050601 «Теплоенергетика», спеціалізації 6.05060102 «Теплофізика» / уклад. О. О. Литвиненко, О. М. Іванченко, І. О. Михайлова. – Х. : НТУ «ХПІ», 2015. – 40 с.
2. Краснощеков Е. А. Задачник по теплопередаче / Е. А. Краснощеков, А. С. Сукомел .– М. : Энергия, 1969. – 288 с.
3. Бойко А. В. Аэродинамика проточной части паровых и газовых турбин: расчеты, исследования, оптимизация, проектирование / А. В. Бойко, А. В. Гаркуша – Х.: ХГПУ, 1999. – 360 с.
4. Бойко А. В. Многокритериальная многопараметрическая оптимизация проточной части осевых турбин с учетом режимов эксплуатации / А. В. Бойко, А. П. Усатый, А. С. Руденко. – Х. : НТУ «ХПІ», 2014. – 220 с.
5. Грицюк Ю. І. Об'єктно-орієнтоване програмування мовою С++ : навч. посіб. / Ю. І. Грицюк, Т. Є. Рак – Львів : Вид-во ДУ БЖД, 2011. – 404 с.
6. Рамбо Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – СПб. : Питер, 2007. – 544 с.
7. Прата С. Язык программирования С++. Лекции и упражнения / С. Прата ; 6 изд. ; пер. с англ. – М. : Издат. дом «Вильямс», 2012. – 1248 с.
8. Страуструп Б. Программирование: принципы и практика использования С++ / Б. Страуструп ; пер с англ. – М. : Издат. дом «Вильямс», 2011. – 1248 с.
9. Ласло М. Вычислительная геометрия и компьютерная графика на С++/ М. Ласло – М. : БИНОМ, 1997. – 304 с.
10. cppreference.com – C++ reference [Електронний ресурс]. – Режим доступу : http://en.cppreference.com/w/
11. cplusplus.com – The C++ Resource Network [Електронний ресурс]. – Режим доступу : http://www.cplusplus.com/
12. GCC, the GNU Compiler Collection [Електронний ресурс]. – Режим доступу : https://gcc.gnu.org/
13. Code::Blocks [Електронний ресурс]. – Режим доступу : http://www.codeblocks.org/
14. Visual Studio Express [Електронний ресурс]. – Режим доступу : https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

ЗМІСТ

Вступ 3
Загальна постановка задачі 4
Лабораторна робота 1. Розробка класів елементів теплових схем 5
Лабораторна робота 2. Розробка програми для розв'язання задачі теплофізики 8
Лабораторна робота 3. Динамічні масиви для зберігання елементів розрахункових теплових схем 10
Лабораторна робота 4. Узагальнене програмування. Шаблони 13
Лабораторна робота 5. Контейнери та алгоритми *STL* 16
Лабораторна робота 6. Успадкування 19
Лабораторна робота 7. Поліморфізм 22
Лабораторна робота 8. Розробка програми для розрахунку фрагменту теплової схеми 24
Додатки 28
Додаток А. Завдання до лабораторної роботи 1 28
Додаток Б. Завдання до лабораторної роботи 2 28
Додаток В. Приклад програми використанням ООП для розв'язання обчислювальної задачі 32
Додаток Г. Приклад реалізації класу *Array* 33
Додаток Д. Завдання до лабораторної роботи 5 35
Додаток Е. Приклад роботи з компонентами *STL* 36
Додаток Ж. Завдання до лабораторної роботи 8 37
Додаток И. Приклад програми для роботи з файлами 39
Список літератури 42

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з курсу
«Технології програмування в задачах теплофізики»
для студентів спеціальностей 8.05060102 «Теплофізика»
та 8.05060402 «Турбіни»

Укладачі: САВЧЕНКО Володимир Миколайович
ЛИТВИНЕНКО Оксана Олексіївна
ШАПОВАЛОВА Наталія Сергіївна

Відповідальний за випуск проф. А. В. Бойко

Роботу до видання рекомендував проф. О.В. Потетенко

В авторській редакції

План 2015 р., поз. 239

Підп. до друку 18.11.2015. Формат 60х84 1/16. Папір офсетний.
Друк – ризографія. Гарнітура Times New Roman. Ум. друк. арк. 2,55.
Наклад 50 прим. Ціна договірна.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК №3657 від 24.12.2009 р.
61002, Харків, вул. Фрунзе, 21.