МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Волинський національний університет імені Лесі Українки
Кафедра комп’ютерних наук та кібербезпеки

Глинчук Л. Я., Гришанович Т.О.

Підручник

ПРОГРАМУВАННЯ

Для студентів спеціальностей
122 «Комп’ютерні науки»
113 «Прикладна математика»
014 «Середня освіта (Інформатика)»

Луцьк 2022

УДК 004.42(075.8)

Г 54

Рекомендовано до друку Вченою радою
Волинського національного університету імені Лесі Українки
(протокол №4 від 31.03.2022 р.)

Рецензенти:
Чернящук Н. Л., доктор педагогічних наук, професор, завідувач кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету.

Кузьмич О. І., кандидат фізико-математичних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету.

Хомяк М. Я., кандидат фізико-математичних наук, доцент, завідувач кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки.

Програмування: підручник [Електронний ресурс] / укладач Л. Я. Глинчук, Т.О. Гришанович; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 3 201 КБ). – Луцьк: ВНУ ім. Лесі Українки, 2022. – 160 с.

Підручник містить теоретичний матеріал, додатковий теоретичний матеріал для самостійного вивчення деяких тем, завдання для лабораторних робіт, завдання для модульного контролю та приклади завдань для підсумкового контролю. Теми укладені згідно програми дисципліни «Програмування» для студентів ІІ-го курсу спеціальностей 122 «Комп'ютерні науки», 113 «Прикладна математика» та 014 «Середня освіта (Інформатика)». Розраховані на семестр навчання згідно годин навчального плану. Підручник розроблено як логічне продовження вивчення дисципліни «Програмування» після І-го курсу, а саме, вивчення мови програмування С++ і особливості програмування графіки. За умови зміщення та пришвидшення вивчення мови програмування С++, даний матеріал можна використовувати і на І-му курсі.

Підручник корисний для викладачів, які працюють зі студентами суміжних напрямів та студентів, які навчаються за галуззю знань 12 «Інформаційні технології». Буде цікавий усім бажаючим для розвитку додаткових вмінь та навичок з використання мови програмування С++. Для студентів вузів, коледжів, учнів.

ЗМІСТ

ТЕМИ ЛЕКЦІЙ

Тема 1. Дружні функції та класи

Як було зазначено раніше, члени (елементи) класу можуть бути *закритими* та *відкритими*. До закритих елементів класу доступ існує тільки всередині класу. Однак, інколи потрібно, щоб елемент класу залишався закритим, а деякі зовнішні функції та методи мали до нього доступ. **Такі зовнішні функції, які мають доступ до закритих елементів класу, називають дружніми функціями**.

Щоб оголосити функцію **як дружню для класу**, потрібно вказати прототип цієї функції в описі класу, але перед цим вказати службове слово friend. Розглянемо приклад використання дружньої функції.

Приклад 1.

```
#include <iostream>
using namespace std;
// Клас з закритим полем:
class MyClass {
   double x;
public:
   MyClass (double z)
   {       x=z;    }
   // Дружня функція
   friend void show (MyClass obj);
};
// Опис дружньої функції
void show (MyClass obj) {
 cout << "x = " << obj.x << endl;
 }
int main()
```

```
{
   MyClass a(10);
   // Дружня функція має доступ до закритих елементів класу
   show(a);
   return 0; }
```

В даному випадку поле x класу MyClass є закритим, тому формально отримати доступ до цього поля класу можна тільки всередині класу. Для заповнення цього поля передбачено конструктор, якому передається один аргумент. Однак, більше ніяких методів для отримання доступу в класі немає. Значення поля x буде виводитися з допомогою зовнішньої функції show (). Щоб функція мала доступ до закритого поля, вона оголошена в класі як дружня, тобто в описі класу добавлено рядок friend void show (MyClass obj). Завдяки тому, що функція дружня до класу, коректною є команда show (a), яка відображає значення закритого поля.

Хоча дружня функція не є елементом класу, її прототип в класі вказано.

Концепція дружніх класів може видатися штучною, проте це не так. Уявимо собі випадок, коли є два класи з закритим полями і необхідно отримати доступ до полів обох класів. Це один з тих випадків, коли доцільним є застосування дружніх функцій.

Розглянемо **приклад 2**.

```
using namespace std;
// Анонс класу
class B;
// Клас з закритим полем:
class A {
   double x;
public:
   A (double z)
   {       x=z;    }
 // Дружня функція з двома аргументами
```

```
 friend double summa (A a, B b);
} a(3.5);
// Клас з закритим полем:
class B {
   double y;
public:
   B (double z)
   {       y=z;    }
 // Дружня функція з двома аргументами
 friend double summa (A a, B b);
} b(2.3);
double summa (A a, B b)
{    return a.x+b.y;}
int main()
{    // Виклик дружньої функції
   cout << "Zn = " << summa(a,b) << endl;
   return 0;}
```

Тут є два класи (A та B), у кожного з яких є закрите поле типу double. Функцією summa () обчислюється сума закритих полів об’єктів, переданих їй в якості аргументів. Перший аргумент функції – об’єкт класу A, другий аргумент функції – об’єкт класу B. Щоб функція мала доступ до полів обох класів, в кожному класі ця функція оголошена як дружня. Крім того, оскільки вперше прототип дружньої функції в програмному коді з’являється до опису класу B, тому до цього місця у програмі наведений анонс класу B. Такий анонс необхідно для того, щоб ідентифікувати як клас оголошення типу у другому аргументі функції. В результаті роботи програми отримали Zn = 5.8.

Дружніми по відношенню до класу можуть бути окремі методи іншого класу або цілий клас. Наведемо приклад, де всі методи дружнього класу мають доступ до закритих полів та методів іншого класу.

Приклад 2. Містить використання дружнього методу.

```
#include <iostream>
using namespace std;
// Анонс класу
class B;
// Клас з закритим полем та методом:
class A {
   double x;
public:
   A (double z)
   {       x=z;     }
 double summa (B b);
} a(3.5);
// Клас з закритим полем та дружнім методом:
class B {
   double y;
public:
   B (double z)
   {        y=z;     }
 // Дружній метод:
 friend double A::summa (B b);
} b(2.3);
int main()
{    // Виклик дружнього методу
   cout << "Zn = " <<a.summa(b) << endl;
   return 0;}
double A::summa (B b)
{     return x+b.y; }
```

Функціональність програми в порівнянні з попереднім прикладом не змінилася, але тепер сума полів класів обчислюється з допомогою методу класу A.

У методу summa () класу A вже один аргумент (об’єкт класу B) – другий об’єкт методу передавати не потрібно, оскільки, з цього об’єкту викликається метод і доступ до об’єкту у методі є автоматично.

В класі B командою friend double A::summa (B b) метод summa () класу A оголошено як дружній. Звертаємо увагу на 2 обставини: по перше, при оголошенні дружнього методу явно вказано, до якого класу він належить (перед ім’ям методу вказано назву класу); по друге, опис методу summa () повинно бути тільки після того, як описані класи, об’єкти яких використовуються в методі – в даному випадку це клас B.

Наведемо приклад програми де один клас дружній до іншого. При цьому в дружній клас добавлено ще один метод product () для обчислення добутку закритих полів класу.

Приклад 3.

```
#include <iostream>
using namespace std;
class B;
class A {
   double x;
public:
   A (double z)
   {       x=z;    }
 double summa (B b);
 double product (B b);
} a(3.5);
class B {
   double y;
public:
   B (double z)
   {       y=z;    }
 friend class A;
```

```
} b(2.3);
int main()
{
   cout << "Zn S = " <<a.summa(b) << endl;
   cout << "Zn D = " <<a.product(b) << endl;
   return 0;
}
double A::summa (B b)
{    return x+b.y;}
double A::product (B b)
{    return x*b.y;}
```

В класі B є інструкція friend class A, якою клас A оголошується як дружній до класу B. В цьому випадку всі методи класу A мають доступ до елементів класу B, в тому числі і закритих. Тому коректними є команди a.summa(b) та a.product(b) в головному методі програми.

Розглянемо ще один приклад дружніх класів.

Приклад 4.

```
#include <iostream>
using namespace std;
class B;
class A {
friend class B;
private:
 int zn = 100;
 } ;
class B {
public:
    void VZn (A a)
   {
      a.zn -= 20;
```

```
        cout << "Zn = " << a.zn << endl;
    }
        void VZn2 (A a)
    {
        a.zn = 0;
        cout << "Zn = " << a.zn << endl;
    }
void VZn3 (A &a)
     {      a.zn += 30 ;
        cout << "Zn = " << a.zn << endl;
    }
};
int main()
{
    A a1;
    B b1;
    b1.VZn(a1);
    b1.VZn2(a1);
    b1.VZn(a1);
    b1.VZn3(a1);
   cout << endl;
    A a2;
    b1.VZn(a2);
    return 0;
}
```

До класу A оголосили дружнім клас B. В класі B описали метод VZn, де знаходили значення змінної zn. Результат програми Zn = 80.

Якщо добавити ще один метод, void VZn2 (A a), результат програми Zn = 80, Zn = 0. Якби клас не був дружнім, то дружніх функцій потрібно було 2, а так оскільки клас B дружній то методи класу можна використовувати.

Якщо у головній програмі ще раз дописати b1.VZn(a1); – отримаємо результат Zn = 80, Zn = 0, Zn = 80. А повинно було б бути останнє значення Zn = 60. Оскільки передаємо об’єкт. Якщо ми добавимо & (посилання) у функцію, результат зміниться.

Основні теоретичні відомості для дружніх функцій:

- дружня функція може бути розміщена в будь якому полі класу – private, public або protected. Та при будь яких обставинах буде мати доступ до private-елементів класу і, навіть якщо вона сама знаходиться в полі private, до її можна буде звернутися в не класу, не використовуючи спеціальних методів;
- коли ми визначаємо дружню функцію, елементи класу необхідно явно передавати у вигляді параметрів функції. Так як вона не являється компонентом класу, вона не отримує вказівник this;
- у вигляді параметру, в дружню функцію необхідно передати вказівник або посилання на об’єкт класу. Інакше вона не побачить дані якого класу їй прийняти та опрацювати;
- функція може використовуватися, як дружня до декількох класів;
- викликаються дружні функції як звичайні функції, тобто не використовується такий спосіб – об’єкт класу.функція (). Після внесення всіх необхідних параметрів в її, при виклику, вона сама побачить з елементами якого класу і об’єкту потрібно працювати.

Основні теоретичні відомості для дружніх класів:

- щоб оголосити клас дружнім, в тілі класу, який має бути дружнім, перед іменем дружнього класу необхідно використати службове слово C++ – friend;
- як і у випадку з дружніми функціями, немає різниці, в якому полі класу ми оголосимо дружній клас – private, public або protected. Ми все одно зможемо звертатися до його методів з головної функції main ();

- якщо визначення дружнього класу розміщене нижче визначення класу, що пропонує дружбу, то оголосити дружній клас потрібно вище. Це допоможе уникнути помилок при компіляції;
- коли клас оголошено дружнім, всі його методи також стають дружніми до того класу в якому він оголошений. При цьому методи класу, який дозволив дружбу не мають доступу до елементів дружнього класу;
- не потрібно зловживати та оголошувати багато дружніх класів.

Питання для самоконтролю

1. До яких елементів класу доступ існує тільки всередині класу?
2. Які функції називаються дружніми?
3. Як оголосити функцію дружню до класу?
4. Що таке прототип функції?
5. Яку можливість в описі класу дає даний рядок «friend void show (MyClass obj)»?
6. Опишіть випадок коли доцільним є застосування дружніх функцій?
7. В якому полі класу може бути розміщена дружня функція?
8. Що таке анонс класу та як він оголошується?
9. У яких випадках потрібно оголошувати анонс класу?
10. Чи можуть бути окремі методи класу дружніми по відношенню до іншого класу?
11. Наведіть приклад, де всі методи дружнього класу мають доступ до закритих полів та методів іншого класу.
12. Як оголосити дружній метод? Наведіть приклад коду.
13. Якщо у класі B є інструкція friend class A, то що вона означає?
14. У якому випадку всі методи класу A мають доступ до елементів класу B, в тому числі і закритих?

15. Коли до функції можна буде звернутися в не класу, не використовуючи спеціальних методів?
16. Чи може дружня функція отримувати вказівник this? Обґрунтуйте відповідь.
17. Як викликаються дружні функції?
18. В якому полі класу можна оголосити інший клас як дружній?
19. Чи можемо ми звертатися до методів дружнього класу з головної функції main ()?
20. Коли клас оголошено дружнім, всі його методи також …до того класу в якому він оголошений.

Тема 2. Шаблони функцій та шаблони класів

Шаблони функцій – це інструкції, згідно яких створюються локальні версії шаблонної функції для визначеного набору параметрів та типів даних.

Розглянемо приклад, нам потрібно запрограмувати функцію, яка буде виводити на екран елементи масиву. Щоб написати таку функцію ми повинні знати тип даних масиву, який будемо виводити. А якщо потрібно вивести одразу декілька масивів з різними типами даних? Щоб функція виводила масиви типу int, double, float, char.

Ми можемо виконати це завдання написавши 4 функції, які виконують одне і теж, але для різних типів даних. Тобто можна використати перевантаження функції.

Приклад 1.

```
#include <iostream>
using namespace std;
void printArray(const int * array, int count)
{
   for (int ix = 0; ix < count; ix++)
```

```
        cout << array[ix] << "   ";
    cout << endl;
}
void printArray(const double * array, int count)
{
    for (int ix = 0; ix < count; ix++)
      cout << array[ix] << "   ";
    cout << endl;
}
void printArray(const float * array, int count)
{
    for (int ix = 0; ix < count; ix++)
         cout << array[ix] << "   ";
    cout << endl;
}
void printArray(const char * array, int count)
{
    for (int ix = 0; ix < count; ix++)
        cout << array[ix] << "   ";
    cout << endl;
}
int main()
{
    const int is=3, ds=3, fs=2, cs=5;
    int i_arr[is]={34,45,67};
    double d_arr[ds]={12.4, 56.7, 23.7};
    float f_arr[fs]={1.23, 2.45};
    char c_arr[cs]={"Hi!!"};
    cout << "Masuv int:";
    printArray(i_arr,is);
```

```
    cout << "Masuv double:";
    printArray(d_arr,ds);
    cout << "Masuv float:";
    printArray(f_arr,fs);
    cout << "Masuv char:";
    printArray(c_arr,cs);
    return 0;
}
```

Результат роботи – буде виведено всі 4 масиви.

З завданням ми справилися, але якщо потрібно буде відсортувати всі масиви, тоді потрібно буде для кожного типу даних масиву писати свою функцію, тобто це багато коду. Тому для таких завдань будемо використовувати шаблони функцій.

Ми створюємо один шаблон в якому описуємо всі типи даних. Таким чином не потрібно буде писати 4 функції.

Приклад 2.

```
template <typename T>
#include <iostream>
using namespace std;
template <typename T>
void printArray(const T * array, int count)
{
    for (int ix = 0; ix < count; ix++)
        cout << array[ix] << "   ";
    cout << endl;
}
int main()
{
    const int is=3, ds=3, fs=2, cs=5;
    int i_arr[is]={34,45,67};
```

```
    double d_arr[ds]={12.4, 56.7, 23.7};
    float f_arr[fs]={1.23, 2.45};
    char c_arr[cs]={"Hi!!"};
    cout << "Masuv int:";
    printArray(i_arr,is);
    cout << "Masuv double:";
    printArray(d_arr,ds);
    cout << "Masuv float:";
    printArray(f_arr,fs);
    cout << "Masuv char:";
    printArray(c_arr,cs);
    return 0;
}
```

Результат роботи програми такий же самий, але код менший.

Всі шаблони функцій починаються зі слова template, після якого йдуть вуглові дужки, в яких перераховується список параметрів. Кожному параметру повинно передувати зарезервоване слово class або typename.

Тобто:

Template <class T>

Або

Template <typename T>

Або

Template <typename T1, typename T2>

Ключове слово typename говорить про те, що в шаблоні буде використовуватися деякий вбудований тип даних.

В такому випадку код програми менший, але це не означає, що пам’яті програма буде використовувати менше, компілятор сам створює шаблони функцій тому і пам’яті використовує стільки, скільки б ви самі написали всі функції, як у прикладі 1.

Розглянемо тепер шаблон класу. Він описується так як ми раніше казали, тобто шаблон класу буде виглядати отак:

```
Template <typename T>
class Name
{
 // тіло шаблону класу
}
```

Де T – це параметр шаблону класу, який може приймати будь який із вбудованих типів даних, те, що нам потрібно. Якщо параметр шаблону класу користувацького типу, наприклад типу Array, де Array – це клас, що описує масив, то шаблон класу буде мати вигляд:

```
Template <class T>
class Name
{
 // тіло шаблону класу
}
```

Створимо шаблон класу для стеку. Де стек – структура даних, де розміщені елементи одного типу. В стек можна поміщати дані та видаляти. Елемент, що добавляється, йде зразу у вершину і видаляються елементи теж починаючи з вершини. В шаблоні класу стек необхідно створити основні методи:

Push – добавити елемент в стек;

Pop – видалити елемент із стеку

Приклад 3.

```
#include <iostream>
using namespace std;
template <class type>
class Stak
{
  private:
```

```
    type st[100];
    int top;
  public:
   Stak ()
   {
      top = -1;
   }
   void push (type var)
   {
    st[++top] = var;
   }
  type pop()
   {
     return st[top--];
   }
};
int main()
{
   Stak <int> s1;
   s1.push(11);
   s1.push(22);
   s1.push(33);
   cout<< "\n 1\t" << s1.pop();
   cout<< "\n 2\t" << s1.pop();
   cout<< "\n 3\t" << s1.pop() << endl;
   Stak <float> s2;
   s2.push(11.45);
   s2.push(22.89);
   s2.push(33.45);
   cout<< "\n 1\t" << s2.pop();
```

```
    cout<< "\n 2\t" << s2.pop();
    cout<< "\n 3\t" << s2.pop();
    return 0;
}
```

Питання для самоконтролю

1. Що таке шаблон функції?
2. Наведіть приклад, коли доцільно використати шаблон функції.
3. Яке службове слово показує, що оголошено шаблон функції?
4. Що записують після службового слова в шаблоні функції?
5. Якщо використовувати шаблони, то код програми збільшиться чи зменшиться? Чому?
6. Наведіть приклад використання шаблону функції, відмінний від прикладу в лекції.
7. Як впливає на пам'ять використання шаблону функції?
8. Який загальний вигляд має шаблон класу?
9. Коли доцільно використовувати шаблон класу?
10. Наведіть приклад шаблону класу, відмінний від прикладу в лекції.

Тема 3. Шаблон функції з різними варіантами змінних та перевантаження шаблонів

Розглянемо приклади.

Приклад 1. Узагальнена функція або шаблон з одним аргументом

```
#include <iostream>
using namespace std;
template <class X>
```

```
void show(X arg)
{   cout << "Value is " << arg << endl; }
int main()
{   int n=5;
   double x=3.6;
   char s='a';
   show(n);
   show(x);
   show(s);
   return 0; }
```

Приклад 2. Узагальнена функція з локальними змінними узагальненого типу.

```
#include <iostream>
using namespace std;
template <class X>
 X Addone(X arg)
{ X t;
  t=arg+1;
  return t; }
int main()
{    int n=5;
   double x=3.6;
   char s='a';
   cout << n << "+ 1 =" << Addone(n)<< endl;
   cout << x << "+ 1 =" << Addone(x)<< endl;
   cout << s << "+ 1 =" << Addone(s)<< endl;
   return 0; }
```

В узагальнених функціях може використовуватися декілька узагальнених типів. Ідентифікатори узагальнених типів в цьому випадку

перераховуються через кому у блоці де вуглові дужки, а перед кожним з них вказується ключове слово class.

Приклад 3. Використання узагальненої функції з 2-ма аргументами.

```
#include <iostream>
using namespace std;
template <class X, class Y>
void show(X x, Y y)
{  cout << "1-st argument:" << x << endl;
   cout << "2-d argument:" << y << endl; }
int main()
{  show(1, 'a');
   show("TEXT", 3.5);
   return 0; }
```

Приклад 4. Узагальнений клас зі службовим словом typename

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
class MyClass
{
   public:
   MyClass (T value)
   {
      this-> value = value;
   };
    void DataTypeSize(T value)
    {
       cout << sizeof(value) << endl;
       }
   private:
```

```
    T value;
    };
int main()
{
    int a = 5;
    MyClass <int> c(a);
    c.DataTypeSize(a);
   double b = 5.4567;
    MyClass <double> d(b);
    d.DataTypeSize(b);
  }
```

Приклад 5. Узагальнений клас з використанням узагальненого типу клас.

```
#include <iostream>
#include <string>
using namespace std;
class Point
{  int x, y, z;
   public:
   Point()
   {
      x = y = z = 0;
   }
   Point(int x, int y, int z)
   {
      this-> x = x;
      this-> y = y;
      this-> z = z;
   }
};
```

```
template <typename T>
class MyClass
{
   public:
   MyClass (T value)
   {
      this-> value = value;
   };
    void DataTypeSize(T value)
    {
        cout << sizeof(value) << endl;
        }
   private:
   T value;
   };
int main()
{
   Point a;
   MyClass <Point> c(a);
   c.DataTypeSize(a);
    return 0;
  }
```

Можна замість цілого типу у класі Point оголосити дійсний чи інший тип і тоді, функція рахуватиме розмір у байтах для того типу, що ви задали.

Приклад 6. Узагальнений клас, що використовує 2 узагальнені типи, один з яких клас.

```
#include <iostream>
#include <string>
using namespace std;
class Point
```

```
{  int x, y, z;
   public:
   Point()
   {
      x = y = z = 0;
   }
   Point(int x, int y, int z)
   {
      this-> x = x;
      this-> y = y;
      this-> z = z;
   }
};
template <typename T1, typename T2>
class MyClass
{
   public:
   MyClass (T1 value, T2 value2)
   {
      this-> value = value;
      this-> value2 = value2;
   };
    void DataTypeSize()
    {
        cout << "Value = " << sizeof(value) << endl;
        cout << "Value2 = "<< sizeof(value2) << endl;
        }
   private:
   T1 value;
   T2 value2;
```

```
    };
int main()
{
    int a = 2;
    Point p;
    MyClass <int, Point> c(a,p);
    c.DataTypeSize();

    int n = 4, m = 8;
    MyClass <int, int> f(n,m);
    f.DataTypeSize();
    return 0;
    }
```

Перевантаження узагальнених функцій (шаблонів)

Для узагальнених функцій можна виконувати явне перевантаження, яке називається **явною спеціалізацією**.

Приклад 7.

```
#include <iostream>
using namespace std;
template <class X>
void change(X &a, X &b)
{  X t;
   t=a;
   a=b;
   b=t;     }
void change(int &a, int &b)
{  int t;
   t=a;
   a=b+1;
   b=t+1; }
```

```
int main()
{  double x=2.3, y=4.5;
   int m=6, n=8;
   change(x,y);
   cout << "x=" << x << endl;
   cout << "y=" << y << endl;
   change(m,n);
   cout << "m=" << m << endl;
   cout << "n=" << n << endl;
   return 0; }
```

Для узагальнених функцій можна також перевантажувати не тільки їх конкретні реалізації, але і перевантажувати цілком весь шаблон узагальненої функції.

Приклад 8.

```
#include <iostream>
using namespace std;
template <class X>
X MySum(X a, X b)
{     return a+b; }
template <class X>
X MySum(X a)
{     return a+1; }
int main()
{ int n=2, m=4;
  double x=4.2, y=3.4;
  char s='a';
  cout << MySum(n,m) << endl;
  cout << MySum(x,y) << endl;
  cout << MySum(s) << endl;
  return 0; }
```

Питання для самоконтролю

1. Що таке шаблон функції?
2. Наведіть приклад шаблону з одним аргументом.
3. Поясніть суть шаблону з одним аргументом.
4. Наведіть приклад шаблону (узагальненої функції) з локальними змінними узагальненого типу.
5. Поясніть суть шаблону з локальними змінними узагальненого типу.
6. Яке ключове слово вказується перед кожним узагальненим типом у шаблоні функції?
7. Розгляньте приклади 4 та 5, чим вони відрізняються?
8. Наведіть інші приклади використання узагальненого (шаблону) класу з узагальненим типом клас.
9. Опишіть особливості узагальненого класу з використанням узагальненого (шаблону) типу клас.
10. Прокоментуйте приклад, де узагальнений клас використовує 2 узагальнені типи, один з яких клас, вкажіть особливості.
11. Що таке перевантаження функцій?
12. Чи можна виконувати перевантаження для шаблонів функцій? Обґрунтуйте.
13. Прокоментуйте наведені приклади 7 та 8, які особливості ви побачили в реалізації?
14. Наведіть інші приклади використання перевантаження шаблонів функцій.

Тема 4. Графіка у C++: налаштування, основні принципи

У C++ мова окремо, графіка окремо, тому є книги по мові програмування, а є по програмуванню графіки. Графіка налаштовується під

різні ОС. Почнемо з графіки 2D. По іншому вона ще називається черепашкова графіка під ОС DOS. Оскільки, компілятор працювати без допоміжних файлів не буде, потрібно йому допомогти. Освоївши цей вид графіки, ви без труднощів зможете засвоїти графіку під ОС Windows. Принципових відмінностей немає, головне зрозуміти концепцію.

Графічний екран дисплея складається з точок, які можна засвічувати певним кольором чи гасити, у результаті чого на екрані утворюється деяке зображення. Точки називаються пікселями. Кількість точок на екрані може бути різною. Це залежить від характеристик і налаштування монітора. Розглянемо екран, який має 640 точок у горизонтальному (x) напрямку і 480 у вертикальному (y). Відлік точок ведуть з лівого верхнього кута екрана. Кожна точка характеризується двома координатами (x,y). Приклади розташування деяких точок на екрані показано на рис.4.1.

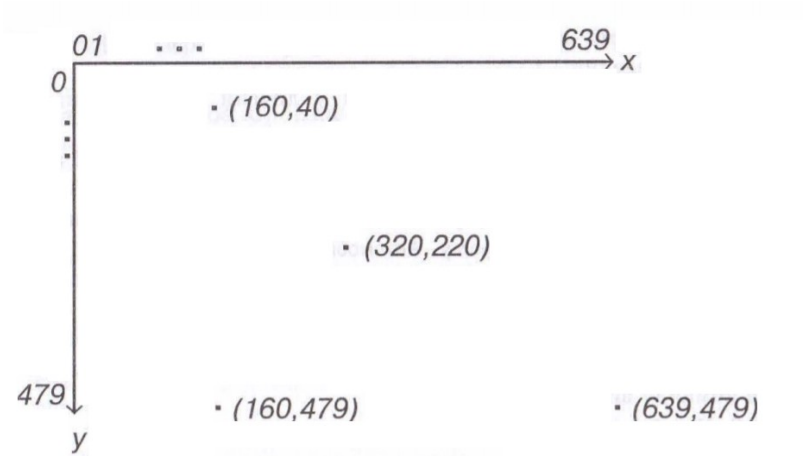


Рис.4.1. - Координати точок на графічному екрані 640x480

Визначити кількість точок уздовж осей ОХ та ОУ можна відповідно за допомогою команд getmaxx() та getmaxy() з графічної бібліотеки.

Програми, що виконують графічні побудови складаються з викликів стандартних графічних функцій, які описані у бібліотечному файлі graphics.h. (Глинський)

Як перейти у графічний режим?

Для того щоб створити зображення та перейти у графічний режим, спочатку треба записати наступний код:

у розділі бібліотек підключити бібліотеку graphics.h, решту бібліотеки за необхідністю, тобто

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
main ()
{  int gdriver = DETECT, gmode, errorcode;
// Задання графічного режиму
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode !=grOk)
{printf("Graphics error: %s\n", grapherrormsg(errorcode));
getch();
exit(1); }
--- щось малюємо ---
getch();
closegraph();
return 0;
}
```

Якщо графічний режим налаштовано правильно, то після виконання (компіляції) даного коду повинно запуститися 2 чорних екрани: один Windows BGI, інший – назва файлу.exe.

Опишемо основні функції, функція initgraph (ініціалізує графіку) задає і режим і дозвіл та вказує шлях до драйверу. Перший аргумент вказує тип відеоадаптера. Він приймає числове значення, тобто аргумент gdriver:

Тип відеоадаптера	Значення аргумента:
Авторозпізнавання	DETECT або 0
CGA	1
MCGA	2

EGA	3
EGA64	4
EGA-Mono	5
Зарезервовано	6
Mono Hercules	7
ATT400	8
VGA	9
PC3270	10

Про всі ці відеоадаптери можна прочитати в книгах по ПК чи технічному забезпеченню ПК. Але всі монітори на сьогодні лишаються вірними стандарту VGA (256 кольорів) та його розширенню. Драйвер, що використовується в даному графічному режимі EGAVGA.BGI дозволяє реалізувати тільки 16 кольорів, більше він не дозволить.

Другий аргумент – графічний режим, іноді його ще називають модою.

Режим	Значення аргумента
VGA 640x200 16 кольорів	0
VGA 640x350 16 кольорів	1
VGA 640x480 16 кольорів	2
EGA 640x200 16 кольорів	0
EGA 640x350 16 кольорів	1

Третій аргумент – шлях до файлу EGAVGA.BGI. Шлях вказується у лапках у вигляді назви диска, усіх папок, двокрапку та слеші або при правильному налаштуванні знаходиться автоматично[6] .

Функція graphresult() – повертає код помилки, якщо неможливо задати графічний режим; 0 – у разі задання.

Функція getch() – дає можливість затримати та показати зображення на графічному екрані.

Функція closegraph() – закриває графічний режим.

Як уже згадувалося, існує 16 стандартних кольорів, що на рис.4.2:

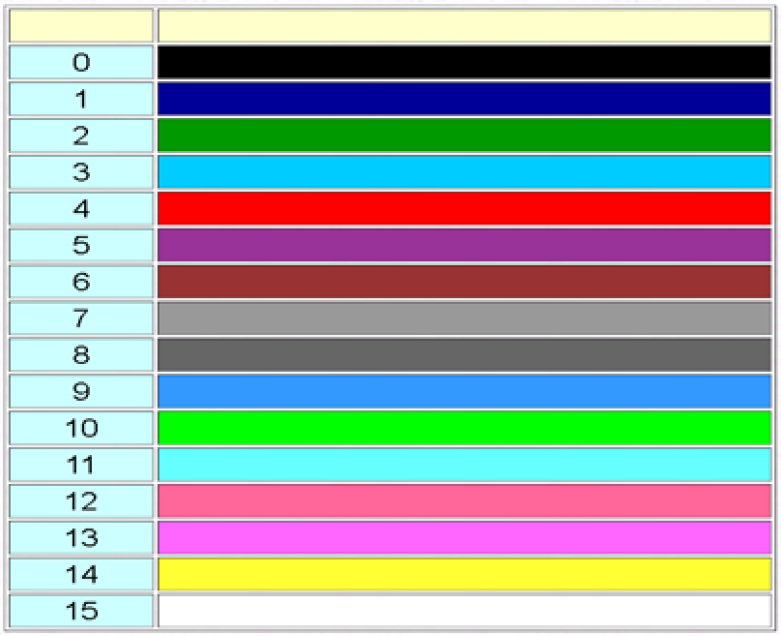


Рис.4.2. – Кольори для графічного режиму

Функція setcolor(int color) – міняє колір на вказаний в дужках і далі все малюється цим кольором. Аналогічна функція setbkcolor(int color) – міняє колір фону, на колір малюнка не впливає. Який колір встановлено можна побачити використавши функції відповідно: getcolor(), getbkcolor().

Додатково ідею створення графічних програм пропоную переглянути у відео ролику Тема 9. Графіка у с++. [7]

Для того, щоб налаштувати графічний режим з використанням бібліотеки graphics.h потрібно зробити декілька кроків з перенесенням та підключенням необхідних файлів.

1. Для версії Code::Blocks 17.12 з джерела [8] завантажити та розпакувати Graphics-Library-master. В архіві містяться наступні файли, рис.4.3:

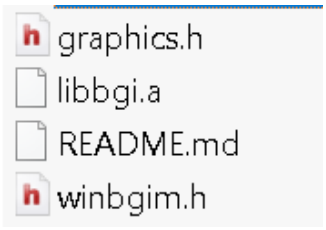


Рис. 4.3. – Файли з архіву

2. Слідуючи інструкції за вказаним джерелом розмістити файли у відповідні папки, тобто:

   a. Скопіювати та вставити файли graphics.h и winbgim.h в папку include директорії Code::Blocks.

   Шлях: C: \ Program Files (x86) \ CodeBlocks \ MinGW \ include.

   b. Скопіювати і вставити файл libbgi.a в папку lib в Code: Blocks.

   Шлях: C: \ Program Files (x86) \ CodeBlocks \ MinGW \ lib.

3. В додатку Code::Blocks, перейти у вкладку Налаштування/ Компілятор, тобто:

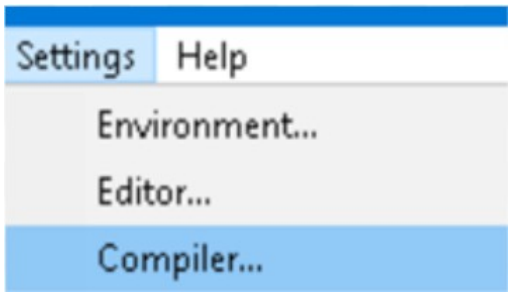


Рис. 4.4. – Налаштування компілятора

Далі потрібно перейти на вкладку Linker Settings. В Link Libraries додайте та перейдіть до C: \ Program Files (x86) \ CodeBlocks \ MinGW \ lib \ та виберіть libbgi.a.

Вставте наступний текст у вкладку Other linker options (тобто з правої сторони):

-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32.

Після попередніх дій картинка повинна мати вигляд:

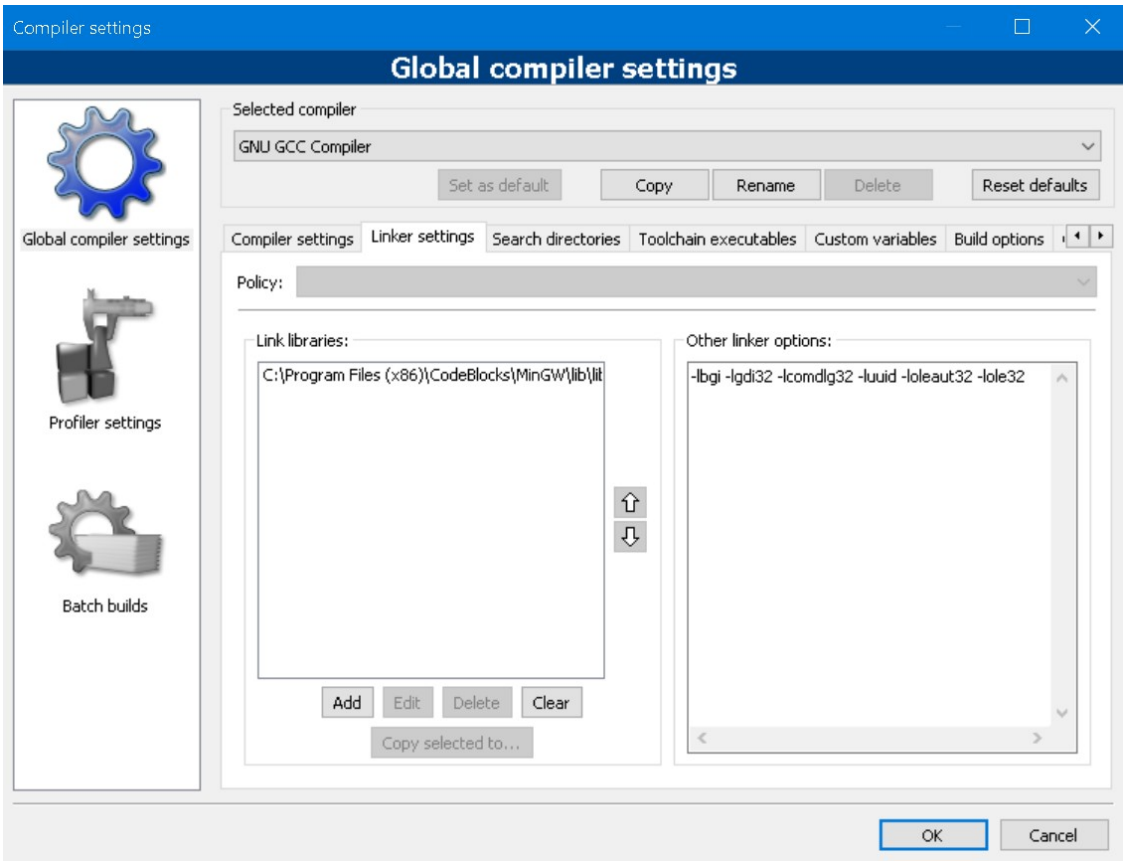


Рис. 4.5. – Прописування бібліотек

Збережіть налаштування та перезавантажте додаток.

Основні функції для графічних побудов:

putpixel(x, y, <колір>) – висвітлює точку (x, y) заданим кольором;

line(x1, y1, x2, y2) – рисує лінію між двома точками;

lineto(x, y) – рисує лінію від поточної точки до точки (x, y);

bar(x1, y1, x2, y2) – рисує зафарбований прямокутник;

bar2d(x1, y1, x2, y2, <об'ємна глибина>, n) – рисує паралепіпед;

rectangle(x1, y1, x2, y2) – рисує прямокутник із заданими координатами діагонально протилежних вершин (лівої верхньої та правої нижньої);

circle(x, y, R) – рисує коло з радіусом R у точці з координатами (x, y);

arc(x, y, <початковий кут>,<кінцевий кут>, <радіус>) – рисує дугу;

pieslice(x, y, <початковий кут>, <кінцевий кут>, <радіус>) – рисує зафарбований сектор;

ellipse(x, y, <початковий кут>, <кінцевий кут>, <горизонт. радіус>, <вертикальн. радіус>) – рисує еліпс чи дугу еліпса;

setfillstyle(<заповнення>,<колір>) – задає спосіб заповнення замкнутої області залежно від значення параметра заповнення:

0 – заповнення кольором фону,

1 – суцільне заповнення,

2 – заповнення товстими горизонтальними лініями,

3 – заповнення нахиленими лініями, ... ,

10 – заповнення точками,

11 – щільне заповнення точками;

floodfill(x, y, <колір межі>) – заповнює замкнену область, що містить точку (x, y);

outtext(<текст>) – виводить заданий текст із поточної позиції;

outtextxy(x, y, <текст>) – виводить текст у заданому місці;

settextstyle(<шрифт>, <напрям>, <колір>) – задає вигляд символів, напрямок виведення: – горизонтально чи 1 – вертикально, а також розміри символів 1, 2, 3.

Розглянемо приклади:

Приклад 1. Виведення точок

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
main ()
{  int gdriver = DETECT, gmode, errorcode;
// Задання графічного режиму
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode !=grOk)
{printf("Graphics error: %s\n", grapherrormsg(errorcode));
getch();
```

```
exit(1); }
//setbkcolor(14); //колір фону
double midx = getmaxx() / 2; //знаходимо координати середини екрану
double midy = getmaxy() / 2; //ділимо x та y пополам
putpixel(midx,midy,10);
for (int i=1 ;i<=100 ;i++)
  putpixel(midx/i,midy/i,11);
getch();
closegraph(); }
```

Приклад 2. Побудова та зафарбування фігур

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
main ()
{  int gdriver = DETECT, gmode, errorcode;
// Задання графічного режиму
initgraph(&gdriver, &gmode, "");
//setbkcolor(14); //колір фону
setcolor(12);
rectangle(120,100,400,300);
setcolor(14);
bar(120,100,400,300);
setcolor(13);
setfillstyle(1,13);
circle(400,300,100);
floodfill(400,300,13);
getch();
closegraph(); }
```

Питання для самоконтролю

1. Чи буде працювати компілятор з графікою без допоміжних налаштувань?
2. Як на графічному екрані утворюється зображення?
3. Як називаються точки на графічному екрані?
4. Від чого залежить кількість точок на екрані?
5. З якої частини екрану ведуть відлік точок?
6. Чим характеризується кожна точка?
7. Як називається найпростіша бібліотека (під ОС DOS) призначена для побудови графічних зображень?
8. З чого складаються програми, що виконують графічні побудови?
9. Опишіть як перейти працювати у графічний режим.
10. Для чого призначена функція initgraph(&gdriver, &gmode, "") та що задає?
11. Опишіть кожен аргумент та його можливе значення з функції initgraph(&gdriver, &gmode, "")?
12. Для чого використовується функція graphresult()?
13. Для чого використовується функція getch()?
14. Для чого використовується функція closegraph()?
15. Назвіть функції для роботи з кольором.
16. Опишіть алгоритм (інструкцію) налаштування графічного режиму.
17. Назвіть основні функції для побудови точки.
18. Назвіть основні функції для побудови лінії.
19. Назвіть основні функції для побудови різних фігур.
20. Як задати заповнення та стиль для фігур?
21. Які функції у графічному режимі призначені для роботи з текстом?
22. Які функції дозволяють отримати координати центру екрану?

23. Наведіть приклад побудови еліпса та заповнення його певним кольором та певним стилем.
24. Для чого призначена функція floodfill()?

Тема 5. Приклади побудови графічних зображень, рух простої фігури

Продовжуємо розглядати графічну бібліотеку. Не менш цікавою є побудова руху графічних об’єктів. Повну інформацію (довідники) для роботи у графічному режимі на мові програмування С++ можна подивитися за наступними посиланнями [9], [10].

А також допоміжні та корисні матеріали можна брати тут [11].

Приклади побудови будинку, дерев та ін. можна подивитися за посиланням [12].

Наступні приклади демонструють виведення тексту та ліній різними стилями.

Для тексту:

"DEFAULT_FONT",

"TRIPLEX_FONT",

"SMALL_FONT",

"SANS SERIF_FONT",

"GOTHIC_FONT"

Приклад 1. Виведення тексту різного стилю

```
setcolor (12);
settextstyle(DEFAULT_FONT, HORIZ_DIR, 4);
outtext("Графічний режим!");
setcolor (10);
outtextxy(midx, midy, "fhgfgfg");
```

Для ліній:

"SOLID_LINE",

"DOTTED_LINE",

"CENTER_LINE",

"DASHED_LINE",

"USERBIT_LINE"

Приклад 2. Побудова ліній певного стилю

```
#include<graphics.h>
#include<conio.h>
#include<dos.h>
main()
{ int gd = DETECT, gm, x, y, color, angle = 0;
  struct arccoordstype a, b;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
  int midx, midy;
  midx = getmaxx() / 2;
  midy = getmaxy() / 2;
  setlinestyle(DOTTED_LINE, 1, 1);
  line(0, 0, midx-10, midy);
  line(midx-10, midy, getmaxx(), 0 );
  setlinestyle(DASHED_LINE, 1, 1);
  line( midx-10, midy, 0, getmaxy() );
  line( midx-10, midy, getmaxx(), getmaxy() );
  setlinestyle(USERBIT_LINE, 1, 1);
  line(0, midy, getmaxx(), midy );
  setlinestyle(CENTER_LINE, 1, 1);
  line( midx-10, 0, midx-10, getmaxy() );
  getch();
  closegraph();}
```

Приклад 3. Побудова кораблика

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
int gdriver = DETECT, gmode;
int i, maxx, maxy;
//Масив координат палуби
int poly[18]={100,100,
100,90,
110,90,
110,60,
130,60,
130,90,
140,90,
140,100,
100,100 };
//Масив координат корпуса
int poly1[8]={70,100,
170,100,
140,130,
100,130 };
initgraph(&gdriver, &gmode, "");
maxx = getmaxx();
maxy = getmaxy();
//Малюємо палубу
setfillstyle(1, 10);
```

```
fillpoly(9, poly);

// Малюємо корпус
setfillstyle(1, 3);
fillpoly(4,poly1);

//Малюємо дим
setfillstyle(1, 1);
fillellipse(185,30,40,15);
fillellipse(140,30,10,10);
fillellipse(130,40,10,10);
fillellipse(120,50,10,10);

//Ілюмінатор
setfillstyle(1,0);
fillellipse(119,80,8,8);

outtextxy(170,25,"ТУ-ТУ!!!");
getch();

closegraph();
return 0; }
```

Приклад 4. Різне заповнення замкнутої області (з поп. лекції)

```
#include <graphics.h>
#include <conio.h>
main ()
{ int gdriver=DETECT;
int gmode;
initgraph (&gdriver,&gmode,"C:\\TC\\bgi");
```

```
setcolor(14);
setfillstyle(2,14);
circle(252,250,100);
floodfill(250,250,14);
setcolor(10);
setfillstyle(1,10);
circle(300,250,100);
floodfill(300,250,10);
getch();
closegraph(); }
```

Приклад 5. Шахматна дошка (початок, решту домалювати самостійно)

```
#include <graphics.h>
#include <conio.h>
main ()
{ int gdriver=DETECT;
int gmode;
initgraph (&gdriver,&gmode,"C:\\TC\\bgi");
setcolor(13);
rectangle(50,70,100,120);
setcolor(13);
bar(50,70,100,120);
---------------------------------
setcolor(13);
rectangle(100,70,150,20);
setcolor(13);
bar(100,70,150,20);
-----------------------------
setcolor(1);
setfillstyle(1,1);
```

```
circle(75,95,20);
floodfill(75,95,1);
----------------------------
getch();
closegraph(); }
```

Приклад 6. Імітація об’ємного зображення (ідея)

Реалізувати:

- функцію, яка малює об’ємну фігуру;
- функцію або у головній програми переміщення фігури;
- побудову іншої об’ємної фігури використовуючи першу.

Приклад 7. Рух кола

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
void okr(int x, int y, int r, int color)
{setcolor(color);
circle(x,y,r); }
main ()
{ int gdriver=DETECT;
int gmode;
int x,y,r=50,i;
int maxx; //koordinata krainei pravoi tochki
initgraph (&gdriver,&gmode,"C:\\TC\\bgi");
setcolor (4);
outtextxy(250,2,"Біжуче коло!!!");
maxx=getmaxx();
for (y=60;y<=400; y+=100)
```

```
 { x=0;
while(x<maxx)
  {
  okr(x,y,r,RED);      //малюємо коло
  delay(20);
  okr(x,y,r,BLACK);    // видаляємо коло
  x+=10;
  }
 }
getch();
closegraph(); }
```

Питання для самоконтролю

1. Який загальний алгоритм створення графічних зображень?
2. Як розмістити текст у графічному режимі горизонтально, вертикально?
3. Які стилі ви побачили для виведення тексту?
4. Які стилі ви побачили для виведення ліній?
5. Розглянувши приклад 3, що ви можете сказати про використання у графічних побудовах масиву?
6. Дослідіть як працює функція fillpoly()?
7. Дослідіть як працює функція fillellipse ()?
8. Продемонструйте приклад використання функції floodfill() відмінний від прикладу 4.
9. В чому полягає ідея графічної побудови шахматної дошки?
10. Яка ідея побудови об’ємного зображення?
11. Опишіть ідею створення рухомого зображення?
12. Наведіть приклад створення руху елементарної фігури, крім фігури з прикладу 7.

Тема 6. Програмування побудови графіка функції, рух складного зображення

Приклад побудови та переміщення (копіювання) об’єкта можна подивитися за посиланням [12].

Розглянемо приклад побудови графіка функції.

Приклад 1. Нарисувати графік функції $y=2\sin 2x+1$ на проміжку [0; 2π] табулюючи функцію з кроком h = 0,1. У результаті експериментів, змінюючи значення амплітуди в пікселях (за допомогою множника M), зобразити графік на екрані як найкраще.

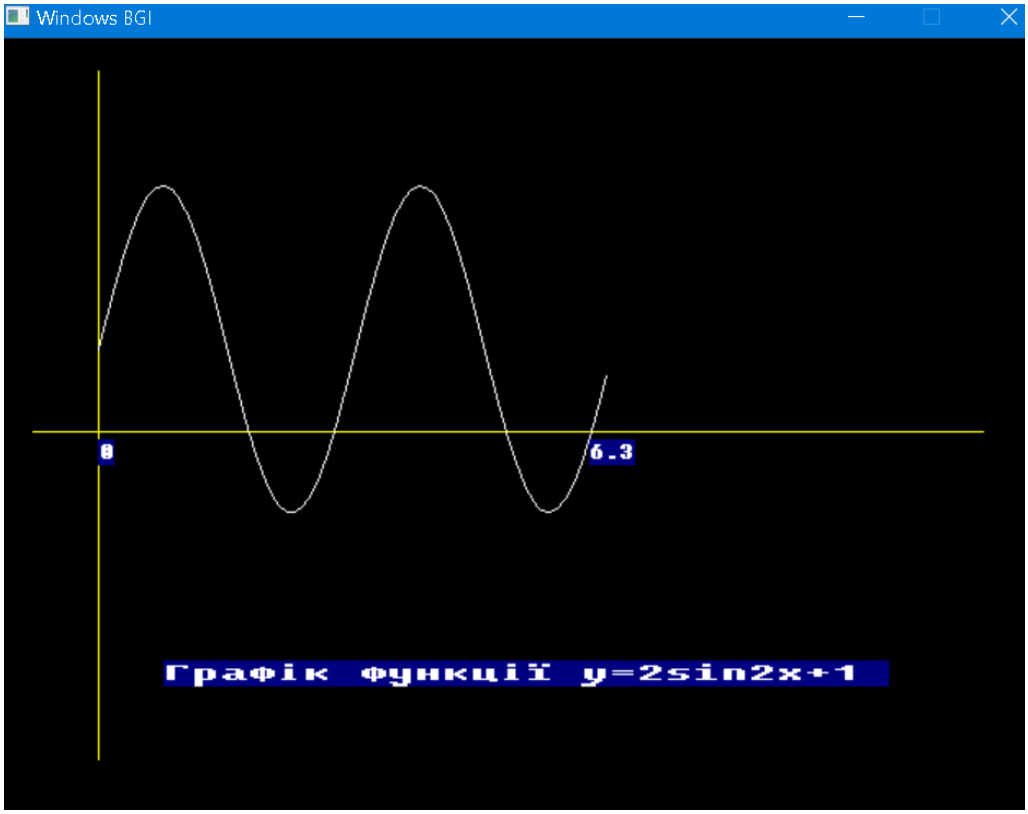


Рис. 6.1. – Приклад програмування графіка функції

```
#include <stdlib.h>
```

```
#include <iostream>
#include <math.h >
#include <conio.h>
#include <graphics.h>
float f(float x);
int main()
{
  const float a=0, b=2* 3.14, h = 0.1;
  const int h1 = 5, x0 = 60, y0 = 240, M = 50;
  int gdriver = DETECT, gmode, errorcode;
  int x1, y1;
  float x, y;
 initgraph(&gdriver, &gmode, "");
 setbkcolor(1);
 setcolor(14);
 line (20, y0, 600, y0);
 line(x0, 440, x0, 20);
 x=a;
 x1=x0+M*x;
 y = f(x);
 y1=y0-M*y;
 moveto(x1,y1);
 setcolor(15);
 do
 { y= f(x);
  y1=y0-M*y;
  lineto(x1, y1);
  x1+=h1;
  x=x+h;
  }
```

```
  while (x <= b);
  settextstyle( 0, 0, 1);
  outtextxy(60, 245, "0");
  outtextxy(360, 245, "6.3");
  settextstyle(0, 0, 2);
  outtextxy(100, 380, "Графік функції y=2sin2x+1 ");
  getch();
  closegraph();
  return 0;
}
float f(float x)
{
float func;
 func = 2 * sin(2*x) + 1;
return func;
}
```

Приклад 2. Написати програму для виведення на екран графіка функцій: y=4*cos(x). (Легший спосіб, проте погане масштабування)

```
#include <stdlib.h>
#include <iostream>
#include <math.h >
#include <conio.h>
#include <graphics.h>
int main()
{  int gdriver = DETECT, gmode, errorcode;
  initgraph(&gdriver, &gmode, "");
  int x,y,x1,y1,x0,y0;
  setcolor(12);
  x0=getmaxx()/2;
```

```
 y0=getmaxy()/2;
 line (0, y0, getmaxx(), y0);
 line(x0, 0, x0, getmaxy());
 setcolor(6);
 x1=x0;
 y1=y0+4*cos(x1);
 putpixel(x1,y1,14);
 for (int i=1; i<300;i++)
 {
    x=x0+i;
    y=y0+4*cos(x);
    putpixel(x,y,14);
 }
 getch();
 closegraph();
 return 0;
}
```

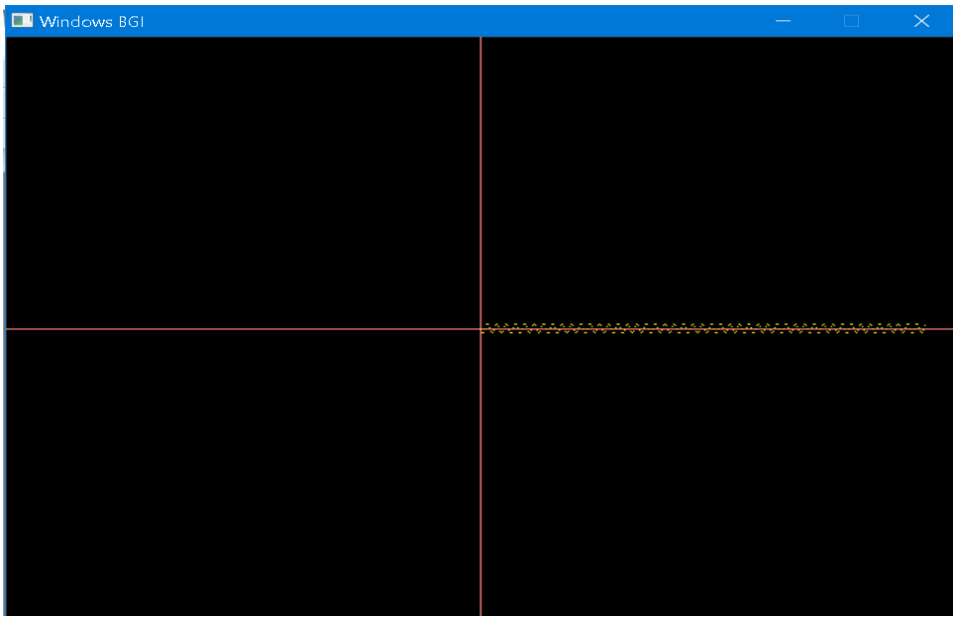


Рис. 6.2. – Приклад виведення на екран графіка функції (невдале масштабування)

Продовжуємо розглядати побудову рухомих зображень.

Приклад 3. Програмуємо рух велосипеда.

```
#include <stdlib.h>
#include <iostream>
#include <math.h >
#include <conio.h>
#include <graphics.h>
int main()
{  int gdriver = DETECT, gmode, errorcode;
  initgraph(&gdriver, &gmode, "");
  for (int i=1;1<600;i++)
  {
  setcolor(14);
  line(50+i,405,100+i,405);
  line(75+i,375,125+i,375);
  line(50+i,405,75+i,375);
  line(100+i,405,125+i,375);
  line(150+i,405,100+i,345);
  line(75+i,375,75+i,370);
  line(70+i,370,80+i,370);
  line(80+i,345,100+i,345);
  circle(150+i,405,30);
  circle(50+i,405,30);
  setcolor(12);
  line(0,436,getmaxx(),436);
  delay(10);
  cleardevice();
  }
  getch();
```

```
  closegraph();
  return 0;
}
```

Для кращого розуміння техніки побудови анімації можна подивитися у циклі уроків C/C++ Graphics Tutorial 9 | Car Animation 4 коротких відео, що на рис. 6.3 [13].

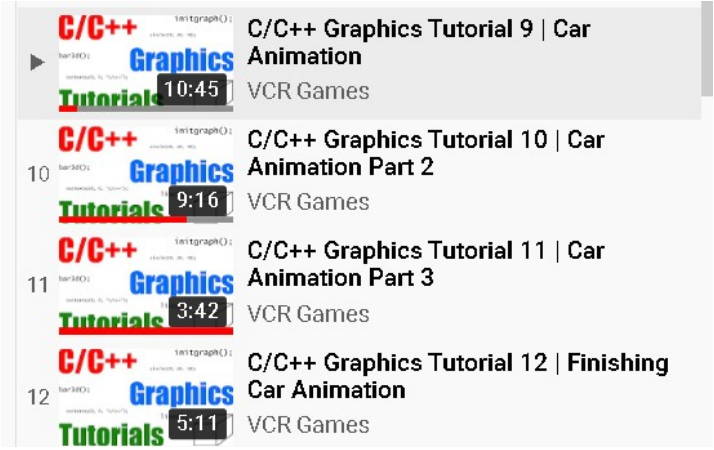


Рис. 6.3. – Фото для орієнтиру, які відео проглянути

Питання для самоконтролю

1. На які частини можна розбити програмування побудови графіка функції?
2. Як побудувати осі координат?
3. Які функції потрібно використати для побудови координатних осей?
4. Як підписати координатні осі?
5. Що таке масштабування та для чого його використовують?
6. Як запрограмувати масштабування графіка функції?
7. Коли використовуються цикли у графіці?
8. Чим відрізняється програмування побудови графіка функції у прикладах 1 та 2?
9. Чому ми розрізняємо просте графічне зображення та складне? Наведіть приклади простого та складного.

10. У яких випадках ми говоримо про рух простої фігури?
11. У яких випадках ми говоримо про рух складної фігури?
12. В чому полягає іде створення руху складного зображення?
13. Опишіть техніку побудови анімації з заданого джерела.
14. Прокоментуйте приклад 3 та його особливості.

Тема 7. Графічна бібліотека OpenGL

Що таке OpenGL?

Стандартна графічна бібліотека OpenGL – один з найпопулярніших прикладних програмних інтерфейсів (API – Application Programming Interface) для розробки застосувань реалістичної двовимірної та тривимірної графіки.

Стандарт OpenGL (Open Graphics Library – відкрита графічна бібліотека) був розроблений і затверджений у 1992 році провідними фірмами в сфері індустрії програмного забезпечення як ефективний апаратно незалежний інтерфейс, придатний для реалізації на різних платформах. Основою стандарту стала бібліотека IRIS GL, розроблена фірмою Silicon Graphics Inc.

На сьогоднішній день графічна система OpenGL підтримується більшістю виробників апаратних і програмних платформ. Ця система доступна тим, хто працює в середовищі Windows, користувачам комп’ютерів Apple. Вільно розповсюджувані коди системи Mesa (пакет API на базі OpenGL) можна компілювати в більшості операційних систем, у тому числі в Linux.

Причиною широкого розповсюдження і розвитку OpenGL як графічного стандарту є такі його ключові характеристики:

- Стабільність. Доповнення і зміни в стандарті реалізуються таким чином, щоб зберегти сумісність з розробленим раніше програмним забезпеченням.
- Надійність і простота перенесення на інші апаратно-програмні платформи. Застосування, що використовують OpenGL, гарантують однаковий візуальний результат в різних операційних системах та на різному обладнанні. Крім того, ці застосування можуть виконуватися як на персональних комп’ютерах, так і на робочих станціях чи суперкомп’ютерах.
- Легкість використання. Стандарт OpenGL має продуману структуру та інтуїтивно зрозумілий інтерфейс, що дозволяє з меншими витратами створювати ефективні застосування, які містять менше рядків коду, ніж створені з використанням інших графічних бібліотек. Необхідні функції для забезпечення сумісності з різним обладнанням реалізовані на рівні бібліотеки і значно спрощують розробку застосувань.

Реалізація бібліотеки OpenGL в операційній системі Windows містить більше 400 різних команд для опису об’єктів і операцій в процесі створення інтерактивних графічних застосувань [1], що, природно, ускладнює її вивчення за експлуатаційною документацією. Використання OpenGL є необхідною передумовою для наступного поглиблення теоретичних знань і практичних навичок зі створення графічних застосувань.

Основні можливості

Основне призначення OpenGL – відображення двовимірних та тривимірних об’єктів у статичних та динамічних сценах. Об’єкти представляються у вигляді сукупності вершин (для геометричних фігур) або пікселів (для растрових зображень). OpenGL спочатку перетворює вихідні дані (примітиви та зображення) в піксельне подання, асоціюючи з кожним

сформованим пікселем необхідні для його відображення і подальшої роботи дані, а потім розташовує результат перетворення в буфері кадру.

Реалізація OpenGL для Windows містить більше 400 функцій (368 базових функцій основної бібліотеки opengl32.dll та 52 функції бібліотеки утиліт glu32.dll).

Базові функції забезпечують побудову зображень графічних примітивів (точки, лінії, багатокутники, растрові зображення), перетворення координат, обмеження області видимості, управління кольором, освітленням, текстурою, туманом.

Функції бібліотеки утиліт є розширенням базового набору функцій і призначені для формування зображень сфер, дисків, конічних циліндрів, управління текстурою і перетвореннями координат, тріангуляції багатокутників, побудови кривих та поверхонь на нерегулярній сітці контрольних точок з використанням форм Без’є та раціональних B-сплайнів.

Всі базові функції можна розділити на п’ять категорій:

- Функції опису примітивів визначають об’єкти нижнього рівня ієрархії (примітиви), які здатна відображати графічна система. У OpenGL примітивами є точки, лінії, багатокутники і т.д.
- Функції опису джерел світла служать для опису положення і параметрів джерел світла, розташованих у тривимірній сцені.
- Функції задання атрибутів. За допомогою задання атрибутів програміст визначає, як будуть виглядати на екрані відображувані об’єкти. Іншими словами, якщо за допомогою примітивів визначається, що з’явиться на екрані, то атрибути визначають спосіб виводу на екран. Як атрибути OpenGL використовує колір, характеристики матеріалу, текстури, параметри освітлення.
- Функції візуалізації дозволяють задати положення спостерігача у віртуальному просторі, параметри об’єктива камери. Знаючи ці параметри, система зможе не тільки правильно побудувати зображення, але і відсікти об’єкти, які не потрапили в поле зору.

- Набір функцій геометричних перетворень дозволяє програмісту виконувати різні перетворення об’єктів – поворот, зсув, масштабування.

Інтерфейс OpenGL

OpenGL складається з набору бібліотек. Усі базові функції зберігаються в основній бібліотеці, для позначення якої надалі будемо використовувати абревіатуру GL. Крім основної, OpenGL містить кілька додаткових бібліотек.

Перша з них – бібліотека утиліт GL (GLU – GL Utility). Усі функції цієї бібліотеки визначені через базові функції GL. До складу GLU увійшла реалізація більш складних функцій, таких як набір популярних геометричних примітивів (куб, куля, циліндр, диск), функції побудови сплайнів, реалізація додаткових операцій над матрицями і т.п.

OpenGL не містить у собі ніяких спеціальних команд для роботи з вікнами чи отримання інформації від користувача. Тому були створені спеціальні бібліотеки для забезпечення функцій, що часто використовуються при взаємодії з користувачем і для відображення інформації за допомогою віконної підсистеми. Найбільш популярною є бібліотека GLUT (GL Utility Toolkit). Формально GLUT не входить у OpenGL, але фактично включається майже в усі його дистрибутиви і має реалізації для різних платформ. GLUT надає лише мінімально необхідний набір функцій для створення OpenGL-застосування. Функціонально аналогічна бібліотека GLX менш популярна.

Крім того, функції, специфічні для конкретної віконної підсистеми, звичайно входять у її прикладний програмний інтерфейс. Так, функції, що підтримують виконання OpenGL, є в складі Win32 API і X Window.

На рис. 7.1 схематично представлена організація системи бібліотек у версії, що працює під управлінням системи Windows. Аналогічна організація використовується й в інших версіях OpenGL

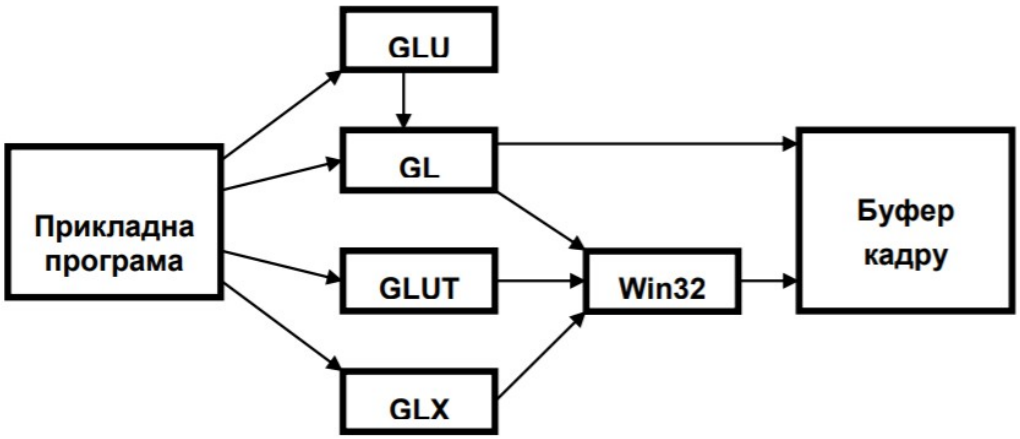


Рис. 7.1. - Організація бібліотеки OpenGL

Архітектура OpenGL

Функції OpenGL реалізовані з використанням моделі клієнт-сервер. Застосування (прикладна програма) виступає в ролі клієнта – воно виробляє команди, а сервер OpenGL інтерпретує і виконує їх. Сам сервер може знаходитися як на тому ж комп’ютері, що й клієнт (наприклад, у вигляді бібліотеки динамічного завантаження – DLL), так і на іншому (для цього може бути використаний спеціальний протокол передачі даних між машинами).

OpenGL обробляє і формує в буфері кадру зображення графічних примітивів з урахуванням деякого числа обраних режимів. Окремий примітив – це точка, відрізок, багатокутник і т.д. Кожен режим може бути змінений незалежно від інших. Визначення примітивів, вибір режимів та інші операції описуються за допомогою команд у формі викликів функцій прикладної бібліотеки.

Примітиви визначаються набором з однієї чи декількох вершин (vertex). Вершина визначає точку, кінець відрізка чи кут багатокутника. З кожною вершиною асоціюються деякі дані (координати, колір, нормаль, текстурні координати і т.д.), які називаються атрибутами. У переважній більшості випадків кожна вершина обробляється незалежно від інших.

Архітектура OpenGL реалізує схему конвеєра, що складається з кількох послідовних етапів обробки графічних даних (рис. 7.2).

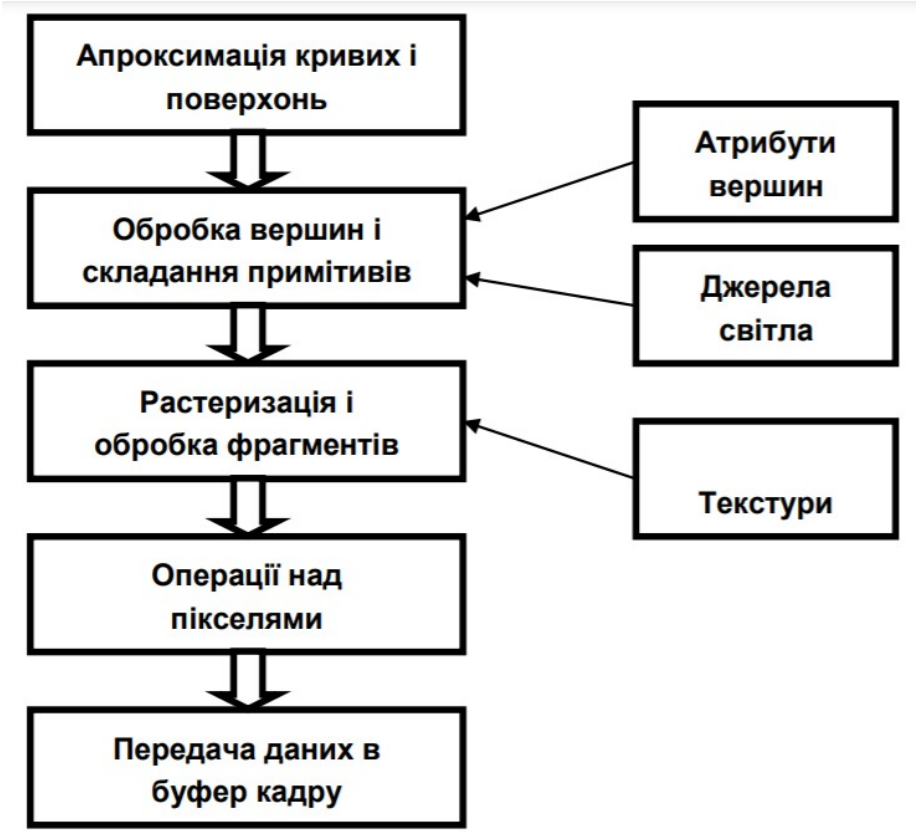


Рис. 7.2. - Схема функціонування конвеєра OpenGL

Команди OpenGL завжди обробляються в порядку їх надходження, хоча можуть відбуватися затримки перед тим, як проявиться ефект від їхнього виконання. У більшості випадків OpenGL реалізує безпосередній інтерфейс, тобто визначення об’єкта викликає його візуалізацію в буфері кадру.

З точки зору розробників, OpenGL – це набір команд, які керують використанням графічної апаратури. Якщо апаратура складається тільки з адресованого буфера кадру, то тоді функції OpenGL повинні бути реалізовані повністю за рахунок ресурсів центрального процесора. Як правило графічна апаратура забезпечує різноманітні рівні прискорення: від апаратної реалізації виводу ліній і багатокутників до витончених графічних процесорів з підтримкою різних операцій над геометричними даними.

OpenGL є прошарком між апаратним рівнем та рівнем користувача, що дозволяє надавати єдиний інтерфейс на різних платформах, використовуючи можливості апаратної підтримки.

Крім того, OpenGL можна розглядати як скінченний автомат, стан якого визначається множиною значень спеціальних змінних і значеннями поточної нормалі, кольору, координат текстури й інших атрибутів і ознак. Уся ця інформація буде використана при надходженні в графічну систему координат вершини для побудови фігури, до якої вона належить. Зміна станів відбувається за допомогою команд, що оформлюються як виклики функцій.

Синтаксис команд

Бібліотеки opengl32.dll та glu32.dll написані на мові C++. Тому для використання цих бібліотек в проектах C++ достатньо підключити їх заголовні файли (gl.h та glu.h відповідно):

#include <GL/gl.h>

#include <GL/glu.h>

Версії цих бібліотек, як правило, включаються у дистрибутиви систем програмування.

На відміну від стандартних бібліотек, пакет GLUT потрібно інсталювати та підключати окремо. Детальніше розглянемо нижче.

Усі команди (процедури і функції) бібліотеки OpenGL починаються з префікса gl, усі константи – з префікса GL_. Команди і константи бібліотек GLU і GLUT аналогічно мають префікси glu (GLU_) і glut (GLUT_).

Крім того, у імена команд входять суфікси, що несуть інформацію про число і тип переданих параметрів. У OpenGL повне ім’я команди має такий вид:

type **glCommand_name** [1 2 3 4] [b i f d ub us ui][v]

(type1 arg1,…, typeN argN)

Ім’я складається з декількох частин (розписано нижче):

Gl	ім’я бібліотеки, у якій описана ця функція: для базових функцій OpenGL це *gl*, для функцій із бібліотек GL, GLU, GLUT, GLAUX – *glu*, *glut*, *aux* відповідно
Command_name	ім’я команди (процедури чи функції)
[1 2 3 4]	число аргументів команди
[b s I f d ub us ui]	тип аргументу: символ *b* – *GLbyte* (аналог *char* у C\C++), символ I – *GLint* (аналог int), символ *f* – *GLfloat* (аналог *float*) і т.д. Повний список типів і їх описів можна подивитися у файлі *opengl.pas*
[v]	наявність цього символу показує, що у якості параметрів функції використовується покажчик на масив значень

Символи в квадратних дужках у деяких назвах не використовуються. Наприклад, описана в бібліотеці OpenGL команда glVertex2i, використовує як параметри два цілих числа, а команда glColor3fv використовує як параметр покажчик на масив із 3-х дійсних чисел.

Приклад найпростішої програми

Перед початком написання найпростішої програми з використанням графічної бібліотеки OpenGL потрібно зробити налаштування. По аналогії до налаштувань бібліотеки graphic.h помістити певні файли у відповідні папки. Детально робоче (перевірене) налаштування описане за посиланням [14] . Якщо після виконання дій, що в інструкції, і далі не працює, то можна спробувати для компілятора прописати праву частину, щоб виглядало так як на рис. 7.3.

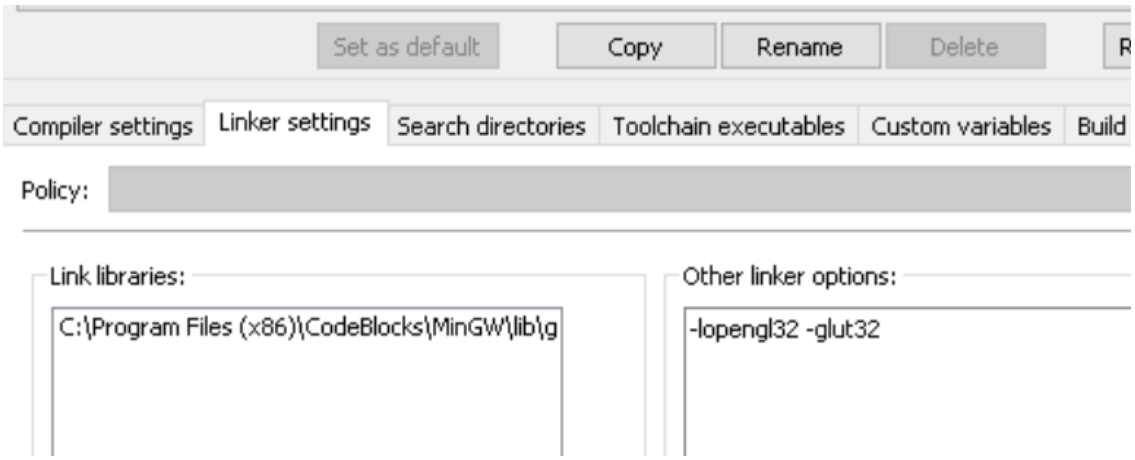


Рис. 7.3. – Прописуємо бібліотеки для компілятора при підключенні OpenGL

Для того, аби перевірити чи ми зможемо програмувати графіку потрібно зайти у File>New>Project… вибрати Glut project і далі за інструкцією, або File>New>Project… вибрати Console application і у головне вікно вставити наступний код:

```
#include <windows.h>
#include <GL/glut.h> // підключаємо бібліотеки
void Drow()
{// щось малюємо
}
int main(int argc, char *argv[])
{  glutInit(&argc, argv);  // ініціалізація графіки стандартно
   glutInitWindowSize(640,480); // задаємо розміри вікна
      glutInitWindowPosition(20,10); // задаємо позицію, з якої буде виводитися вікно
            glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH); // задаємо кольори дисплею
   glutCreateWindow("GLUT Shapes"); // задаємо назву графічного вікна
   glutDisplayFunc(Drow); // щось малюємо
   glutMainLoop(); } // функція відповідає за обробку подій
```

В результаті виконання з’явиться 2 вікна: одне – назва.exe, інше – створено програмно, воно буде мати біле дно і назву таку як задали у функції glutCreateWindow("GLUT Shapes"). Особливістю програмування графіки є те, що усі функції бібліотек мають префікс назви бібліотеки, в даному випадку glut.

Наведемо приклад програмування побудови трикутника з різними кольорами у вершинах:

```
static void Drow(void)
{   glColor3d(1,0,0);
   glBegin (GL_TRIANGLES);
   glColor3f (12.0, 0.0, 0.0); //червоний
   glVertex3f (0.0, 0.0, 0.0);
   glColor3f(0,1,0); //зелений
   glVertex3f (1.0, 0.0, 0.0);
   glColor3f(0.0,0.0,1.0); //блакитний
   glVertex3f (1.0, 1.0, 0.0);
   glEnd ();

 glColor3f(1.0,1.0,0.0);
 glBegin(GL_QUADS);
 glVertex2f(0.5, 0.5);
 glVertex2f(-0.5,0.5);
 glVertex2f(-0.5, -0.5);
 glVertex2f(0.5, -0.5);
 glEnd ();
glFlush (); }
```

Команди мають префікс gl, так програмувати можна, оскільки, на початку програми підключили зразу 2 бібліотеки #include <GL/glut.h>. Звертаю увагу на те, що і тут застосовано принцип “сендвіча”: є glBegin(), сама побудова фігури та glEnd ().

Розглянемо детальніше функцію int main даного прикладу.

Функція glutInit(&argc, argv) виконує початкову ініціалізацію самої бібліотеки GLUT.

Команда glutInitDisplayMode(GLUT_RGB) ініціалізує буфер кадра і налаштовує повноекранний (непалітровий) режим RGB.

glutInitWindowSize(Widht, Height) використовується для задання початкових розмірів вікна.

glutCreateWindow("GLUT Shapes") задає заголовок вікна та візуалізує саме вікно на екрані.

Можуть бути ще команди:

glutDisplayFunc(Display);

glutReshapeFunc(Reshape);

glutKeyboardFunc(Keyboard);

реєструють функції Display(), Reshape(), та Keyboard() як функції, що будуть викликані, відповідно, при перемалюванні вікна, зміни розмірів вікна, натисненні клавіш на клавіатурі.

Контроль усіх подій і викликів потрібних функцій відбувається в середині безкінечного циклу у функції glutMainLoop().

Питання для самоконтролю

1. Що таке OpenGL?
2. Охарактеризуйте стандарт OpenGL.
3. Які ОС підтримують використання OpenGL?
4. Опишіть характеристику стабільність.
5. Опишіть характеристику надійність і простота.
6. Опишіть характеристику легкість використання.
7. Яке основне призначення OpenGL?
8. У вигляді чого в OpenGL представляються об’єкти?

9. Побудову який графічних примітивів забезпеують базові функції?
10. Для формування яких зображень призначені функції з розширення базового набору?
11. Охарактеризуйте функції опису примітивів.
12. Охарактеризуйте функції задання джерел світла.
13. Охарактеризуйте функції задання атрибутів.
14. Охарактеризуйте функції візуалізації.
15. Охарактеризуйте функції геометричного перетворення об’єктів.
16. Які додаткові бібліотеки містить OpenGL?
17. Для чого призначені бібілотеки утиліт GL, GLU?
18. Для чого призначені бібілотеки утиліт GLUT, GLX?
19. Опишіть архутектуру OpenGL.
20. Прокоментуйте схему функціонування конвеєра OpenGL.
21. Які особливості OpenGL пов’язані з апаратним рівнем та рівнем користувача?
22. OpenGL можна розглядати як скінченний автомат. Чому? Обгрунтуйте.
23. Як почати використовувати бібілотеки OpenGL у C++?
24. Яка особливість команд бібілотеки OpenGL?
25. Що таке префікс та суфікс у командах OpenGL?
26. Який вигляд має повне ім’я команди в OpenGL?
27. Опишіть алгоритм налаштувань графічної бібліотеки OpenGL.
28. Прокоментуйте початковий код з ініціалізації бібліотеки.
29. Розгляньте приклад побудови трикутника, як задають вершини та кольори?
30. Яка технологія програмування побудови фігури використовується і у цій графічній бібілотеці?
31. Для чого служить функція glutInit()?
32. Для чого служить функція glutInitDisplayMode()?

33. Для чого служить функція glutInitWindowSize()?
34. Для чого служить функція glutCreateWindow()?
35. Для чого служать функції: glutDisplayFunc(Display); glutReshapeFunc(Reshape); glutKeyboardFunc(Keyboard)?
36. Яка функція контролює усі події та виклики інших функцій?

Тема 8. Побудова, зафарбовування та обертання примітивів з використанням OpenGL

Процес оновлення зображення

Як правило, задачею програми, що використовує OpenGL, є обробка тривимірної сцени і її відображення в буфері кадру. Сцена складається з набору тривимірних об’єктів, джерел світла і віртуальної камери, яка визначає поточне положення спостерігача.

Звичайно застосування OpenGL у нескінченному циклі викликає функцію оновлення зображення у вікні. У цій функції і зосереджені виклики основних команд OpenGL. Якщо використовується бібліотека GLUT, то це буде функція зі зворотним викликом, зареєстрована за допомогою виклику glutDisplayFunc. GLUT викликає цю функцію, коли операційна система інформує про те, що вміст вікна необхідно перемалювати (наприклад, якщо вікно було перекрито іншим). Створюване зображення може бути як статичним, так і анімованим, тобто залежати від деяких параметрів, що змінюються з часом. У цьому випадку краще викликати функцію оновлення самостійно.

Як правило типова функція оновлення зображення забезпечує:

- очищення буферів OpenGL:
- встановлення положення спостерігача;
- перетворення і малювання геометричних об’єктів.

Очищення буферів забезпечується командою:

glClear (mask: bitfield)

Ця команда очищує задані буфери з поточним значенням. Параметр mask є комбінацією наступних значень:

GL_COLOR_BUFFER_BIT	буфер кольору
GL_DEPTH_BUFFER_BIT	буфер глибини
GL_ACCUM_BUFFER_BIT	буфер-накопичувач
GL_STENCIL_BUFFER_BIT	буфер трафарету

Типова програма викликає команду glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) для очищення буферів кольору і глибини.

Вершини та їх положення в просторі

Вершина є атомарним графічним примітивом OpenGL і визначає точку, кінець відрізка, кут багатокутника і т.д. Всі інші примітиви формуються за допомогою завдання вершин, що входять у даний примітив. Наприклад, відрізок визначається двома вершинами, що є кінцями відрізка.

З кожною вершиною асоціюються її *атрибути*, основними з яких є положення вершини в просторі, її колір і вектор нормалі (розглянемо нижче).

Положення вершини визначається завданням її координат у дво-, три-, чи чотиривимірному просторі (однорідні координати). Це реалізується за допомогою декількох варіантів команди glVertex:

glVertex [2 3 4] [s i f d] (coords: GLtype)

glVertex [2 3 4] [s i f d]v (coords: ^GLtype)

Кожна команда задає чотири координати вершини: x, y, z, w. Команда glVertex2* одержує значення x і y. Координата z у такому випадку встановлюється за замовчуванням рівною 0, координата w — рівною 1. Команда glVertex3* одержує координати x, y, z і заносить у координату w значення 1. Команда glVertex4* дозволяє задати всі чотири координати.

Для асоціації з вершинами кольорів, нормалей і текстурних координат використовуються поточні значення відповідних даних, що відповідає

організації OpenGL як скінченного автомата. Ці значення можуть бути змінені в будь-який момент за допомогою виклику відповідних команд.

Атрибути примітивів

З кожною вершиною будь-якого примітиву зв'язані наступні атрибути:

Атрибут	**Команда**
Поточний колір – колір вершини (остаточний колір вираховується з урахуванням освітлення)	glColor[3 4][b s i f d ub us ui][v](args).
Поточні координати текстури – координати текстури, що відповідають цій вершині	GlTexCoord
Поточна нормаль – вектор нормалі, що відповідає даній вершині	GlNormal
Атрибути точок:	
Розмір точки	glPointSize (GLfloat size)
Параметрами згладжування антиаліасінг	glEnable (GL_POINT_SMOOTH) glDisable (GL_POINT_SMOOTH)
Атрибути ліній:	
Ширина лінії	glLineWidth(GLfloat width)
Параметрами згладжування антиаліасінг	glEnable (GL_LINE_SMOOTH) glDisable (GL_LINE_SMOOTH)
Штрихування лінії	glLineStipple(GLint factor, GLushort pattern) glEnable (GL_LINE_STIPPLE) glDisable (GL_LINE_STIPPLE)
Атрибути трикутників, чотирикутників. багатокутників:	
Параметрами згладжування	glEnable (GL_POLYGON_SMOOTH)

антиаліасінг	glDisable (GL_POLYGON_SMOOTH)
Штрихування лінії	glPolygonStipple (GLubyte *pattern) glEnable (GL_POLYGON_STIPPLE) glDisable (GL_POLYGON_STIPPLE)
Режим малювання	glPolygonMode (GLenum face, Glenum mode)
Правило розрізнення граней	glFrontFace (GLEnum mode) glEnable (GL_CULL_FACE) glDisable (GL_CULL_FACE)

Операторні дужки glBegin/glEnd

Ми розглянули завдання атрибутів однієї вершини. Однак, щоб задати атрибути графічного примітива, лише координат вершин недостатньо. Ці вершини треба об’єднати в одне ціле, визначивши необхідні властивості. Для цього в OpenGL використовуються так звані операторні дужки, що є викликами спеціальних команд OpenGL. Визначення примітива чи послідовності примітивів відбувається між викликами команд

glBegin (mode: GLenum)

glEnd

Параметр mode визначає тип примітива, який задається всередині і може приймати наступні значення, що у таблиці нижче і будують наступні примітиви, що на рис. 8.1:

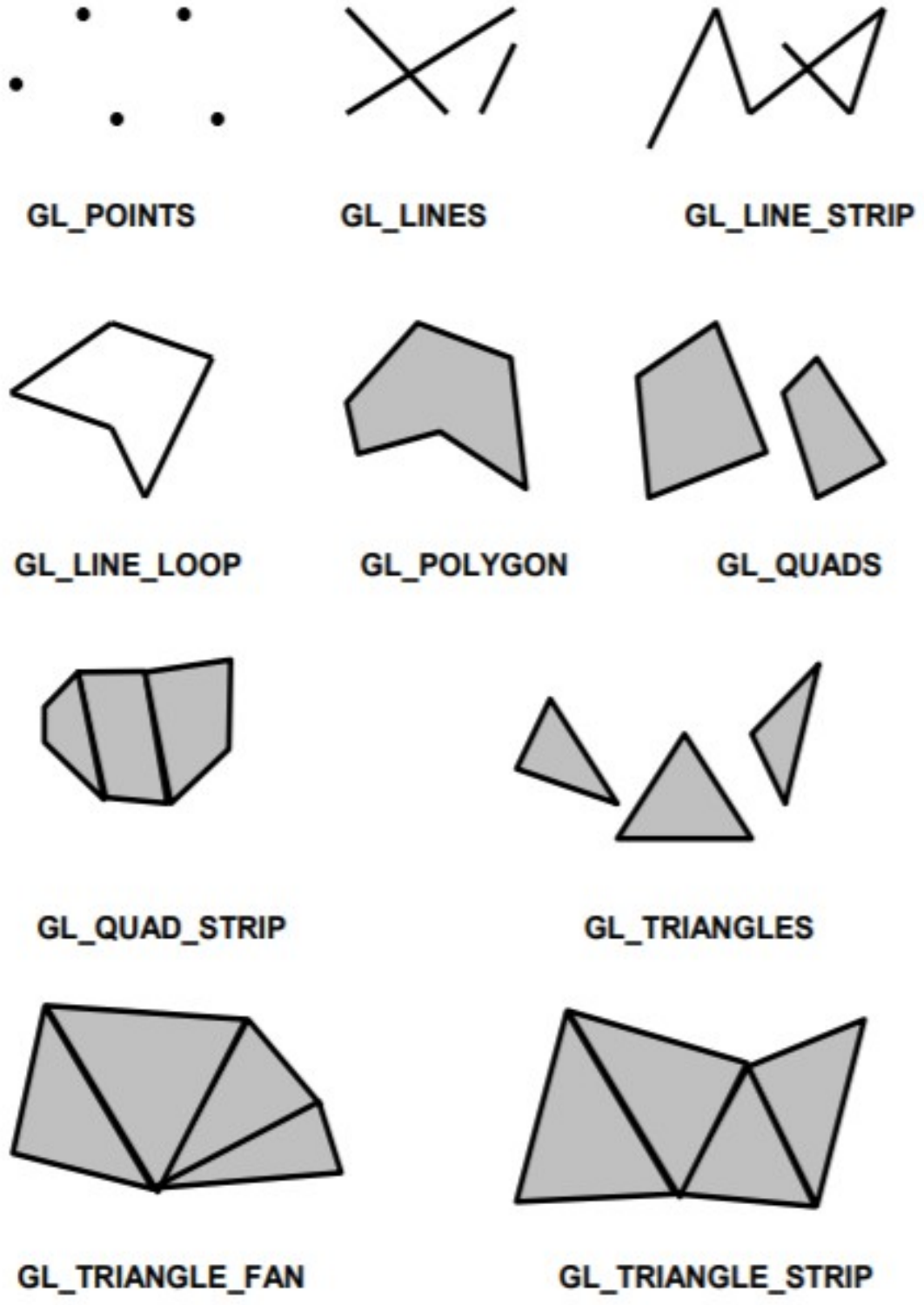
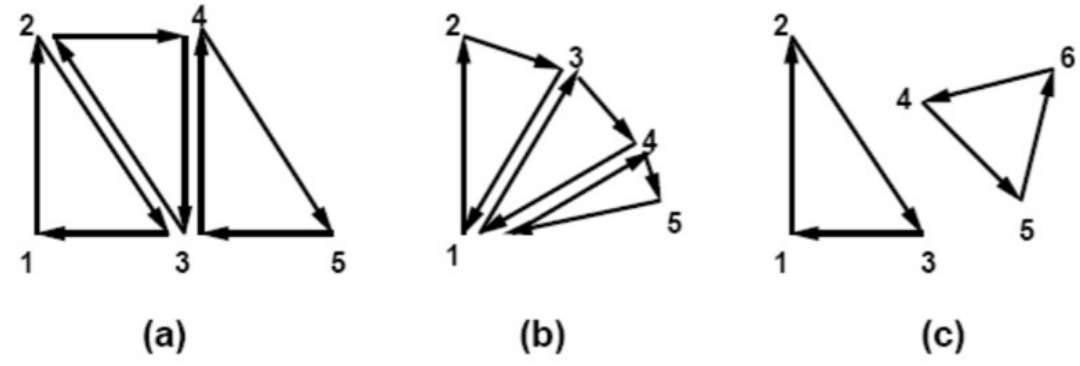


Рис. 8.1. – Вигляд примітивів OpenGL

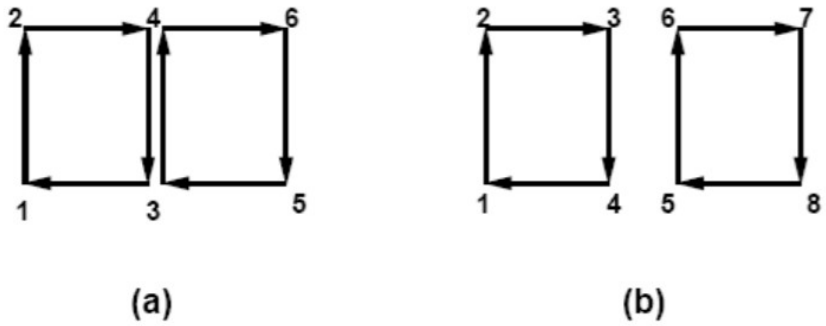
GL_POINTS	кожна вершина задає координати деякої точки
GL_LINES	кожна окрема пара вершин визначає відрізок; якщо задане непарне число вершин, то остання вершина ігнорується
GL_LINE_STRIP	кожна наступна вершина задає відрізок разом з попередньою
GL_LINE_LOOP	відмінність від попереднього примітива тільки в тім, що останній відрізок визначається останньою і першою вершиною, утворюючи замкнуту ламану
GL_TRIANGLES	кожні окремі три вершини визначають трикутник; якщо задане не кратне трьом число вершин, то останні вершини ігноруються
GL_TRIANGLE_STRIP	кожна наступна вершина задає трикутник разом із двома попередніми
GL_TRIANGLE_FAN	трикутники задаються першою вершиною і кожною наступною парою вершин (пари не перетинаються)
GL_QUADS	кожна окрема четвірка вершин визначає чотирикутник; якщо задане не кратне чотирьом число вершин, то останні вершини ігноруються
GL_QUAD_STRIP	чотирикутник з номером n визначається вершинами з номерами 2n-1, 2n, 2n+2, 2n+1
GL_POLYGON	послідовно задаються вершини *опуклого* багатокутника

Рис. 8.2. – Опис примітивів



Режими побудови трикутників:
a – GL_TRIANGLE_STRIP; b – GL_TRIANGLE_FAN; c – GL_TRIANGLES.

Рис. 8.3. – Побудова трикутників



Режими побудови прямокутників:
a – GL_QUAD_STRIP; b – GL_QUADS.

Рис. 8.4. – Побудова прямокутників

Системи координат в OpenGL

В OpenGL використовуються як основні три системи координат: лівобічна, правобічна і віконна (рис. 8.5). Перші дві системи є тривимірними і відрізняються одна від одної напрямком осі Z: у правобічній вона спрямована на спостерігача, в лівобічній — у глибину екрана. Вісь X спрямована вправо щодо спостерігача, вісь Y— вгору.

Лівобічна система використовується для завдання значень параметрам команди gluPerspective, glOrtho, які будуть розглянуті далі. Правобічна система координат використовується у всіх інших випадках. Відображення тривимірної інформації відбувається в двовимірну віконну систему координат.

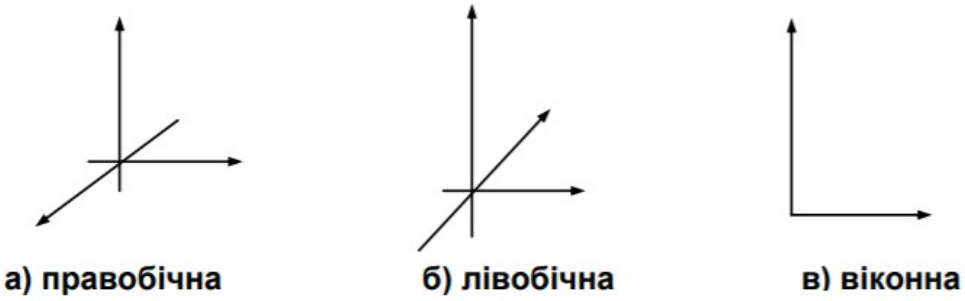


Рис. 8.5. - Системи координат в OpenGL

OpenGL дозволяє шляхом маніпуляцій з матрицями моделювати як праву, так і ліву систему координат. Але на даному етапі краще піти простим шляхом і запам’ятати: основною системою координат OpenGL є правобічна система.

Програмування побудови примітивів на екрані

В темі № 7, для того щоб перевірити чи вірно налаштована графіка, було продемонстровано код програмування примітивів: трикутника та квадрата, задано їх фарбування. Розглянемо детальніше код, дещо добавимо і опишемо.

Отож, трикутник можна задати трьома точками. В нашому випадку у 3Д просторі ми кажемо про вершину. Вершина має 3 координати: X, Y, Z. Ви можете уявити для себе ці координати так:

X – вправо

Y – вверх

Z – за спину

Але є ще один спосіб представлення цієї системи координат: правило правої руки:

X – великий палець правої руки

Y – вказівний палець правої руки

Z – середній палець правої руки

Тепер якщо ви направите свій великий палець направо, вказівний вверх, то середній буде показувати на вас.

Спробуємо тепер вибрати координати так, що трикутник був по середині екрану:

```
glBegin (GL_TRIANGLES); // для примітиву трикутник
glVertex3f (-1.0, -1.0, 0.0);
glVertex3f (1.0, -1.0, 0.0);
glVertex3f (0.0, 1.0, 0.0);
glEnd ();
```

Перша вершина (-1.0, -1.0, 0.0). Це означає, що точка буде на екрані в позиції (-1.0, -1.0). Відео карта трактує екранні координати так, що центр екрану це 0, а крайні точки -1 та 1 по X та Y. Ось що виходить на широкоформатному моніторі (рис. 8.6):

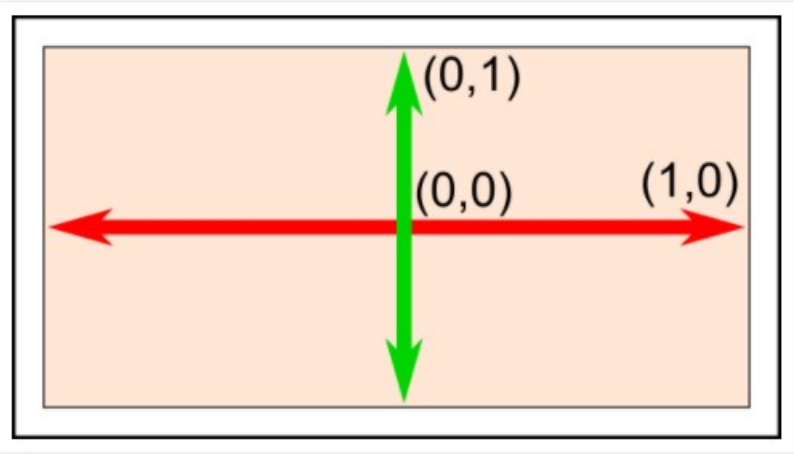


Рис. 8.6. - Трактування координат відео картою

Не дуже зручно, але і поміняти це не можливо. Тому наш трикутник буде розміщений за такими координатами: (-1, -1) – це нижній лівий куток, (1, -1) – нижній правий куток, (0,1) – верхня центральна точка.

В даний код до побудови трикутника перед glBegin можна ще добавити 2 рядки коду, для того аби очистити попередні побудови, якщо такі були:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 // очищення екрану та буферу глибина
glLoadIdentity();     // скидання перегляду
```

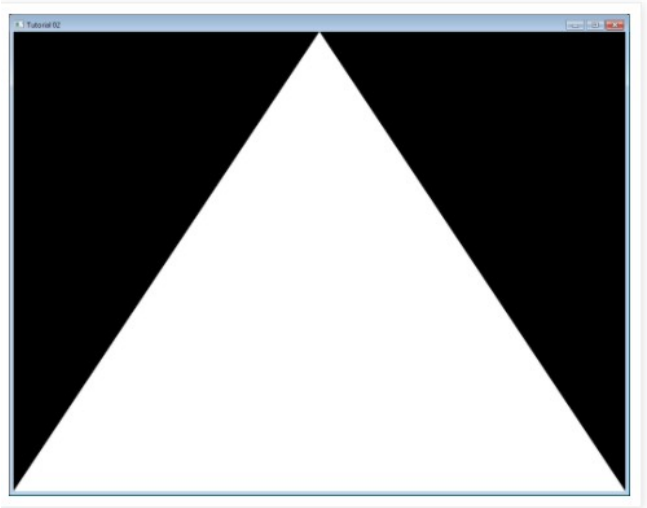
Саме виклик функції glLoadIdentity() встановлює початок системи координат в центр екрану.

Тому, маємо

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glBegin (GL_TRIANGLES);
glVertex3f (-1.0, -1.0, 0.0);
glVertex3f (1.0, -1.0, 0.0);
glVertex3f (0.0, 1.0, 0.0);
```

```
    glEnd ();
```

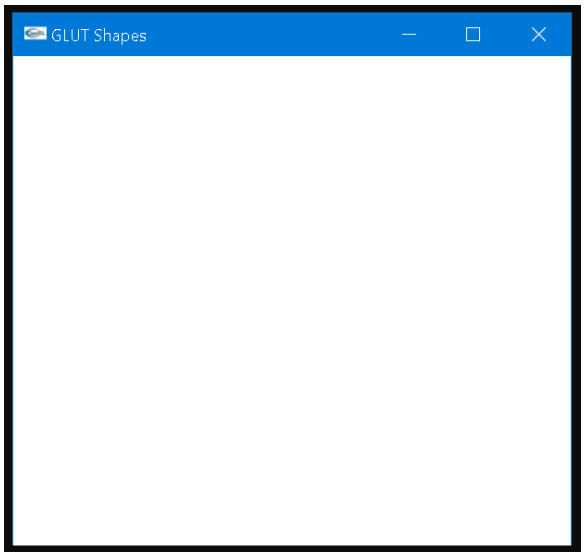
Результатом буде білий трикутник в центрі екрану:



Розглянемо побудову квадрата (чотирикутника):

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
    glBegin(GL_QUADS);
    glVertex3f(-1.0f, 1.0f, 0.0f);  // зліва зверху
    glVertex3f( 1.0f, 1.0f, 0.0f);  // справа зверху
    glVertex3f( 1.0f,-1.0f, 0.0f);  // справа знизу
    glVertex3f(-1.0f,-1.0f, 0.0f);  // зліва знизу
    glEnd();
 glFlush ();
```

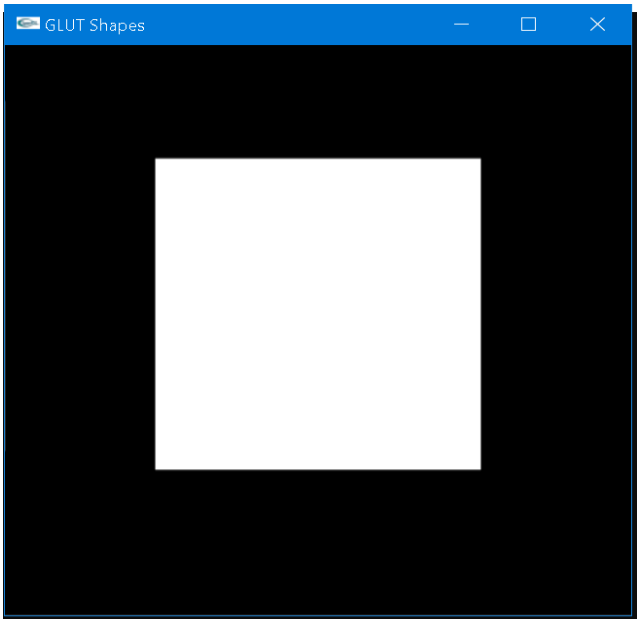
Результат, білий квадрат на весь екран:



Для того, аби змінити розміри фігури – потрібно змінювати значення координат вершин від -1 до 1. Наприклад, побудуємо квадрат в центрі екрану:

```
glBegin(GL_QUADS);
glVertex3f(-0.5f, 0.5f, 0.0f);
glVertex3f( 0.5f, 0.5f, 0.0f);
glVertex3f( 0.5f,-0.5f, 0.0f);
glVertex3f(-0.5f, -0.5f, 0.0f);
glEnd();
```

Результат:



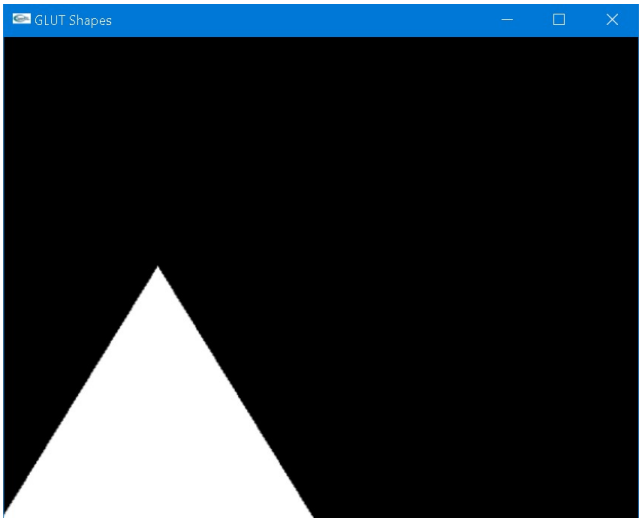
Переміщення примітивів

Для переміщення примітивів на екрані, потрібно використати функцію glTranslate(x, y, z) – вона переміщає осі координат на вказані значення. Переміщення осей координат здійснюється не відносно центру екрану, а від їх біжучого положення.

Наприклад, перемістимо вправо вниз трикутник, який був побудований в центрі екрану, на певні значення:

```
glTranslatef(-0.5f,-1.0f,-0.5f);
glBegin(GL_TRIANGLES);
glVertex3f( 0.0f, 1.0f, 0.0f);
glVertex3f(-1.0f,-1.0f, 0.0f);
glVertex3f( 1.0f,-1.0f, 0.0f);
glEnd();
```

Результат:



Аналогічно виконується переміщення інших примітивів.

Зафарбовування примітивів кольорами

Колір для примітиву можна задавати, задаючи кольори для вершин: однотонні, кожній вершині різний, градієнтом і т.д.

Для завдання поточного кольору вершини використовуються команди:

glColor [3 4] [b s i f] (components: GLtype)

glColor [3 4] [b s i f]v (components: ^GLtype)

Перші три параметри задають R, G і B компоненти кольору, а останній параметр визначає коефіцієнт непрозорості (так звану альфакомпоненту). Якщо в назві команди зазначений тип f (float), то значення всіх параметрів повинні належати відрізку [0,1], при цьому за замовчуванням значення альфа-компоненти встановлюється рівним 1, що відповідає повній непрозорості. Тип ub (unsigned byte) вказує, що значення повинні лежати у відрізку [0,255].

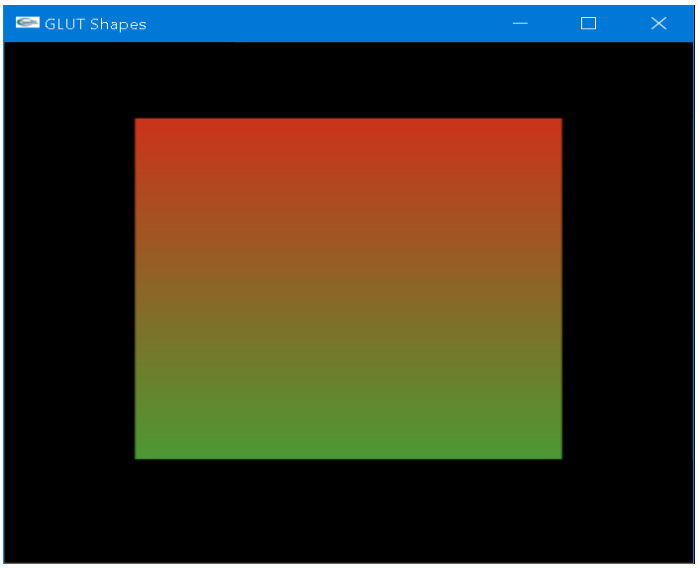
Якщо для зафарбовування вершин використовується функція glColor3f(r, g, b). Три параметри функції вказують насиченість червоним, зеленим та синім кольорами. Кожен з инх може приймати значення від 0.0f до 1.0f.

Для прикладу з теми № 7 – вершини трикутника мали різний колір, тому трикутник зафарбувався у 3 різні кольори, кожен колір виходив зі своєї вершини і зливався з іншими. Вершини чотирикутника були зафарбовані всі одним кольором, тому чотирикутник вийшов жовтий.

Для того, щоб зафарбувати фігуру кольором градієнту, один колір переходить в інший, це можна зробити до прикладу так, задати колір для двох верхніх вершин та інший колір для двох нижніх вершин:

```
glBegin(GL_QUADS);
 glColor3f(0.8,0.2,0.1);
 glVertex3f(-0.6f, 0.6f, 0.6f);
 glVertex3f( 0.6f, 0.6f, 0.6f);
 glColor3f(0.3,0.6,0.2);
 glVertex3f( 0.6f,-0.6f, 0.6f);
 glVertex3f(-0.6f,-0.6f, 0.6f);
 glEnd();
```

Результат:



Обертання примітивів

Обертання примітивів вздовж осі виконується за допомогою функції glRotatef(Angle, Xtrue, Ytrue, Ztrue). Перший параметр – це деяке число, кут який задає на скільки ви хочете повернути об’єкт. Xtrue, Ytrue, Ztrue – параметри які можуть набувати значень 0.0f або 1.0f. Якщо один з параметрів рівний 1.0f – це означає, що OpenGL буде повертати об’єкт вздовж тієї осі. Тому, якщо маємо, наприклад, наступний запис glRotatef(10.0f,0.0f,1.0f,0.0f), він означає, що об’єкт буде повертатися на 10 градусів по осі Y. Якщо напишемо glRotatef(5.0f,1.0f,0.0f,1.0f) – об’єкт буде повертатися на 5 градусів по обох осях X та Z.

Детальніше можна подивитися приклади за посиланням [15].

Продовження вивчення графіки OpenGL міститься у додаткових темах цього підручника, але при бажанні освоїти цю графіку ще глибше можна використати посилання вище, з прикладами та поясненням.

Питання для самоконтролю

1. З чого складається графічна сцена?
2. Коли GLUT викликає функцію оновлення зображення у вікні?

3. Що таке статичне зображення, динамічне зображення?
4. Що забезпечує типова функція оновлення зображення?
5. Для чого програма викликає команду glClear?
6. Що таке вершина і чим вона характеризується в OpenGL?
7. Що таке атрибути вершини?
8. Як визначається положення вершин?
9. Прокоментуйте команду glVertex?
10. Які атрибути та команди повязані з кольором та текстурою?
11. Які атрибути та команди для точок, ліній. Чотирикутників, багатокутників?
12. Для чого використовуються так звані операторні дужки?
13. Як відбувається визначення примітива чи послідовності примітивів?
14. Які примітиви дозволяє будувати OpenGL?
15. У яких режимах можна побудувати трикутник, прямокутник?
16. Які системи координат використовуються в OpenGL?
17. Яка система координат вважається основною?
18. Як саме формується спосіб представлення системи координат за правилом правої руки?
19. Як вибрати координати, щоб трикутник обудувати по центру екрану?
20. Яку функцію потрібно викликати, щоб втановити початок системи координат?
21. Розгляньте та прокоментуйте приклади побудови трикутника та чотирикутника білого кольору.
22. Яка функція в OpenGL використовується для переміщення примітивів?
23. Як відбувається переміщення осей координат?
24. Розгляньте та прокоментуйте приклад переміщення трикутника.
25. Як задається колір для примітиву?

26. Що таке компоненти кольору та коефіцієнти непрозорості?
27. Як зафарбувати фігуру кольором градієнту?
28. Розгляньте та прокоментуйте приклад фарбування чотирикутників градієнтом.
29. За допомогою якої функції виконується обертання примітивів вздовж осі?

ДОДАТКОВІ ТЕМИ, ЯКІ МОЖУТЬ БУТИ ВИНЕСЕНІ НА САМОСТІЙНЕ ОПРАЦЮВАННЯ

Матеріал даної частини підручника вивчається в паралельному до програмування курсі «Вибрані питання теоретичної інформатики та інформаційних технологій», але якщо в майбутньому навчальний план зміниться, то дані теми можна використати на самостійне опрацювання в межах курсу «Програмування».

Тема 9. Повторення: динамічні масиви, вкладені цикли

Динамічні масиви

Для того, щоб динамічно створити одновимірний масив, потрібно, по-перше, оголосити у програмі вказівник на тип, що відповідає типу елементів цього масиву, а, по-друге, виділити пам'ять під масив, використовуючи одну з функцій malloc або calloc. Нижче розглянемо приклад програми, де динамічно створено одновимірний цілочисловий масив.

Приклад 1.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int  *a, n;  // вказівник на масив, розмір масиву
printf("Input a size of an array: ");
scanf("%d", &n);
// виділяємо пам’ять під масив
a = (int*)malloc(n * sizeof(int));
// якщо помилка, то виходимо з програми
if (!a)
{ printf("Error: there is no memory.\n");
   return 0; }
```

```
// вводимо елементи масиву
printf("Input elements of the array: ");
for (int i = 0; i < n; ++i)
   scanf("%d", &a[i]);
// можно ввести елементи масиву і так
printf("Input elements of the array: ");
for (int i = 0; i < n; ++i)
   scanf("%d", a + i);
// щось робимо з масивом
printf("You input the array: ");
for ( int i = 0; i < n; ++i)
   printf("%d ", a[i]);
printf("\n");
// вивільняємо виділену пам’ять
free(a);
return 1; }
```

Пам'ять під багатовимірні динамічні масиви виділяється по рядках, починаючи з першого індексу. Це робиться для того аби забезпечити застосування операції індексування [ ], стільки разів яка розмірність багатовимірного масиву. В цьому випадку тип динамічного масиву оголошується як вказівник, який містить оператор звернення за адресою «*» стільки разів, яка розмірність масиву. Наприклад, вказівники на двовимірних та трьохвимірний цілочислові масиви можуть бути оголошені таким чином:

```
int  **a;       // вказівник на двовимірний масив
int  ***b;     // вказівник на трьохвимірний масив
```

Наведемо приклад динамічного створення та знищення двовимірного масиву.

Приклад 2.

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{ int **a, n, m; // вказівник на масив, розмірність масиву
int i, j; // індекси елементів матриці
printf("Input two sizes of an array: ");
scanf("%d%d", &n, &m);
// виділяємо пам'ять для вказівників на рядки
a =  (int**)malloc(n * sizeof(int*));
// якщо помилка, то виходимо з програми
if (!a)
{ printf("Error: there is no memory.");
   return 0; }
// виділяємо пам'ять для рядків
for (i = 0; i < n; ++i)
{   a[i] = (int*)malloc(m * sizeof(int));
   // якщо помилка, то звільняємо пам’ять і виходимо з програми
   if (!a[i])
   { for (j = 0; j < i; ++j)
         free(a[j]);
      free(a);
      printf("Error: there is no memory.\n");
      return 0;
   } }
// вводимо елементи масиву
printf("Input elements of the matrix:\n");
for (i = 0; i < n; ++i)
  for (j = 0; j < m; ++j)
    scanf("%d", &a[i][j]);
// щось робимо з матрицею
printf("You input the matrix:\n");
for (i = 0; i < n; ++i)
```

```
{ for (j = 0; j < m; ++j)
      printf("%d ", a[i][j]);
   printf("\n"); }
// вивільняємо пам’ять
for (i = 0; i < n; ++i)
   free(a[i]);
free(a);
return 1; }
```

(джерело [16])

Задачі з використанням вкладених циклів

Будь-який складний алгоритм завжди складається з декількох простих алгоритмів. Потрібно навчитись їх бачити та комбінувати.

1. Задачі з побудовою алгоритму. Приклади за джерелом [17] .
2. Задачі на знаходження суми сум, суми добутків і т.д. Побудова таблиці значень для перевірки програми.

Приклад 3.

$$\sum_{i=0}^{n=3}\sum_{j=1}^{m=4}(i+2\cdot j)=¿((0+2\cdot 1)+(0+2\cdot 2)+(0+2\cdot 3)+(0+2\cdot 4))+((1+2\cdot 1)+(1+2\cdot 2)+(1+2\cdot 3)+(1+2\cdot 4))+(($$

Таблиця значень суми сум:

i		0					1					2					3			
j		1	2	3	4		1	2	3	4		1	2	3	4		1	2	3	4
S1	0	2	6	12	20	0	3	8	15	24	0	4	10	18	28	0	5	12	21	32
S2	0				20	20				44	44				72	72				104

Код програми буде мати вигляд:

```
int main()
{  int S2=0, n=3, m=4;
   for (int i=0; i<=n; i++)
```

```
{  int S1=0;
   for(int j=1; j<=m; j++)
      S1+=i+2*j;
 S2+=S1;     }
cout << "Syma sym = " <<S2;
return 0; }
```

Для перевірки обрахунків можна виводити проміжні значення.

Приклад 4.

$$\sum_{i=0}^{n=3}\prod_{j=1}^{m=4}(i+2\cdot j)=¿((0+2\cdot 1)\cdot(0+2\cdot 2)\cdot(0+2\cdot 3)\cdot(0+2\cdot 4))+((1+2\cdot 1)\cdot(1+2\cdot 2)\cdot(1+2\cdot 3)\cdot(1+2\cdot 4))+((2+2$$

Таблиця значень сума добутків буде будуватися аналогічно прикладу 3.

Код програми буде мати аналогічний вигляд, відмінність тільки в тому, що змінна добутку повинна мати початкове значення 1, а в середині циклу буде знак множення, а не додавання:

```
int main()
{  int S=0, n=3, m=4;
   for (int i=0; i<=n; i++)
   {  int D=1;
      for(int j=1; j<=m; j++)
         D*=i+2*j;
    S+=D;     }
   cout << "Syma sym = " <<S;
   return 0; }
```

Аналогічно обчислюється добуток добутків та добуток суми (зробити та перевірити самостійно).

Питання для самоконтролю

1. Що таке динамічний масив?
2. Як оголосити динамічний масив?
3. Прокоментуйте приклад 1.

4. Як виділяється пам'ять під багатовимірний динамічний масив?
5. Які функції використовуються для виділення пам'яті?
6. Навідіть приклад оголошення одновимірного, двохвимірного та тривимірного динамічного масиву цілого типу.
7. Як знищити (вивільнити пам'ять) динамічний масив?
8. Які функції використовуються для вивільнення пам'яті з під динамічного масиву?
9. Які особливості опрацювання динамічний масивів?
10. Що таке вкладені цикли? Наведіть приклади.
11. Як розв'язують задачі на вкладені цикли?
12. Прокоментуйте приклад 3 та 4.
13. Які особливості початкових значень змінних, що відповідають за обчислення суми та добутку.
14. Пркоментуйте таблиці зі значеннями.Як вони обчислюються?

Тема 10. Програмування математичних функцій за допомогою рядів

Обчислення синуса

Для того, щоб запрограмувати обчислення будь якої математичної функції на C++, ми підключаємо бібліотеку math.h. Вона містить функції, які можна використовувати для обчислення, але ми повинні розуміти як же вони прописані там у бібліотеці. Тому розглянемо деякі можливі способи.

Як відомо, функція sin(x) може бути обчислена у вигляді ряду:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots = \sum_{i=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

На практиці при обчисленні значення функції sin(x) відповідний ряд обмежують або шукають з заданою точністю. В даному випадку розглянемо приклад з обмеженням, тобто розглядають наближений вираз:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{(-1)^N x^{2N+1}}{(2N+1)!} = \sum_{n=0}^{N} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Але, оскільки доданок під знаком суми у чисельнику містить піднесення до степеня, у знаменнику – знаходження факторіалу, бачимо, наприклад, щоб запрограмувати знаменник потрібно використати рекурсію, а при великих значеннях N, програмі не вистачить ресурсів рахувати факторіал, аналогічно зі степенем. Тому потрібно використати ідею спрощення обчислень, тобто максимально звести обчислення до звичайних дій: додавання, віднімання, множення та ділення. Використаємо спосіб рекурентних співвідношень, знайдемо рекурентні формули для елементів цих рядів, або по іншому, у скільки разів потрібно змінити перший елемент ряду, щоб отримати наступний:

$$q_n = \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

$$q_{n-1} = \frac{(-1)^{n-1} x^{2(n-1)+1}}{(2(n-1)+1)!}$$

Знайдемо $\frac{q_n}{q_{n-1}} = \frac{(-1)^n x^{2n+1}}{(2n+1)!} \cdot \frac{(2n-2+1)!}{(-1)^{n-1} x^{2n-2+1}} = ¿$

$$¿ \frac{(-1)^n x^{2n+1}}{(2n+1)!} \cdot \frac{(2n-1)!}{(-1)^n (-1)^{-1} \cdot x^{2n-1}} = \frac{(-1) x^2 (2n-1)!}{(2n-1)! \cdot 2n \cdot (2n+1)} = \frac{(-1) x^2}{2n \cdot (2n+1)}$$

Дану частинку можемо використати при програмуванні. Аргумент x вводиться користувачем з клавіатури, параметр N може також бути введене користувачем з клавіатури, а може задаватися цілочисловою константою.

Програма на C++ буде мати вигляд:

```
#include <iostream>
using namespace std;
const int N=100;
int main()
{
double x, q, s=0;
int n;
```

```
cout << "Enter x = ";
cin>>x;
q=x;
// Обчислення синуса:
for (n=1; n<=N; n++)
        s+=q;
        q* = (-1)*x*x/(2*n)*(2*n+1);
    //Результат:
    cout<<"sin("<<x<<") "<<s<<end1;
    return 0;
}
```

Для того, щоб перевірити чи правильно рахується наближене значення функції за допомогою даної програми, можна в код дописати ще обчислення значення функції від того ж аргументу, тільки використовуючи вбудовану функцію sin(x) бібліотеки math.h.

Ряд для експоненти: використання класу та динамічного масиву

Розглянемо ще один приклад. Програмування рядів використовуючи ідею рекурентних елементів, але елементи ряду вписуються у динамічний масив, який є полем класу. Потім всі елементи ряду додаються як елементи масиву. Масив заповнюється при створенні відповідного об’єкта.

Ряд для експоненти має вигляд:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Аналогічно попередньому прикладу знайдемо частку:

$$\frac{q_n}{q_{n-1}} = \frac{x^n}{n!} \cdot \frac{(n-1)!}{x^{n-1}} = \frac{x^n \cdot (n-1)!}{n \cdot (n-1)! \cdot x^{n-1}} = \frac{x}{n}$$

Аргументами конструктору передається верхній індекс ряду int n і аргумент експоненти double x.

Програма на C++ буде мати вигляд:

```
#include <iostream>
using namespace std;
class MyExp{
public:
int n;
double *p;
MyExp (int i, double x)
{ n=i;
p=new double[n+1];
p[0]=1;
for (int k=1; k<=n; k++)
    p[k]=p[k-1]*x/k;
}
            MyExp ()
{delete [ ] p;}
}
int  main ( )
{ int n, i;
double x, s=0;
cout<<"enter n= ";
cin>>n;
cout<< "enter x= ";
cin>>x;
MyExp obj (n, x);
for (i=0; i<=n; i++)
s+=obj.p [i] ;
cout<<"exp ("<<x<<") = "<<s<<end1;
return 0;
}
```

Результат виконання програми для прикладу:

enter n= 100

enter x= 1

exp(1)= 2.71828

Користувачем вводяться значення для верхньої границі ряду і аргумент експоненти (змінні *int* n і *double* x в головному методі програми). Ці аргументи передаються конструктору при створенні об’єкту командою *MyExp obj(n, x).* Конструктором перший аргумент присвоюється в якості значення полю n об’єкта *obj*. Це ж значення використовується коли виділяється пам'ять під динамічний масив, вказівник на перший елемент якого присвоюється в якості значення полю p об’єкта *obj*. Після виділення місця під масив виконується заповнення елементами масив. В якості значення елементу з нульовим індексом присвоюється 1, а інші елементи заповняються на основі попереднього значення згідно отриманої частки. У деструкторі виконується очищення пам’яті, виділеної під динамічний масив.

Питання для самоконтролю

1. Яку бібілотеку потірбно підключити для обчислення будь-якої математичної функції на C++?
2. За допомогою якого математичного ряду може бути обчисленна функція sin(x)?
3. Для чого відповідний ряд обмежують?
4. Коли і для чого у програмуванні максимально зводять обчилення до звичайних дій?
5. Які дії називаються звичайними?
6. Який спосіб використовують до зведення обчислень звичайними діями?
7. Прокоментуйте обчислення за формулами.
8. Прокоментуйте код програми для обчислненн функції sin(x).
9. Як перевірити чи правильно ви запрограмували обчислення?

10. За допомогою якого математичного ряду може бути обчисленна функція exp(x)?
11. Розгляьне та прокоментуйте приклад для функції exp(x).
12. Поясніть використання класу у прикладі коду дл функції exp(x).
13. Дя чого використаний динамічний масив?
14. Які ще особливості коду ви помітили?

Тема 11. Шаблони (продовження) та перевантаження операторів

Перевантаження операторів

Розглянемо приклад з перевантаженням операторних функцій в узагальнених класах. Приклад досить простий: у програмі оголошується 2 різних класи A і B. Клас A має цілочислове поле k, а клас B має поля x та y типу double. В кожному з класів перевантажується оператор додавання. Оператор додавання перевантажується так, що в результаті додавання двох об’єктів отримуємо об’єкт того ж класу. При цьому відповідні поля об’єктів додаються. Кожен клас має метод show() для відображення значень поля чи полів. Також у програмі оголошується узагальнений клас MyTemp1, в якому є одне поле res (об’єкт класу A чи B). В узагальненому класі MyTemp1 також перевантажується оператор додавання: при додаванні об’єктів класу MyTemp1 додаються їх поля res. Для відображення значень полів об’єкта-поля res використовується метод show(). При його реалізації викликається метод з таким же ім’ям із об’єкта res. Програмний код у прикладі 1.

Приклад 1.

```
#include <iostream>
using namespace std;
class A
{
   public:
```

```
    int k;
    A operator+(A obj)
    {
      A tmp;
      tmp.k = k + obj.k;
      return tmp;
    }
    void show ()
    {
       cout<< "k = " <<k<<endl;
    }
};
class B
{
    public:
    double x,y;
    B operator+(B obj)
    {
      B tmp;
      tmp.x = x + obj.x;
      tmp.y = y + obj.y;
      return tmp;
    };
    void show ()
    {
       cout<< "x = " <<x<<endl;
       cout<< "y = " <<y<<endl;
    }
};
template <class X> class MyTemp1
```

```
{
   public:
   X res;
   MyTemp1 operator+(MyTemp1 obj)
   {
      MyTemp1 tmp;
      tmp.res = res + obj.res;
      return tmp;
    }
    void show()
    {
       res.show();
    }
};
int main()
{
   MyTemp1<A> a1,a2,a3;
   MyTemp1<B> b1,b2,b3;
   a1.res.k = 1;
   a2.res.k = 2;
   a3 = a1 + a2;
   a3.show();
   b1.res.x = 10.1;
   b1.res.y = 100.1;
   b2.res.x = 20.2;
   b2.res.y = 200.2;
   b3 = b1 + b2;
   b3.show();
   return 0;
}
```

В результаті виконання програми отримуємо:

k = 3

x = 30.3

y = 300.3

Варто відмітити, що оскільки в узагальненому класі метод show() через виклик методу show() з об’єкту res, в якості типу-параметру X можна використовувати тільки імена класів, в яких оголошений такий метод (та обов’язково перевантажений оператор додавання).

Перестановка елементів масиву

В наступному прикладі створюється узагальнений клас з полем-масивом, тип елементів якого є параметром узагальненого класу. В узагальненому класі перевантажений оператор [ ] так, що з його допомогою можна звертатися до елементів поля-масиву (при виході за межі масиву виконується циклічний перехід на початок масиву), а також оператор інкременту, дія якого зводиться до циклічної перестановки елементів у масиві (перший елемент зміщується на місце другого, другий зміщується на місце третього і т.д., а останній елемент переходить на місце першого елемента). В класі також описується метод show() для відображення елементів поля-масиву. Програмний код у прикладі 2.

Приклад 2.

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int n = 10;
template <class X> class MyClass
{
   public:
   X array[n];
   X operator[](int k)
```

```
    {
        return array[k%n];
    }
    MyClass operator++()
    {
        X tmp;
        int i;
        tmp = array[n-1];
        for (i=n-1; i>0; i--)
            array[i]=array[i-1];
        array[0]=tmp;
        return *this;
    }
    void show()
    {
      for (int i=0; i<n; i++)
            cout<<array[i]<<" ";
        cout<<endl;
    }
    MyClass()
    {
       for (int i=0; i<n; i++)
            array[i] = (X)(rand()%25+100);
    }
};
int main()
{
    MyClass <int> obj1;
    obj1.show();
    for (int i=n; i<2*n; i++)
```

```
        cout <<obj1[i]<< " ";
    cout<<endl;
    MyClass <char> obj2;
    obj2.show();
    ++obj2;
    obj2.show();
    return 0;
}
```

Результат виконання програми може мати наступний вигляд:

```
116 117 109 100 119 124 103 108 112 114
116 117 109 100 119 124 103 108 112 114
i x j f o t x u f o
o i x j f o t x u f
```

При створенні об’єкта викликається конструктор для випадкового заповнення масиву-поля створюваного об’єкта. Масив заповняється наступним чином: генерується випадкове ціле число, після чого це значення перетворюється до типу, який мають елементи масиву.

В головному методі програми проілюстрований процес створення об’єктів з різним типом-параметром, індексація об’єктів, результат вказування індексу за межами розміру масиву, а також циклічна перестановка елементів масиву.

Пошук співпадінь

Розглянемо задачу про пошук співпадінь деякого значення зі значеннями елементів масиву. Масив реалізовано у вигляді поля узагальненого класу. Для пошуку співпадінь створюється узагальнена функція, першим аргументом якої вказується значення, що перевіряється на предмет співпадіння, а другий аргумент – об’єкт з масивом, в якому виконується пошук. В результаті виклику функції на екран виводиться

повідомлення про кількість співпадінь. Відповідний програмний код у прикладі 3.

Приклад 3.

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int n = 20;
template <class X> class Elements
{
   public:
   X array[n];
   X operator[](int k)
   {
      return array[k];
   }
   Elements ()
   {
     for (int i=0; i<n; i++)
     {
        array[i] = (X)(rand()%20+100);
        cout<<array[i]<<" ";
     }
     cout<<endl;
   }
};
template <class X> void FindElement(X s, Elements<X> obj)
{
   int i, count=0;
   for (i=0; i<n; i++)
   if (s==obj[i]) count++;
```

```
    cout<<"resultat = "<<count<<endl;
};
int main()
{
    Elements<int> a;
    FindElement(101,a);
    Elements<char> b;
    FindElement('f',b);
    return 0;
}
```

В узагальненому класі Elements перевантажується оператор [ ] для індексації об’єктів. При створенні об’єкту елементи масиву заповняються з допомогою генератора випадкових чисел з одночасним виведенням значень на екран.

Як вказувалося, в узагальненій функції FindElement() два аргументи, причому за типом першого аргументу визначається тип параметра для об’єкту класу Elements, що передається другим аргументом функції. Пошук співпадінь здійснюється шляхом послідовного перебору елементів масиву, що є полем об’єкту – другого аргументу функції FindElement().

В головному методі програми на основі узагальненого класу Elements створюються два об’єкти. На їх основі ілюструються методи виклику функції FindElement(). В результаті виконання програми отримуємо, наприклад, наступний результат:

101 107 114 100 109 104 118 118 102 104 105 105 101 107 101 111 115 102 107 116

resultat = 3

o h f q p f e t v s k j o v m p k w s r

resultat = 2

Відмітимо також, що перевантаження оператора [ ] виконувалося виключно ради зручності.

Питання для самоконтролю

1. Що таке шаблони у програмуванні?
2. Що таке узагальнений клас та узагальнена функція?
3. Що таке перевантаження операторів?
4. Як відбувається перевантаження оператора?
5. Розгляньте та прокоментуйте приклад 1.
6. Розгляньте та прокоментуйте результат виконання програми для прикладу 1.
7. Які особливості коду ви для себе відмітили?
8. Як створити узагальнений клас з полем-масивом тип елементів якого є параметром узагальненого класу?
9. Як перевантажили оператор [ ]?
10. Як перевантажили оператор інкременту?
11. Розгляньте та прокоментуйте приклад 2.
12. Розгляньте та прокоментуйте результат виконання програми для прикладу 2.
13. Як заповняється масив?
14. Опишіть особливості узагальної функції у прикладі пошуку співпадінь.
15. Розгляньте та прокоментуйте приклад 3.
16. Розгляньте та прокоментуйте результат виконання програми для прикладу 3.

Тема 12. Робота з текстовими файловими потоками у стилі C++

У C++ стандартна бібліотека містить три класи потоків для роботи з файлами:

- ifstream – вхідні файлові потоки (для зчитування);

- ofstream – вихідні файлові потоки (для записування);
- fstream – двонапрямлені файлові потоки (для зчитування та записування).

При роботі з файлами цих класів треба долучати до програми заголовочний файл <fstream.h>.

Об’єкти файлових потоків створюються за допомогою конструкторів відповідних класів, наприклад:

```
// Створення (відкриття) вихідного потоку (записування)
ofstream outfile("Test.dat");
// Створення (відкриття) вхідного потоку (зчитування)
ifstream fin ("Test.dat");
// Створення (відкриття) введення-виведення (записування і зчитування)
fstream f_in_out("Test.dat");
```

До створених у такий спосіб потоків можна застосовувати операції “помістити в потік” (<<) і “взяти з потоку” (>>). Перевага цих операцій, які працюють з текстовими файлами є простота використання і автоматичне розпізнавання типів даних. Розглянемо, наприклад, такий код:

Приклад 1.

```
int i=1, j=25, il, j1; double a=25e6, al;
char s[40], sl[40];
strcpy(s, "Іванов");
ofstream fout ("Test.dat"); // Створення файла як вихідного потоку
if(!fout)
{ShowMessage("Файл не вдається створити"); return;}
fout << i << ' ' << j << ' ' << a << ' ' << s << endl;
fout.close(); // Закриття файла
ifstream fin("Test.dat"); // Відкриття файла як вхідного потоку
if(!fin)
```

```
{ShowMessage("Файл не вдається відкрити");
return;}
fin >> i1 >> j1 >> a1 >> s1;
fin.close(); // Закриття файла
```

У цьому коді створюється файл з ім’ям Test.dat, і до нього записуються у текстовому вигляді два цілі числа – i та j, дійсне число a й рядок s, який містить одне слово, після чого маніпулятором потоку endl здійснюється перехід до нового рядка. Причому записування всіх цих даних здійснюється одним оператором, що містить зчеплені операції “помістити в потік”. Якщо порівняти це з аналогічними кодами, то можна переконатися в компактності й простоті застосування цієї операції. Після того як файл закриється, в ньому буде записано текст “1 25 2.5e+07 Іванов”. Наступні команди створюють вхідний потік, пов’язаний з цим файлом, і за допомогою операції “узяти з потоку” читають дані.

Послідовне записування до файлу і зчитування з файлу

Для записування певної інформації до файлу використовується операція “помістити в потік” (за аналогією з cout):

```
<< блок1 << блок2 << ... << блокN;
```

Для зчитування з файлу певної інформації використовується операція “взяти з потоку” (за аналогією з cin>>)

```
filename >> блок1 >> блок2 >> ... >> блокN;
```

Однак при читанні з файлу за допомогою потоків зчитування здійснюється до пробілу чи символу нового рядка. Тому це зчитування придатне лише для чисел та окремих слів. Для зчитування цілого рядка використовується функція getline(), яка має прототип

```
getline (char*, int, char='\n');
```

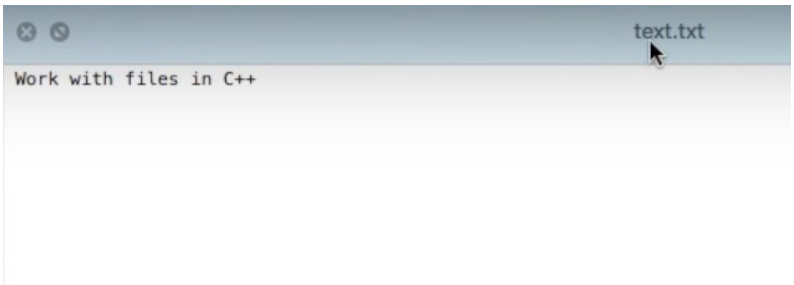
Наприклад, оператор

fin.getline(s, n); читає з потоку не більше за n-1 символів й записує їх до змінної s. Для роботи з файловим потоком слід до програми долучити бібліотеку <fstream.h>.

Розглянемо аналогічний приклад запису даних у файл:

```
#include <iostream>
#include <fstream>

using namespace std;

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

    ofstream file;
    file.open("text.txt");
    file << "Work with files in C++";
    file.close();

    cin.get();
    return 0;
}
```

Після запуску на виконання отримаємо записані дані у файл.



Або можна написати скорочено так, забираємо команду open:

```
#include <iostream>
#include <fstream>

using namespace std;

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

    ofstream file("text.txt");
    file << "Work with files in C++";
    file.close();

    cin.get();
    return 0;
}
```

Читання з файлу (аналогічно попередньому) реалізовано наступним чином:

```
#include <iostream>
#include <fstream>

using namespace std;

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

    char text[50];

//    ofstream file("text.txt");
//    file << "Work with files in C++";
//    file.close();

    ifstream file("text.txt");
    if (!file.is_open())
        cout << "Error! File is not found!" << endl;
    else {
        file >> text;
        cout << text << endl;

        file.getline(text, 50);
        cout << text << endl;
        file.close();
    }

    cin.get();
    return 0;
}
```

Результат:

```
Work
 with files in C++
```

Закоментувавши

```
//        file >> text;
//        cout << text << endl;
```

Отримаємо:

```
Work with files in C++
```

Якщо після цього, створені файли у папці витерти і не створювати програмно (закоментувала з рядки), то повинно спрацювати повідомлення про помилку:

```
#include <iostream>
#include <fstream>

using namespace std;

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

    char text[50];

//    ofstream textFile("text2.txt");
//    textFile << "Hi! Everything works great!";
//    textFile.close();

    ifstream file("text2.txt");
    if (!file.is_open())
        cout << "Error! File is not found!" << endl;
    else {
        file.getline(text, 50);
        cout << text << endl;
        file.close();
    }

    cin.get();
    return 0;
}
```

Результат:

```
Error! File is not found!
```

Як було зазначено раніше, файл треба описати. У C++ це можна здійснити в такий спосіб:

ofstream filename("C:\Test.dat", ios::out);

Тут у подвійних лапках зазначено ім’я файлу на диску C, а filename – ім’я файлової змінної, тобто ім’я, за допомогою якого здійснюватиметься відкривання файлу у програмі. Якщо файл не існує, то його буде створено. Далі, після коми, слід зазначити режим відкривання файлу (табл. 1).

Таблиця 1.

Режими відкривання файлів

Режим	Значення режиму
ios::in	Відкрити файл для зчитування
ios::out	Відкрити файл для записування
ios::trunc	Вилучити дані з файла
ios::nocreate	Якщо файл не існує, відкривання файла не відбудеться
ios::ate	Після відкриття перемістити позицію зчитування-записування на кінець файла
ios::app	Відкрити для записування даних в кінець файла
ios::binary	Відкрити файл, вважаючи його за бінарний (не текстовий)

Режими ios::ate та ios::app є схожими, оскільки переміщують позицію зчитування-записування (файловий покажчик) на кінець файлу. Відмінність поміж ними полягає в тому, що режим ios::app дозволяє дописувати дані лише в кінець файлу, в той час як режим ios::ate просто переміщує покажчик на кінець файлу за аналогією роботи С-функції fseek(f,0,SEEK_END).

Імена констант відкриття є абревіатурами виконуваних дій: app – append (долучити), ate – at end (на кінець), trunc – trancate (урізати, відкинути) тощо.

Режими відкриття файлу за суттю є бітовими масками, а тому можна задавати два та більше режимів, поєднуючи їх операцією логічного “АБО”. Наприклад, щоб відкрити бінарний файл з ім’ям Test.dat як для зчитування, так і для записування, слід записати функцію

```
fstream filename("Test.dat",ios::in|ios::out|ios::binary);
```

Тут символ “|” поєднує два режими ios::in та ios::out.

Щоб перевірити правильність відкриття файлу, можна скористатися умовним оператором

```
if(!filename)
{ ShowMessage("Неможливо відкрити файл!");
return; }
```

Завершення роботи з файлом здійснюється за допомогою функції

```
filename.close();
```

Для детальнішого розуміння пропоную переглянути відеоролики:

1) Работа с файлами с++ [18].
2) Работа с файлами с++. Запись в файл. с++ ofstream. Изучение С++ для начинающих. Урок #115 [19].
3) Работа с файлами с++. Чтение из файла с++ ifstream. Изучение С++ для начинающих. Урок #116 [20].
4) Чтение и запись в файл с++ используя класс fstream с++. Изучение С++ для начинающих. Урок #118 [21].

Питання для самоконтролю

1. Які класи у С++ стандартної бібілотеки використовуються для роботи з файлами?
2. Для чого служить клас ifstream?
3. Для чого служить клас ofstream?
4. Для чого служить клас fstream?
5. Як створюються обєкти цих класів для записування (відкриття) вихідного потоку?
6. Як створюються обєкти цих класів для зчитування вихідного потоку?
7. Як створюються обєкти цих класів для введення-вивведення (записування і зчитування) вихідного потоку?
8. Які операції використовують для того аби «помістити у потік»?
9. Які операції використовують для того аби «взяти з потоку»?
10. Яка перевага таких опреацій?
11. Розглянути та прокоментувати приклад 1 та результат виконання програми.
12. Які операції використовують для послідовного записування у файл та зчитування з файлу? Наведіть приклади.
13. Яка функція використовуєтьс дл зчитування цілого рядка з файлу?

14. Розгляньте та прокоментуйте приклад запису даних у файл та результат виконання програми.
15. Розгляньте та прокоментуйте приклад читання даних з файлу та результат виконання програми.
16. Розгляньте та прокоментуйте приклад, коли спрацює код на виявлення помилки.
17. Які ви знаєте режими відривання файлів?
18. Що означає кожен з режимів?
19. Наведіть приклади коду відкривання файлів з різними режимами.

Тема 13. Довільне записування до файлу і зчитування з файлу. Запис у файл об’єкта класу. Бінарні файли

Довільне записування до файлу і зчитування з файлу

Розглянемо варіант довільного записування і зчитування даних. Це означає, що записування даних до файлу і зчитування цих даних здійснюватимуться з довільної зазначеної позиції файлу. Позиціонування (встановлення курсора на певну позицію) виконується за допомогою методу seekp() класу ifstream при записуванні даних до файлу, і за допомогою методу seekg() класу ofstream – при зчитуванні даних з файлу. Ці методи мають два аргументи: перший зазначає на скільки байтів відносно позиції, зазначеної у другому аргументі, слід зрушитися. Існують такі константні значення позицій:

ios::beg – початок потоку (встановлюється за замовчуванням);

ios::end – кінець потоку;

ios::cur – поточна позиція потоку.

Наприклад:

f.seekg(n); – позиціонування на n-й байт від початку файлу;

f.seekg(n, ios::cur); – позиціонування на n-й байт уперед від поточної позиції;

f.seekp(k, ios::end); – позиціонування на k-тий байт від кінця файлу;

f.seekp(0, ios::end); – позиціонування на кінець файлу.

Методи tellg() та tellp() повертають поточну позицію файлу для потоків введення та виведення відповідно.

Для створювання текстових файлів довільного доступу існує багато способів. Мова C++ не накладає вимог щодо вмісту текстових файлів, але записані у файлі рядки мають бути однакової фіксованої довжини. Це надає можливість легко визначати точне місцезнаходження будь-якого рядка відносно початку файлу. У такому файлі дане може бути записано в будь-яке місце та змінено без перезаписування всього файлу. Для організації такого записування використовується маніпулятор setw(num), який задає довжину поля виведення з num позицій. Для його використання слід долучити заголовний файл <iomanip.h>. Приклад його використання:

fout<<setw(15)<<name<<setw(20)<<surname<<setw(7)<<year<<endl.

Запис у файл об’єкта класу

Інформація за джерелами [22], [23].

Бінарні файли

У бінарному (двійковому) файлі число, на відміну від текстового, зберігається у внутрішньому його поданні. У двійковому форматі можна зберігати не лише числа, а й рядки та цілі інформаційні структури. Причому останні зберігати зручніше, завдяки тому що відсутня потреба явно зазначати кожен елемент структури, а зберігається вся структура як цілковита одиниця даних. Хоча цю інформацію не можна прочитати як текст, вона зберігається більш компактно і точно. Тому, що саме і в якій послідовності розміщено в бінарному файлі, має бути відомо програмі.

Робота з бінарними файловими потоками у стилі C++

Для роботи з бінарним форматом файлу використовується прапорець ios::binary.

Ці файли відкриваються здебільшого в режимі зчитування-записування. Для цього зручними є об’єкти класу fstream. Дані до бінарних файлів записуються за допомогою методу write() класу ofstream, а зчитуються – за допомогою методу read() класу ifstream.

Наприклад:

fout.write((char)*&A, sizeof(A));

fin.read((char)*&A, sizeof(A));

Перший параметр у цих функцій – адреса змінної A. Ці функції працюють з байтами (типу char). Для них немає жодного значення, в який спосіб організовано й що собою являють дані, – вони просто переносять їх побайтово з буфера до файлу та навпаки. Другий параметр – довжина змінної A у байтах. Тому перший оператор обчислює значення виразу sizeof(A) й копіює до файлу, пов’язаного з об’єктом fout, обчислену кількість байтів, розпочинаючи з вказаної адреси. Другий оператор копіює кількість байтів sizeof(A) з файлу fin до змінної A. У бінарному файлі також існує можливість довільного доступу до файлу за допомогою методів seekp() та seekg(). При повторному відкритті файлу з використанням тієї ж самої файлової змінної потік слід очищувати за допомогою методу clear(), наприклад: f.clear().

Для кращого розуміння пропоную скористатися посиланням [24], [25], [26].

Питання для самоконтролю

1. Що таке довільне записування у файл та зчитування даних з файлу та як воно відбувається?
2. За допомогою якого методу відбувається позиціонування курсора у файлі?

3. Наведіть та прокоментуйте приклади коду позиціонування курсора у файлі.
4. Які методи повертають поточну позицію файлу для потоків введення та виведення?
5. Якої довжини повинні бути рядки, що записані у файл? Що це дає?
6. Яка команда використовується для огранізації такого записування. Наведіть приклади.
7. Як записати у файл об’єкт будь-якого коритувацького класу?
8. Як зчитати з файлу об’єкт будь-якого коритувацького класу?
9. Які файли називають бінарними?
10. Яка відмінність бінарного файту від текстового?
11. Який прапорець показує, що робота буде вестися з бінарним форматом файлу?
12. За допомогою якого методу дані записуються у бінарний файл?
13. За допомогою якого методу дані зчитуються з бінарного файлу?
14. Як у бінарному файлі організована можливість довільного доступу?

Тема 14. Створення 3D-об’єктів вручну (заданням координат вершин), використовуючи вбудовані функції та з використанням масивів

Створення 3D-об’єктів вручну: побудова піраміди

Детальніше можна розглянути у джерелі за посиланням [27].

Не забувайте, що якщо будете писати одиниці, так як у прикладі за посиланням, то буде залито все білим, бо сітка дозволяє до одиниці координати, а одиниці це межі графічного екрану.

Приклад побудови піраміди та її зміщення і поворот відносно осі x, (за посиланням вище – поворот по осі y).

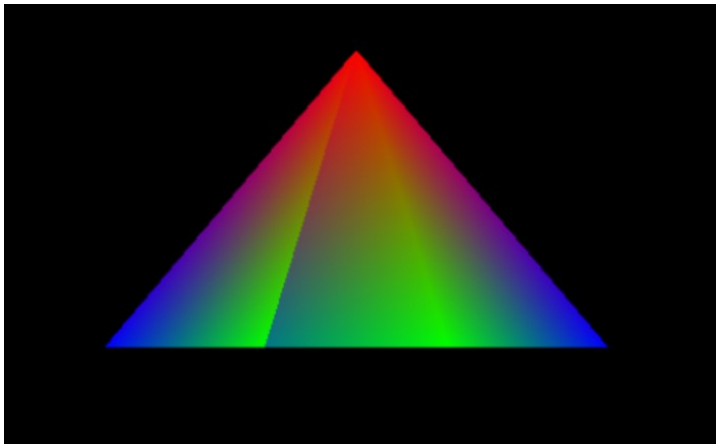
Приклад 1.

```
static void Drow(void)
```

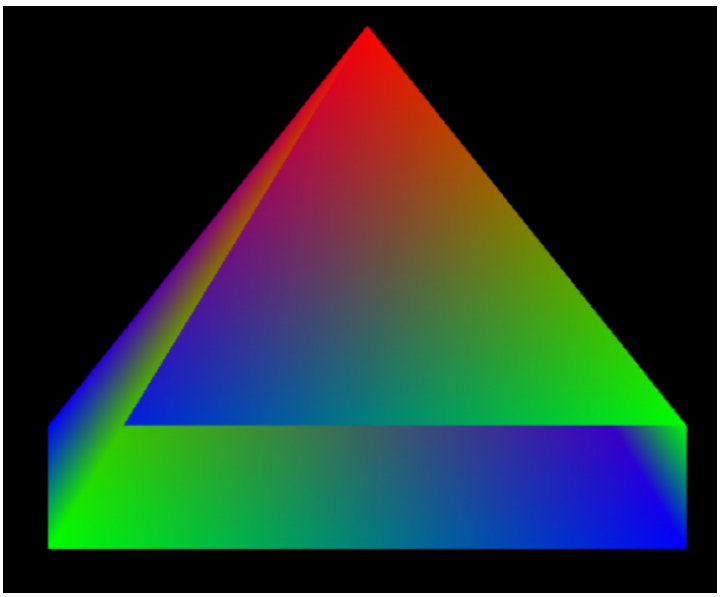
```
        {
          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // Очищення екрану та буфера глибини
        glLoadIdentity();                    // Скидання перегляду
        glTranslatef(-0.25f,0.2f,-0.15f);        // Зсув вліво і вглибину екрану
экрана
        glRotatef(15,0.8f,0.0f,0.0f);            // Обертання піраміди по осі X
        glBegin(GL_TRIANGLES);
              glColor3f(1.0f,0.0f,0.0f);                 // Червоний
              glVertex3f( 0.0f, 0.6f, 0.0f);          // Верх трикутника (Передня)
              glColor3f(0.0f,1.0f,0.0f);                 // Зелений
              glVertex3f(-0.6f,-0.6f, 0.6f);              // Лева точка
              glColor3f(0.0f,0.0f,1.0f);                 // Синій
              glVertex3f( 0.6f,-0.6f, 0.6f);              // Права точка

             glColor3f(1.0f,0.0f,0.0f);                // Червоний
              glVertex3f( 0.0f, 0.6f, 0.0f);              // Верх трикутника (Права)
              glColor3f(0.0f,0.0f,1.0f);                 // Синій
              glVertex3f( 0.6f,-0.6f, 0.6f);              // Ліво трикутника (Права)
              glColor3f(0.0f,1.0f,0.0f);                 // Зелена
              glVertex3f( 0.6f,-0.6f, -0.6f);          // Право треугольника (Права)

              glColor3f(1.0f,0.0f,0.0f);                 // Червоний
              glVertex3f( 0.0f, 0.6f, 0.0f);              // Низ трикутника (Ззаді)
              glColor3f(0.0f,1.0f,0.0f);                 // Зелений
              glVertex3f( 0.6f,-0.6f, -0.6f);              // Ліво трикутника (Ззаді)
              glColor3f(0.0f,0.0f,1.0f);                 // Синій
              glVertex3f(-0.6f,-0.6f, -0.6f);          // Право трикутника (Ззаді)

              glColor3f(1.0f,0.0f,0.0f);                 // Червоний
```

```
    glVertex3f( 0.0f, 0.6f, 0.0f);              // Верх трикутника (Ліво)
    glColor3f(0.0f,0.0f,1.0f);                  // Синій
    glVertex3f(-0.6f,-0.6f,-0.6f);              // Ліво трикутника (Ліво)
    glColor3f(0.0f,1.0f,0.0f);                  // Зелений
    glVertex3f(-0.6f,-0.6f, 0.6f);              // Право трикутника (Ліво)

  glEnd();
  glFlush (); }
```

Продемонструвати обертання піраміди зі зміною кута та координат. З масштабуванням:



Без масштабування, інший поворот та кут:



Аналогічно можна намалювати, змістити та повернути куб (за тим же самим посиланням можна подивитися як малювати зафарбований куб).

Крім функцій повороту та переміщення є ще і функція масштабування. Функція void **glScale[f d]**(GLtype x, GLtype y, GLtype z) виконує масштабування об’єкта (стиснення або розтяг), домножуючи відповідні координати його вершин на значення своїх параметрів. Ці перетворення будуть застосовуватися до примітивів, опис яких знаходиться нижче у програмі.

Створення 3D-об’єктів: побудова, поворот куба та сфери без заливки

Для того, щоб намалювати **куб** (не зафарбований, просто лініями, каркас) потрібно використати функцію **glutWireCube** (розмір сторони), але щоб побачити, що це куб – його потрібно повернути на декілька градусів:

Приклад 2.

```
glClear(GL_COLOR_BUFFER_BIT);          // Очищається буфер кадра
//glScalef(0.5,0.5,0.5);
glRotatef(20.0f, 1.0f, 1.0f, 1.0f);  // Поворот на 30 градусів навкого вектора (1, 1, 1)
glColor3f(0.7f, 0.25f, 0.55f);          // Задається біжучий колір примітивів
glutWireCube(1.0f);
glFlush ();
```

Для того, щоб після повороту не порушувати систему координат, бо вона повертається теж, можна скористатися такими командами:

glPushMatrix(); – запам’ятати локальну систему координат;

glPopMatrix(); – відновити локальну систему координат.

Тобто, на прикладі нижче можна побачити, що після повороту новий куб намалювався там само.

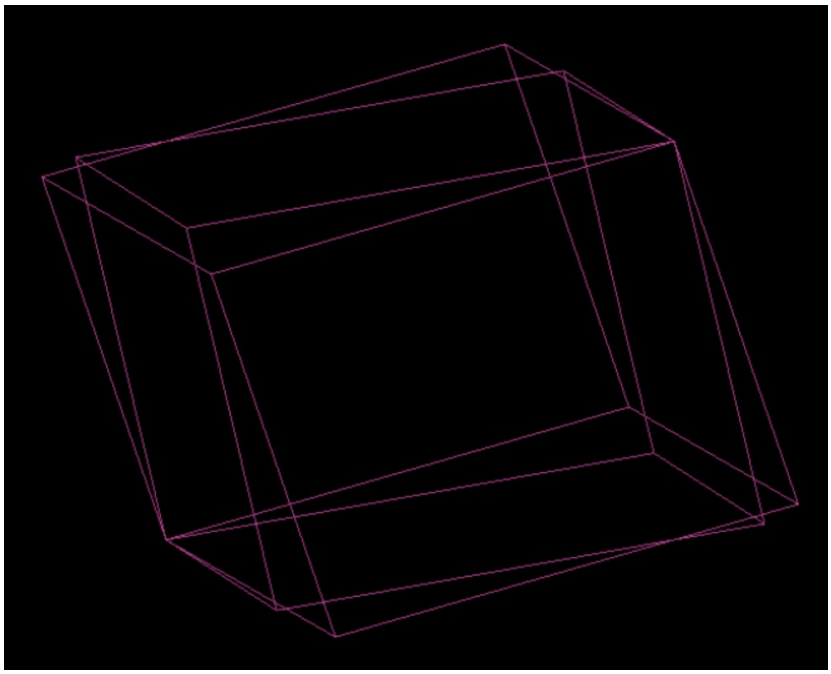
Приклад 3.

```
glClear(GL_COLOR_BUFFER_BIT);          // Очищається буфер кадра
```

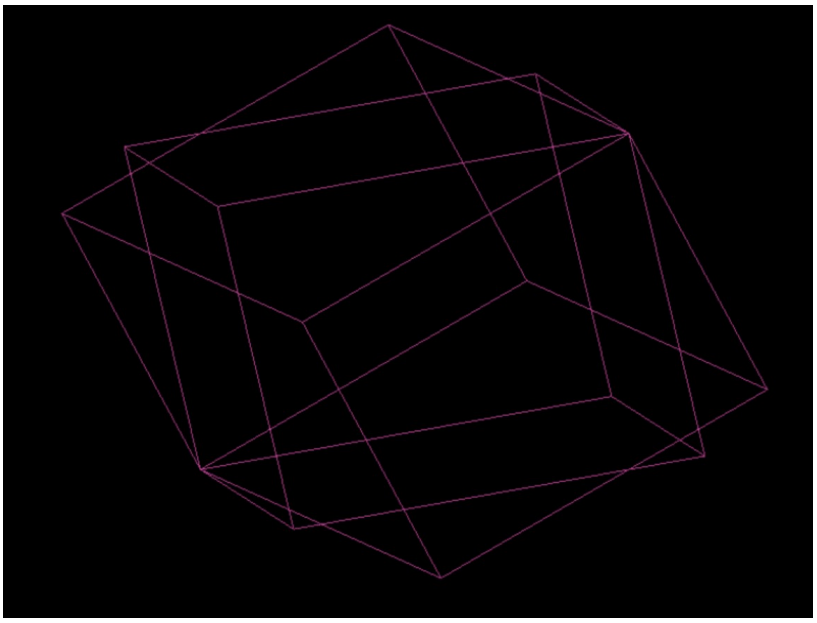
```
glPushMatrix();
glRotatef(20.0f, 1.0f, 1.0f, 1.0f);  // Поворот на 30 градусів навколо вектора (1, 1, 1)
glColor3f(0.7f, 0.25f, 0.55f);        // Задається біжучий колір примітивів
glutWireCube(1.0f);
glPopMatrix();
//glScalef(0.5,0.5,0.5);
 glRotatef(20.0f, 1.0f, 1.0f, 1.0f);        // Поворот на 30 градусів навколо вектора (1, 1, 1)
 glColor3f(0.7f, 0.25f, 0.55f);         // Задається біжучий колір примітивів
 glutWireCube(1.0f);
 glFlush ();
```

ЗВЕРНІТЬ УВАГУ!!!

Якщо змінити градус повороту наступного куба та використати команди збереження системи координат, то будемо мати:



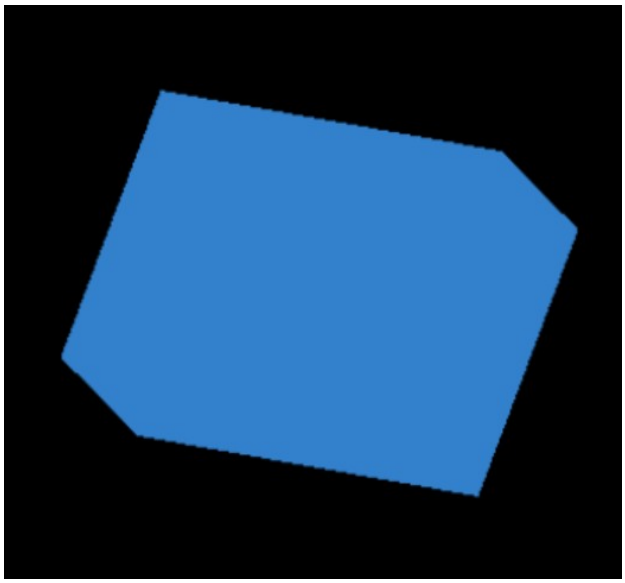
Без збереження координат:



Або, void **glutSolidCube**(GLdouble size) – зафарбований куб.

Приклад 4.

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.2f, 0.5f, 0.8f);
glRotatef(-25.0f, 1.0f, 1.0f, 1.0f);
glutSolidCube(0.5);
glFlush ();
```



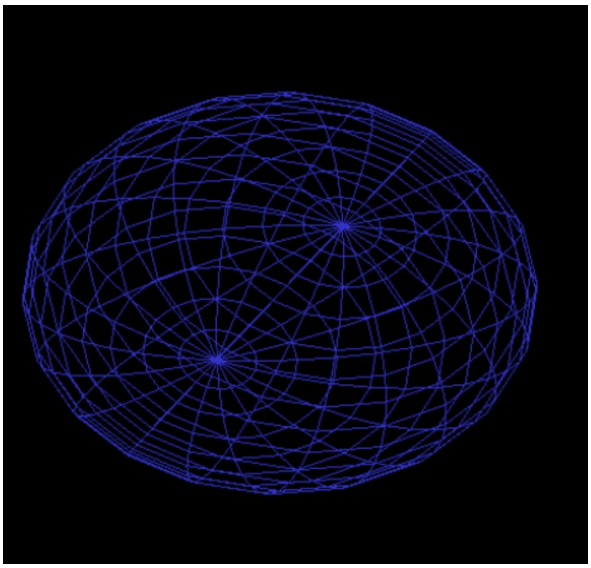
Крім розглянутих стандартних примітивів у бібліотеках GLU та GLUT описані більш складні фігури, такі як сфера, циліндр, диск (у GLU) і сфера, конус, тор, тетраедр, додекаедр, ікосаедр, октаедр та чайник (у GLUT).

Щоб побудувати каркас сфери, потрібно використати команду: void **glutWireSphere**(GLdouble radius, GLint slices, GLint stacks), вказавши радіус та кількість поділів сфери на відрізки по горизонталі та вертикалі. Щоб побудувати суцільну сферу: void **glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks).

Обов’язково повернути на певний кут та навколо якоїсь осі, щоб був вигляд збоку.

Приклад 5.

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.2f, 0.2f, 0.8f);
glRotatef(30.0f, -1.0f, 1.0f, 1.0f);
glutWireSphere(0.5, 20, 20);
glFlush ();
```



Аналогічно можна побудувати інші об’ємні примітиви (детальніше дослідити самостійно при виконанні лабораторної роботи).

Створення 3D-об’єктів з використанням масивів (приклад куба)

Детально приклад розглянутий за посиланням [28].

Розглядаємо кількість вершин, граней, та як вони відображаються на координатах.

Питання для самоконтролю

1. Які особливості коду ви відмітили за посиланням [27]?
2. Розгляньте та прокоментуйте приклад 1 та результат виконання програми.
3. Як відбувається побудова, зміщення та поворот піраміди?
4. Які особливості повороту, зміщення та повороту куба?
5. Яка функція дозволяє масштабування?
6. Прокоментуйте вхідні аргументи функції масштабування? Як вона працює?
7. Яка функція будує каркас кубу?
8. Розгляньте та прокоментуйте приклад 2 та результат виконання програми.
9. Які команди потрібно використати аби не порушити систему координат?
10. Розгляньте та прокоментуйте приклад 3 та результат виконання програми.
11. Яка функція будує зафарбований куб?
12. Розгляньте та прокоментуйте приклад 4 та результат виконання програми.
13. Які інші 3D-об’єкти можна будувати?
14. Розгляньте та прокоментуйте приклад 5 та результат виконання програми.
15. Як по-іншому описана побудова 3D-об’єктів у джерелі [28]?

ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Робота з класами у C++. Дружні функції та класи

1. Побудувати клас КАЛЕНДАР. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
2. Створити клас КООРДИНАТИ ТОЧКИ. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
3. Створити клас ДІЙСНЕ ЧИСЛО (у закритій частині класу знаходиться значення дійсного типу). Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
4. Створити клас ДРІБ – раціональних чисел, що є відношенням двох цілих чисел. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
5. Створити клас РЯДОК СИМВОЛІВ. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
6. Створити клас ДІЙСНІ ЧИСЛА – у закритій частині класу знаходяться значення дійсного типу. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
7. Створити клас ЦІЛІ ЧИСЛА – у закритій частині класу знаходяться значення цілого типу. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.

8. Визначити клас ЗАПИСНИК, який містить прізвище, ім'я, номер телефону, день народження. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
9. Визначити клас СКЛАД з назвами товарів, їх кількістю та ціною. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
10. Визначити клас ВЕКТОР. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
11. Визначити клас КОМПЛЕКСНЕ ЧИСЛО. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.
12. Створити клас ЧЕРГА. Визначити необхідні методи, конструктори, деструктори. Організувати реалізацію дружньої функції, дружнього методу, дружнього класу.

Лабораторна робота № 2
Шаблони функцій, шаблони класу

1. Використовуючи шаблони функцій, запрограмувати функції знаходження мінімального та максимального елементів для
   а) одновимірного масиву;
   б) двовимірного масиву.
   Шаблони функцій повинні правильно працювати для різних типів даних: int, double, char.

2. Використовуючи шаблони функцій, запрограмувати функції сортування (мінімум 2 методи) елементів

а) одновимірного масиву;

б) двовимірного масиву.

Шаблони функцій повинні правильно працювати для різних типів даних: int, double, char.

3. Реалізувати шаблон класу на прикладі стеку (лекційний приклад) –

   а) використати для шаблону структуру дерево;

   б) використати для шаблону структуру список;

   або будь яку іншу цікаву структуру

   Шаблон повинен працювати для різних типів даних int, double та ін.

   Завдання брати наприклад, 1 а, 2 б, 3 а, або 1 б, 2 а, 3 б.

Лабораторна робота № 3
Узагальнений клас та функція

1. Використовуючи лекційні приклади реалізувати та продемонструвати:

   а) узагальнену функцію або шаблон з одним аргументом: аргумент змінити певним чином – збільшити, зменшити та вивести на екран; функція повинна працювати для багатьох типів даних;

   б) узагальнену функцію з локальними змінними узагальненого типу: приклад для демонстрації придумати самостійно; функція повинна працювати для багатьох типів даних;

   в) використання узагальненої функції з 2-ма аргументами: приклад для демонстрації придумати самостійно; функція повинна працювати для багатьох типів даних.

2. Використовуючи лекційні приклади реалізувати та продемонструвати:

а) узагальнений клас зі службовим словом typename: приклад для демонстрації придумати самостійно; клас повинен працювати для різних типів даних;

б) узагальнений клас з використанням узагальненого типу клас: приклад для демонстрації придумати самостійно; клас повинен працювати для різних типів даних;

в) узагальнений клас, що використовує 2 узагальнені типу, один з яких клас: приклад для демонстрації придумати самостійно; клас повинен працювати для різних типів даних.

Лабораторна робота № 4

Налаштування графічного режиму у середовищі програмування с++ та побудова примітивів

1. Згідно лекційного матеріалу та інструкції налаштувати на своїх ПК графічний режим у консольному середовищі програмування с++. Звернути увагу на те, що після налаштування графіки мають правильно працювати графічні приклади з лекції. (Для доказу про налаштування або показати скріни екрану, що працює та налаштовано або принести та продемонструвати).

2. Використовуючи основні примітиви (коло, квадрат, прямокутник і т.д.) намалювати простий рисунок на власну фантазію (без зафарбовування).

3. Використовуючи основні примітиви проявити творчу фантазію і побудувати емблему нашого факультету, якою бачите її ви (без зафарбовування, тільки цікаві, правильні та складні контури).

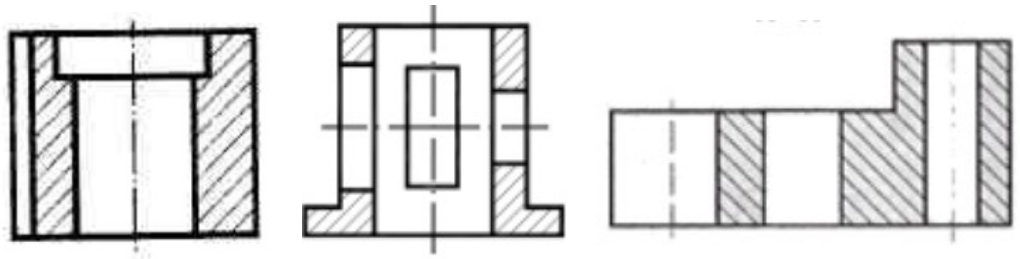
Лабораторна робота № 5

Побудова графічних зображень використовуючи функції бібліотеки graphic.h

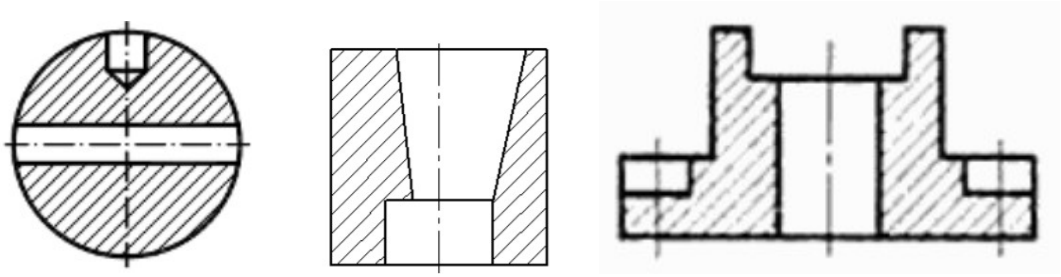
1. Використовуючи приклад 1 лекції № 5 вивести список студентів (для кожного прізвища інший колір, інший шрифт) вашої підгрупи звичайним способом (по горизонталі), по вертикалі, зі зміщенням – кожне наступне прізвище та ім’я буде зміщене від попереднього на 15 пікселів.

2. Використовуючи приклад 2, 4 лекції № 5 запрограмувати створення простенького технічного рисунка, до прикладу як на картинках. Використати: лінії різного стилю та заливку різного стилю (малюнок вибираємо по списку)

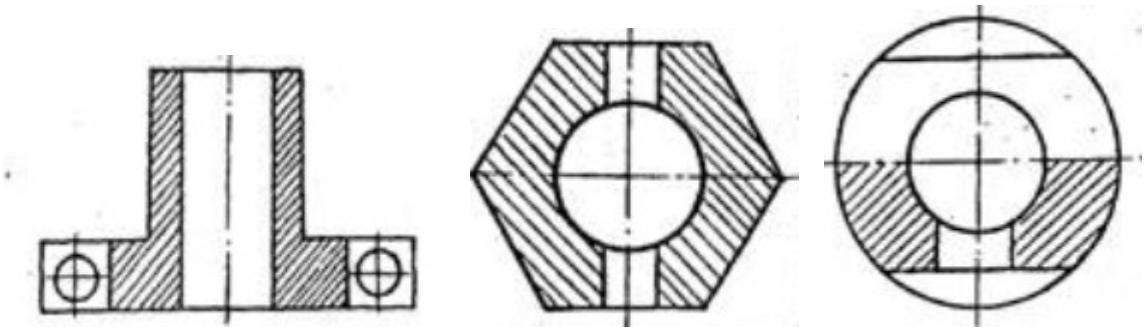
1) 2) 3)



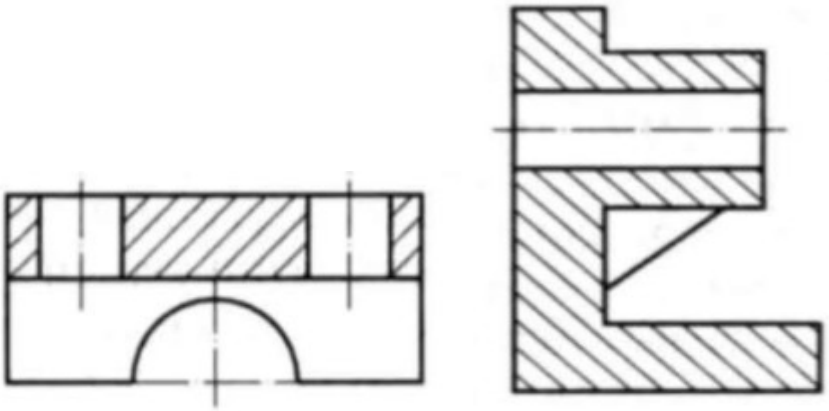
4) 5) 6)



7) 8) 9)

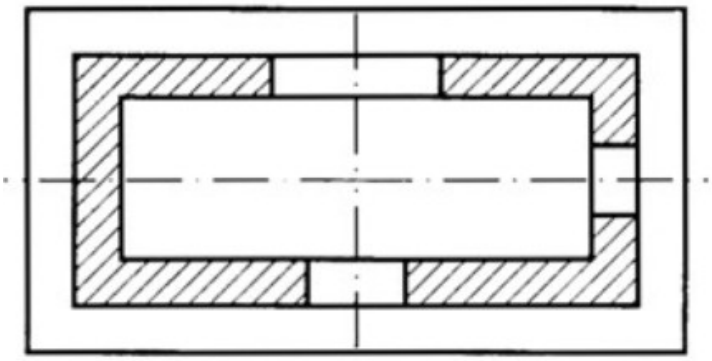


10)



11)

12)



3. Використовуючи ідею прикладу 6 лекції № 5 виконати імітацію побудови простого об’ємного елемента. Запрограмувати функції: побудови елемента, переміщення (копіювання предмету), видалення предмету.

Лабораторна робота № 6

Побудова графічних зображень: простий та складний рух

1. Використовуючи приклад 7 лекції № 5, запрограмувати рух будь якого одного примітивного зображення (крім поданого у лекції) по колу, по горизонталі та по вертикалі.

2. Запрограмувати рух будь яких примітивних фігур: щоб одночасно рухалися з багатьох точок екрану різними кольорами.

3. Використовуючи приклад 5 лекції № 5, запрограмувати:
   - завершити побудову шахматної доски (методом переміщення та копіювання однакових частин);
   - підписати латинськими літерами та цифрами;
   - правильний (за правилами гри) рух будь якої шахматної фігури.

Лабораторна робота № 7

Побудова графіка функції та складнішого руху об’єкта

1. Використовуючи графічні функції та приклад 1 з лекції № 6, побудувати графік функції (вибрати функцію, так щоб змогли гарно масштабувати, але звіритеся, щоб однакових не було!). На екран мають бути виведені підписані осі, сам графік та проміжок на якому будуємо. Знизу справа написано графік якої функції зображено та прізвище хто виконав (при виведенні тексту використати якийсь цікавий стиль).

2. Використовуючи приклад 3 та рекомендовані відеоролики за посиланням з лекції № 6, враховуючи техніку побудови рухомих зображень, реалізувати та вивести на екран будь яке складніше рухоме зображення (наприклад, рухається повітряна куля і т.д.).

Лабораторна робота № 8

Побудова примітивів з використанням графіки OpenGL та задання кольорів

1. Використовуючи інструкцію налаштування графіки OpenGL з лекції № 7 продемонструвати заданий там приклад з обертом червоних фігур.

2. Використовуючи приклад з лекції № 7 та особливості побудови примітивів з лекції № 8 нарисувати УСІ примітиви (крім показаних у лекції трикутника та квадрата) з рисунку лекції № 8 (примітиви нарисувати по замовчуванню білим кольором).

3. Використовуючи приклади лекцій № 7, 8 продемонструвати побудову УСІХ (крім показаних у лекції трикутника та квадрата) примітивів різними кольорами заливки: суцільний колір, кожна вершина має свій колір, градієнт – перехід від одного до іншого.

Лабораторна робота № 9

Переміщення та повороти примітивів з використанням графіки OpenGL

1. Використовуючи приклади лекції № 8 продемонструвати переміщення УСІХ примітивів (крім показаних у лекції трикутника та квадрата) у різні області графічного екрану.

2. Продемонструвати симетричне виведення на графічний екран цікавих примітивів.

3. Продемонструвати повороти УСІХ (крім показаних у лекції трикутника та квадрата) примітивів навколо 3-х осей з заданням різного кута повороту.

4. Використовуючи примітиви побудувати простий змістовний рисунок (хатинку і т.д.).

Лабораторна робота № 10

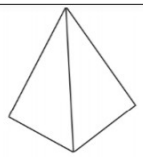
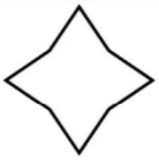
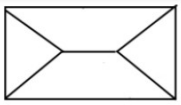
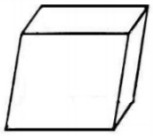
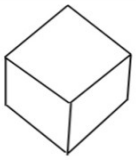
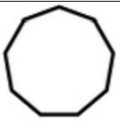
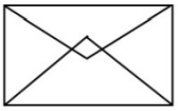
Побудова кола та дослідження атрибутів примітивів з використанням графіки OpenGL

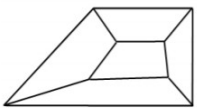
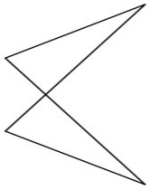
1. Дослідити самостійно як будувати коло з використанням бібліотеки OpenGL. Продемонструвати побудову кола різними кольорами та у різному місці графічного екрану. (Спробуйте за цим посиланням [29]).

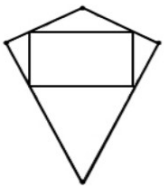
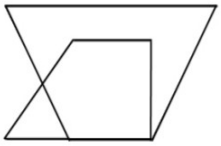
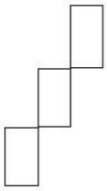
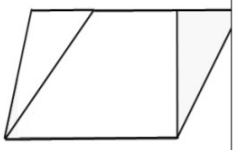
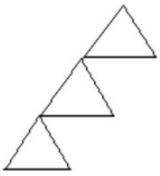
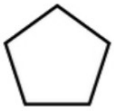
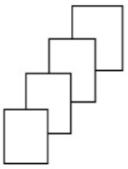
2. Використовуючи атрибути (лекція № 8) примітивів, дослідити та продемонструвати:
   - точки: розмір точок, режим відображення зі згладжуванням та без згладжування;
   - лінії: ширина лінії, штрихування, режим відображення зі згладжуванням та без згладжування;
   - трикутники, чотирикутники, багатокутники: режим відображення зі згладжуванням та без згладжування, режим малювання (glPolygonMode), штриховка.

3. Створити зображення за допомогою примітивів згідно варіанту за таблицею 2.

Таблиця 2.

1.		9.		17.	
2.		10.		18.	
3.		11.		19.	

4.		12.		20.	
5.		13.		21.	

6.		14.		22.	
7.		15.		23.	
8.		16.		24.	

ДОДАТКОВІ ЛАБОРАТОРНІ РОБОТИ ДО ТЕМ 9-14

Лабораторна робота № 11

Повторення: вкладені цикли. Динамічні масиви

1. Завдання: використовуючи динамічні матриці реалізувати знаходження суми та добутку 2-х будь яких матриць. Обов'язково використати функцію для підрахунку часу виконання програми та вивести час на екран для різної розмірності масивів, зробити висновки (Підготувати приклади на яких можна перевірити правильність виконання).
2. Перевірка ДЗ (по лекції сума сум, добуток добутків, сума добутків, добуток сум). (Підготувати приклади на яких можна перевірити правильність виконання).
3. Вибрати завдання від 1 до 16, у зворотньому порядку: 1-ший бере 16-те, 2-гий - 15-те і т.д.:
    1. Надрукувати всі числа з інтервалу [100, 200], цифровий корінь яких кратний 3 (3, 6, 9).
    2. Дано число ***a***. Зайдіть ***a*** перших простих чисел.
    3. Дано число ***a***. Знайти просте число, що більше ***a***.
    4. Дано число ***a***. Знайдіть 5 простих чисел, більших ***a***.
    5. Дано число ***a***. Знайти найближче до нього просте число.
    6. Знайдіть всі числа з інтервалу (100, 200), цифровий корінь яких є простим числом (1, 2, 3, 5, 7).
    7. Знайдіть всі числа з інтервалу (100, 200), які кратні своєму цифровому кореню.
    8. Знайдіть цифрові корені чисел Фібоначчі, що належать інтервалу (100, 1000).
    9. Обчисліть та надрукувати цифрові корені досконалих чисел, що належать діапазону (1; 10000). Натуральне число називається

досконалим, якщо воно дорівнює сумі своїх дільників, із урахуванням 1 і за винятком самого числа. Наприклад, число 6 - досконале (6=1+2+3).

10. Знайдіть всі симетричні числа з інтервалу [10000, 1000000].
11. Знайдіть всі симетричні паліндроми з інтервалу [1000000, 1000000000]. Пояснення: *паліндром* – це число, яке читається однаково справа наліво та зліва направо, тобто саме число дорівнює перевернутому числу.
12. Надрукувати з чисел Фібоначчі в інтервалі від 1 до 100, тільки прості числа, а також їх порядкові номери в ряду Фібоначчі.
13. Для всіх чисел, що належать діапазону [100; 120] обчислити та надрукувати ті дільники, що є членами послідовності Фібоначчі.
14. Надрукувати всі чотирьохзначні симетричні числа та знайдіти їх кількість.
15. Знайдіть цифрові корені всіх симетричних чисел, що належать інтервалу (10000, 100000).
16. Надрукувати перші 10 п’ятизначних паліндромів, що є простими числами. Пояснення: *паліндром* – це число, яке читається однаково справа наліво та зліва направо, тобто саме число дорівнює перевернутому числу. [32]

Лабораторна робота № 12

Програмування математичних функцій за допомогою рядів ч.1.

Завдання: аналогічно до прикладу теми № 10 програмування наближеного обчислення значення sin(x). Запрограмувати обчислення наступних функцій (аргумент x вводиться користувачем з клавіатури, а границя ряду визначається як константа):

1. $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{i=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$
2. $\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$
3. $1 + 2x + 3x^2 + 4x^3 + \ldots = \sum_{n=0}^{\infty} (n+1)x^n = \frac{1}{(1-x)^2}$, $|x|<1$, ( )
4. $1 - 2x + 3x^2 - 4x^3 + \cdots = \sum_{n=0}^{\infty} (-1)^n (n+1) x^n = \frac{1}{(1+x)^2}$, $(|x|<1)$
5. $\ln(1+x) = x - \frac{x^3}{3} + \frac{x^5}{5} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)}$, $|x|<1$, ( )
6. $\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n+1)!}$

Всі завдання обов’язково, перед тим як здавати, мають бути розписані і в коді програми перевірені аналогічними вбудованими функціями, або обмеженнями вказаними у завданні.

Варіанти вибирати наступним чином: 1, 3, 5 або 2, 4, 6 або 2, 3, 6 або 1, 4, 5 і т. д.

Лабораторна робота № 13

Програмування математичних функцій за допомогою рядів ч.2.

Завдання: аналогічно до прикладу ряд для експоненти теми № 10 наближеного обчислення значення exp(x). Запрограмувати обчислення наступних рядів, використовуючи клас та динамічний масив:

1. $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$.

2. $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}.$

3. $ch(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}.$

4. $sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}.$

5. $1 + 2x + 3x^2 + 4x^3 + \ldots = \sum_{n=0}^{\infty} (n+1) x^n = \frac{1}{(1-x)^2}.$

6. $1 - 2x + 3x^2 - 4x^3 + \ldots = \sum_{n=0}^{\infty} (-1)^n (n+1) x^n = \frac{1}{(1+x)^2}.$

7. $1 + 2^2 x + 3^2 x^2 + 4^2 x^3 + \ldots = \sum_{n=1}^{\infty} n^2 x^{n-1} = \frac{1+x}{(1-x)^3}.$

8. $1 + 3x^2 + \frac{5x^4}{2!} + \frac{7x^6}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{(2n+1) x^{2n}}{n!} = (1 + 2x^2) \exp(x^2).$

Всі завдання обов’язково, перед тим як здавати, мають бути розписані і в коді програми перевірені аналогічними вбудованими функціями або їх обмеженнями, які вказані в завданнях.

Варіанти вибирати наступним чином: 1-ший по списку – 1, 3, 5, 7; 2-гий по списку – 2, 4, 6, 8; 3-тій по списку – 1, 4, 5, 8; 4-тий по списку – 2, 5, 7, 8; 5-тий по списку – 1, 5, 6, 8; і так далі …

Лабораторна робота № 14

Шаблони (продовження) та перевантаження операторів

Використовуючи приклади теми № 11 виконати завдання:

1. Написати програму з узагальненим класом, у якого є поле-масив. Перевантажити оператор додавання так, щоб при додаванні до об’єкту класу цілого числа виконувався циклічний зсув елементів масиву на відповідну

кількість позицій (вправо для додатного цілочислового операнда і вліво для від’ємного).

2. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Перевантажити оператор [] так, щоб індексувати об’єкти узагальненого класу.

3. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Перевантажити оператор інкремента так, щоб всі рядки в масиві циклічно переміщалися на одну позицію.

4. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Перевантажити оператор інкремента так, щоб всі стовпці в масиві циклічно переміщалися на одну позицію.

5. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Описати метод, з допомогою якого міняються місцями два рядки масиву. Індекси рядків масиву передаються аргументами методу.

6. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Описати метод, з допомогою якого міняються місцями два стовпці масиву. Індекси рядків масиву передаються аргументами методу.

7. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Описати метод, з допомогою якого виконується пошук елемента в масиві. В якості результату повертається загальна кількість співпадінь (кількість входжень елемента в масив).

8. Написати програму з узагальненим класом, у якого є поле – двовимірний масив. Описати метод, з допомогою якого виконується порядковий пошук елемента в масиві. В якості результату виводяться індекси елементів, що співпадають з елементом пошуку.

Завдання вибирати наступним чином: 1, 3, 5, 7 або 2, 4, 6, 8 або 1, 4, 5, 8 або 2, 3, 6, 7 або 1, 3, 6, 8.

Лабораторна робота № 14

Файли

У даній лабораторній роботі запрограмувати меню (використовуючи команду вибору (switch…case)), яке дозволяє працювати з файлами: створення (запис диних у файл) файлу, читання файлу (виведення вмісту файлу на екран), пошук за заданими умовами (виведенн на екран результату), завершення роботи з файлом.

У завданнях з типізованими файлами (коли дані різного типу), організовуєте дані у вигляді класу з відповідними полями, об'єкт записуєте у файл та читаєте з файлу. Проміжних структур не використовувати.

Варіанти завдань:

1. Взяти завдання по списку у журналі починаючи з кінця, тобто 1-ший бере останнє і т.д., 2-гий бере передостаннє (стор. 142-146) [3]
2. Взяти завдання по списку у журналі починаючи з початку, тобто 1-ший бере 1-ше і т.д. (стор. 149-152) [3]
3. Взяти завдання по списку, але рахунок починати з кінця, тобто 1-ший бере 25-те, 2-гий бере 24-те і т.д. (стор. 166-167) [3]

Можна і треба дивитися як програмувалися аналогічні завдання у джерелі [3] вище від варіантів завдань і обов’язково зверніть увагу на результат виведення у прикладі.

Лабораторна робота № 15

Створення 3D-об’єктів вручну (заданням координат вершин), використовуючи вбудовані функції та з використанням масивів

Запрограмувати побудову:

1. **Куба:**

   а) аналогічно як будували піраміду, прорисовуючи кожну точку і задаючи колір або кожній точці, або цілій стороні, користуючись посиланням з лекції, або іншим зрозумілим для вас джерелом;

б) задаючи масив координат для вершин, кольорів та порядку індексів (користуючись джерелом пункту 3 теми № 14);

в) використовуючи каркас куба побудувати цегляну стіну, нахилену під певним градусом, щоб видно було, що вона складається з кубів; аналогічно рисунку, щоб при поворотах було видно об’єм або задати повороти мишкою чи клавіатурою.

2. **Призму (для полегшення завдання можна будувати трикутну):** аналогічно кубу з завдання 1., або прорисовуючи кожну точку, або задаючи масиви даних.

   Побудову призми по частинах (вершинах) можна подивитися у джерелі (перший код) [30] .

3. Дослідити та побудувати **інші об’ємні фігури** (з використанням вбудованих функцій), які дають можливість будувати бібліотеки GLU та GLUT, окрім сфери та куба (продемонстровано на лекції): циліндр, диск, куля (у GLU), конус, тор, тетраедр, додекаедр, ікосаедр, октаедр та чайник (у GLUT) та інші.

   Можна скористатися джерелами [30], [31].

Питання до модульного контролю № 1

Теми: “Дружні функції та класи”, “Шаблони функцій та шаблони класів”

(Орієнтовні питання)

1. Що таке функція, її особливості?
2. Які функції називаються дружніми?
3. Як оголосити функцію дружньою? Навести приклад.
4. До яких елементів класу має доступ дружня функція?
5. Знайдіть помилку в коді і виправіть:

```
class MyClass {
   double x;
public:
   MyClass (double z)
  {      x=z;    }
 // Дружня функія
   friend show (MyClass obj);
};
```

6. Знайдіть помилку в коді і виправіть:

```
class MyClass {
   double x;
public:
   MyClass (double z)
  {      x=z;    }
 // Дружня функія
   show (MyClass obj);
};
```

7. Який випадок показує доцільність застосування дружніх функцій?

8. Щоб функція мала доступ до полів обох класів, в кожному класі ця функція повинна бути оголошена як …
9. В якому полі класу може бути розміщена дружня функція?
10. Функція може використовуватися, як дружня до СКІЛЬКОХ класів?
11. Як викликаються дружні функції?
12. Як оголосити клас дружнім?
13. Що таке дружній клас?
14. Коли клас оголошено дружнім, всі його методи також …
15. Що таке дружній метод?
16. Як він оголошується?
17. Для чого використовується дружній метод?
18. Що таке шаблони функцій?
19. Для чого і коли використовують шаблони функцій?
20. Синтаксис створення шаблону функції: …
21. Яка між записами різниця?

    Template <class T>

    Або

    Template <typename T>

22. Коли використовується шаблон функції як працює компілятор з пам’яттю?
23. Для чого і коли створюється дружній клас?
24. Особливості написання коду для того, щоб такий запис не видавав помилку?

```
int main()
{
   Stak <int> s1;
    s1.push(11);
    s1.push(22);

   Stak <float> s2;
```

```
s2.push(11.45);
s2.push(22.89);
```

Питання до модульного контролю № 2

Тема: “Програмування графіки у С++”

(Орієнтовні питання)

1. Які особливості налаштування графіки (двовимірної) в с++?
2. Які функції дозволяють малювати основні примітиви? (коло, арка, прямокутник, лінія, точка і.т.д)
3. З допомогою якої функції зафарбувати примітив одним кольором або задати штрихування?
4. Які функції дозволяють виводити у графічному режимі слова? Які додаткові параметри необхідно вказати?
5. Описати технологію побудови графіка функції.
6. Описати технологію створення рухомого зображення.
7. Практичне: зобразити побудову прямокутника зафарбованого будь якою текстурою.

   (І-в прямокутник в правому верхньому куті екрану, синього кольору зовнішня лінія,

   ІІ-в прямокутник в правому нижньому куті екрану, червоного кольору зовнішня лінія,

   ІІІ-в прямокутник по центрі екрану, фіолетового кольору зовнішня лінія)
8. Що таке OpenGl?
9. Вкажіть характерні особливості OpenGl?
10. На які 5 категорій розділені функції OpenGl?
11. Які бібліотеки входять в OpenGl?
12. Архітектура OpenGl?

13. Графічно зобразити функціонування конвеєру OpenGl.

14. Які бібліотеки OpenGl потрібно інсталювати та підключати окремо, а які бібліотеки можуть працювати зразу?

Практичне завдання № 1

А) Використовуючи C++ та можливості графіки (OpenGL та graphics.h або інші графічні бібліотеки) максимально близько запрограмувати розгортку автомобіля (вигляд зверху).

Б) Описати та показати по основних кроках створення графічного об’єкту: крок 1 (декілька речень + програмний код) – скрін 1, що будується; крок 2 (декілька речень + програмний код) – скрін 2 і т.д.

Практичне завдання № 2.

Використовуючи C++ та можливості графіки OpenGL (дослідити роботу з клавіатурою) запрограмувати:

А) графічний екран у вигляді сітки;

Б) поставити маркер певного кольору в певну точку;

В) зробити можливість перемістити маркер клавіатурою по сітці (стрілка вниз, вверх, вліво, вправо);

Г) зробити можливість малювати лінію по сітці тим кольором який стоїть на маркері і в ту сторону, яку ми хочемо (стрілка вниз, вверх, вліво, вправо).

Орієнтовні завдання на залік

І-частина (робота з графікою)

1. Запрограмувати: вивести на графічному екрані текст «23.12.2020 – ЗАЛІК» по горизонталі та по вертикалі, по центру, у верхньому правому куті екрану.
2. Запрограмувати функцію, яка малює коло та зафарбовує в центрі екрану з радіусом 35, ободок кола – зелений.
3. Запрограмувати функцію, яка малює коло та зафарбовує у верхньому правому куті екрану з радіусом 25, ободок кола – червоний.
4. Запрограмувати функцію, яка малює коло та зафарбовує у верхньому лівому куті екрану з радіусом 30, ободок кола – жовтий.
5. Запрограмувати функцію, яка малює квадрат та зафарбовує у нижньому лівому куті екрану зі стороною 30, ободок квадрату – жовтий.
6. Запрограмувати функцію, яка малює квадрат та зафарбовує у нижньому правому куті екрану з стороною 25, ободок квадрату – червоний.
7. Запрограмувати функцію, яка буде рухати коло з заданого розміщення кола по прямій горизонтально.
8. Запрограмувати функцію, яка буде рухати коло з заданого розміщення кола по прямій вертикально.
9. Запрограмувати функцію, яка буде рухати квадрат з заданого розміщення квадрату по прямій горизонтально.
10. Запрограмувати функцію, яка буде рухати квадрат з заданого розміщення квадрату по прямій вертикально.

ІІ-га частина (дружній клас)

1. Запрограмувати клас точка з 2-ма координатами (створення, видалення точки). До нього організувати дружній клас, який шукає векторний добуток 2-х точок заданих координатами, вивести результат.
2. Запрограмувати клас комплексне число (створення, видалення). До нього організувати дружній клас, який шукає добуток комплексних чисел, вивести результат.
3. Запрограмувати клас студент, що задане прізвищем та середнім балом (створення, видалення студента). До нього організувати дружній клас, який шукає студента з прізвищем на задану літеру або каже, що такого не має, вивести результат.
4. Запрограмувати клас студент, що задане прізвищем та середнім балом (створення, видалення студента). До нього організувати дружній клас, який шукає студентів з однаковим балом або каже, що таких не має, вивести результат.
5. Запрограмувати клас книга, що задане автором та назвою (створення, видалення книги). До нього організувати дружній клас, який шукає книгу за заданим автором або каже, що такого не має, вивести результат.
6. Запрограмувати клас авто, що задане номером та маркою (створення, видалення авто). До нього організувати дружній клас, який шукає авто за заданим номером або каже, що такого не має, вивести результат.
7. Запрограмувати клас дата народження, що задане числом, місяцем та роком (створення, видалення дати). До нього організувати дружній клас, який шукає дату за місяцем або каже, що такого не має, вивести результат.
8. Запрограмувати клас дата народження, що задане числом, місяцем та роком (створення, видалення дати). До нього організувати дружній клас, який визначає найстаршу особу, вивести результат.

9. Запрограмувати клас предмет-залік, що заданий назвою та балами (створення, видалення). До нього організувати дружній клас, який рахує кількість нездач заліку або каже, що здали всі, вивести результат.
10. Запрограмувати клас предмет-залік, що заданий назвою та датою (створення, видалення). До нього організувати дружній клас, який визначає чи є в одну дату 2 предмети або каже, що в ту дату предмету немає ніякого, вивести результат.

III-тя частина (шаблон функції)

1. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що переставляє місцями 1-й та 3-тій рядки матриці. Вивести результат.
2. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю, елементи якої впорядковані по спаданню, перемішати елементи. Вивести результат.
3. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що переставляє місцями 2-й та 4-тій стовпці матриці. Вивести результат.
4. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що переставляє місцями діагоналі матриці. Вивести результат.
5. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю у якої по головній діагоналі стоять однакові елементи, всі інші різні. Вивести результат.
6. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю у якої по головній діагоналі стоять різні елементи, всі інші елементи=0. Вивести результат.

7. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю у якої по другорядній діагоналі стоять однакові елементи, всі інші різні. Вивести результат.
8. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю у якої по другорядній діагоналі стоять різні елементи, всі інші елементи=1. Вивести результат.
9. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю у якої кількість стовпців=кількості рядків, поміняти місцями перший рядок на перший стовпець. Вивести результат.
10. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що будує двовимірну матрицю, елементи якої впорядковані по зростанню, перемішати елементи. Вивести результат.

Додаткові завдання:

1. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить середнє арифметичне від’ємних елементів масиву. Вивести результат.
2. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить суму додатних елементів масиву. Вивести результат.
3. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить добуток від’ємних елементів масиву. Вивести результат.

4. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить кількість додатних елементів масиву. Вивести результат.
5. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить середнє арифметичне додатних елементів масиву. Вивести результат.
6. Запрограмувати шаблон функції та продемонструвати роботу для різних типів даних: функція, що знаходить кількість від’ємних елементів масиву. Вивести результат.

Тестові завдання

Тести

1. Для того, щоб динамічно створити одновимірний масив, потрібно:
   Відповіді:
   1. **По-перше, оголосити у програмі вказівник на тип, що відповідає типу елементів цього масиву.**
   2. **По-друге, виділити пам'ять під масив.**
   3. По-перше, оголосити у програмі масив, що відповідає типу елементів цього масиву.
   4. По-друге, наповнити масив елементами.

2. Щоб виділити пам'ять під масив, використовують одну з функцій:
   1. Malloc.
   2. **Malloc або calloc.**
   3. Printf.
   4. Printf та scanf.

3. Чи потрібно вивільняти пам’ять після роботи з динамічним масивом?
   1. **Так**.
   2. Ні.
   3. Як мені потрібно буде.

4. Якщо потрібно вивільнити пам'ять після роботи з динамічним масивом, то це можна зробити з використанням функції:
   1. Print(a).
   2. **Free(a).**
   3. Disaut(a).
   4. Freem(a).

5. Пам'ять під багатовимірні динамічні масиви виділяється по …, починаючи з першого індексу.
   1. Стовпцях.
   2. **Рядках**.
   3. Блоках.
   4. Сегментах.

6. Вказівник на двовимірний цілочисловий масив може бути оголошений таким чином:
   1. **int **a.**
   2. int *a.
   3. int *«a».
   4. int ***a.

7. Як правильно виділити пам'ять для роботи з двовимірним динамічним масивом?
   1. **Виділити пам'ять для вказівників, потім для рядків.**
   2. Виділити пам'ять для рядків, потім для стовпців.
   3. Виділити пам'ять для рядків 2 рази.

8. Як правильно вивільнити пам’ять після роботи з двовимірним динамічним масивом?
   1. Вивільнити 2 рази для рядків.
   2. Вивільнити для рядків потім для стовпців.
   3. **Вивільнити пам'ять для рядків, потім для вказівників.**

9. Як розв’язуються задачі на вкладені цикли?
   1. Пишеться стільки циклів по порядку скільки потрібно.

2. **Спочатку складний алгоритм розбивається на простіші, тоді стає видно де треба цикл, а де не потрібно та скільки їх.**
3. Вкладаються цикли у порядку їх старшинства.

10. Дано натуральне число ***n***. Знайдіть його цифровий корінь (від слова цифра, не плутати з коренем квадратним!). Який цифровий корінь числа 3456?
    1. 6.
    2. 8.
    3. 3.
    4. **9.**

11. Дано натуральне число ***n***. Знайдіть його цифровий корінь (від слова цифра, не плутати з коренем квадратним!). Який цифровий корінь числа 197?
    1. 6.
    2. **8.**
    3. 3.
    4. 7.

12. Дано натуральне число ***n***. Знайдіть його цифровий корінь (від слова цифра, не плутати з коренем квадратним!). Який цифровий корінь числа 151?
    1. 6.
    2. 8.
    3. 3.
    4. **7.**

13. Напишіть перше симетричне 6-ти значне число:
    100100

14. **…** - це число, яке читається однаково справа наліво та зліва направо, тобто саме число дорівнює перевернутому числу.
    1. Симетричне.
    2. **Паліндром.**
    3. Симетричне парне.

15. При програмуванні задач на знаходження суми сум, початкові значення сум мають бути:
    1. Рівними 1.
    2. Одне рівне 1, друге - 0.
    3. **Обидва рівні 0**.
    4. Перше - 0, друге - 1.

16. При програмуванні задач на знаходження суми добутків, початкові значення мають бути:
    1. Рівними 1.
    2. Одне - 1, друге - 0.
    3. Обидва рівні 0.
    4. **Перша - 0, друге - 1.**

17. При програмуванні задач на знаходження добутку добутків, початкові значення мають бути:
    1. **Рівними 1.**
    2. Одне - 1, друге - 0.
    3. Обидва рівні 0.
    4. Перша - 0, друге - 1.

18. При програмуванні задач на знаходження добутку сум, початкові значення мають бути:

1. Рівними 1.
2. **Одне - 1, друге - 0.**
3. Обидва рівні 0.
4. Перша - 0, друге - 1.

19. При програмуванні задач на знаходження суми сум і т.д. використовують:
    1. Розгалуження.
    2. Вкладені цикли.
    3. Розгалуження в циклі.
    4. Лінійний запис коду.
    5. Цикл в розгалуженні.
    6. **Вкладені цикли та лінійний запис коду.**

20. При програмуванні задач на знаходження суми сум, добутку сум і т.д. для перевірки правильності проміжних результатів краще використати:
    1. Калькулятор онлайн.
    2. **Таблицю розрахунків.**
    3. Схему розрахунків.
    4. Розрахунок вручну.

Лекція 2. Програмування математичних функцій за допомогою рядів

21. Для того, щоб запрограмувати обчислення будь якої математичної функції на С++, ми підключаємо бібліотеку:
    1. **math.h.**
    2. funkmath.h.
    3. iosmath.h.
    4. mathc.h.

22. Який спосіб програмування математичних функцій можна використати, для того, щоб продемонструвати правильність розрахунків функцій бібліотеки math.h?
    1. Програмування з використанням матриць.
    2. **Програмування з використання рядів.**
    3. Програмування з використанням класів.
    4. Програмування з використанням структур.

23. На практиці при обчисленні значення функції sin(x), cos(x) та ін. відповідний ряд …
    1. Тільки обмежують.
    2. **Обмежують або шукають з заданою точністю.**
    3. Шукають тільки з заданою точністю.
    4. Вибирають максимальне число та з ним порівнюють.

24. Якщо при програмуванні функцій є вираз, що містить піднесення до степені, як діємо?
    1. Використовуємо у програмі функцію піднесення до степені.
    2. Використовуємо рекурсію.
    3. **Використовуємо спрощення, щоб позбутися піднесення до степені.**
    4. Використовуємо для обчислення цикл.

25. Якщо при програмуванні функцій є вираз, що містить факторіал, як діємо?
    1. Використовуємо у програмі функцію обчислення факторіалу.
    2. Використовуємо рекурсію.
    3. **Використовуємо спрощення, щоб позбутися факторіалу.**
    4. Використовуємо для обчислення цикл.

26. До яких дій максимально зводять процеси обчислення при програмуванні?
    1. До циклів, розгалужень.
    2. **До віднімання, додавання, множення та ділення.**
    3. До використання функцій.
    4. До використання об’єктів.

27. Для того, щоб перевірити чи правильно рахується наближене значення функції за допомогою програми потрібно:
    1. **Використати вбудовану відповідну функцію бібліотеки math.h.**
    2. Порахувати вручну.
    3. Скласти таблицю обрахунків та використати бібліотеку iostream.

28. Програмування рядів використовує ідею:
    1. Рекурсії.
    2. Вкладених циклів.
    3. **Рекурентних елементів.**
    4. Рекурентних рівнянь.

29. Програмувати ряди можна з використанням …
    1. Структур.
    2. Файлів.
    3. Рядків та символів.
    4. **Класу та динамічного масиву.**

30. Ряд для функції – це …
    1. Нескінченне диф. рівняння.
    2. **Нескінченна сума.**
    3. Скінченна сума.
    4. Скінченний добуток.

31. Що можна оголосити шаблоном?

1. Структуру.
2. Файл.
3. **Клас**.
4. Об’єкт.
5. **Функцію.**

32. Яке службове слово треба використати щоб показати, що використовуємо шаблон?

1. Shablon.
2. **Template.**
3. TemplateS.
4. Shab<T>.

33. Що відбувається A operator+(A obj)?

1. Додається оператор та функція.
2. Додається оператор та об’єкт.
3. **Перевантажується оператор +.**
4. Перевантажується оператор +(.
5. Перевантажується оператор +().

34. Чи можна так додавати: a1.res.k = 1; a2.res.k = 2; a3 = a1 + a2?

1. Не можна.
2. Можна, але треба тільки оголосити об’єкти.
3. Можна, але треба тільки оголосити частини об’єктів.
4. Інколи можна, інколи ні.
5. **Можна, але треба перевантажити оператор +.**

35. Що відбувається template <class X> class MyTemp1?

1. Оголошується клас X.
2. Оголошується клас MyTemp1.
3. Оголошується шаблон класу X.
4. **Оголошується шаблон класу MyTemp1.**
5. Оголошується 2 класи X та MyTemp1.

36. Що відбувається MyTemp1<A> a1,a2,a3?

1. Оголошується об’єкт A класу MyTemp1.
2. Оголошуються об’єкти класу A.
3. Оголошується об’єкт MyTemp1 класу A.
4. Оголошується шаблон класу A.
5. **Оголошуються об’єкти класу MyTemp1.**

37. Маємо запис: MyTemp1<A> a1,a2,a3, на що вказує <A>?

1. A – об’єкт класу a1.
2. На тип об’єкту MyTemp1.
3. **На тип узагальненого класу MyTemp1.**
4. На 4-тий параметр класу.

38. Маємо запис MyTemp1<A> a1,a2,a3; MyTemp1<B> b1,b2,b3. Чи можна написати a3.show(); b3.show()?

1. Не можна, бо об’єкти різних класів.
2. **Можна, але для кожного класу написати свої методи.**
3. Можна, але перевантажити класи.
4. Не можна, бо методи не знатимуть з якими об’єктами працювати.

39. Чи можна створювати узагальнений клас з полем масивом, тип елементів якого є параметром узагальненого класу?

1. Можна, але потрібно узагальнити клас масивом.
2. Не можна, з типом масив так не працюють.
3. **Можна.**

40. Що відбувається MyClass operator++()?

1. **Перевантажується оператор ++.**
2. Оголошується оператор ++.
3. Оголошується оператор ++().
4. Перевантажується оператор +.

41. Що буде відбуватися при ++obj2; якщо оператор ++ перевантажений?

1. Виникне помилка.
2. **Те, що написано при перевантаженні оператору.**
3. Те, що описано перед цими діями в коді головної програми.
4. Збільшиться змінна obj2 на +1.

42. Якого типу масив, що на картинці?

```
template <class X> class MyClass
{
   public:
   X array[n];
```

1. Типу class.
2. Типу array.
3. Узагальненого типу class X.
4. Типу X.
5. **Узагальненого типу X.**

43. Де помилка?

```
b1. b2.x = 10.1;
b1.res.y = 100.1;
b2.res.x = 20.2;
b2.res.y = 200.2;
b3 = b1 + b2;
```

1. Так додавати не можна.
2. Замість змінної res потрібно написати скрізь змінну b2.
3. Замість змінної res потрібно написати скрізь змінну b3.
4. Замість змінної x потрібно написати скрізь змінну y.
5. **Замість змінної b2 у першому рядку потрібно написати змінну res.**

44. Що зображено на картинці?

```
template <class X> void FindElement(X s, Elements<X> obj)
{
   int i, count=0;
   for (i=0; i<n; i++)
   if (s==obj[i]) count++;
   cout<<"resultat = "<<count<<endl;
};
```

1. Шаблон класу FindElement.
2. Шаблон класу Elements.
3. Узагальнений клас X.
4. **Шаблон функції FindElement**.
5. Шаблон функції class X.

45. Що відбудеться в результаті виконання коду:

```
template <class X> void FindElement(X s, Elements<X> obj)
{
   int i, count=0;
   for (i=0; i<n; i++)
   if (s==obj[i]) count++;
   cout<<"resultat = "<<count<<endl;
};
```

1. Виведеться знайдений елемент.
2. Виведеться кількість елементів масиву.
3. Виконається цикл.
4. **Виведеться кількість знайдених елементів таких як s.**
5. Виведеться кількість знайдених елементів таких як i.

46. Що відбувається Elements<int> a;?

1. На основі узагальненого класу Elements створюється об’єкт int.
2. На основі узагальненого класу Elements створюється об’єкт <int> .
3. На основі узагальненого класу Elements створюється узагальнений об’єкт a.
4. **На основі узагальненого класу Elements створюється об’єкт a.**

47. Чи потрібно писати такий код?

```
X operator[](int k)
{
   return array[k];
}
```

1. Потрібно, перевантажує оператор.
2. **Можна не писати, використовується лише для зручності.**
3. Писати обов’язково, повертає масив.

48. Що відбувається?

```
Elements ()
{
  for (int i=0; i<n; i++)
  {
     array[i] = (X)(rand()%20+100);
     cout<<array[i]<<" ";
  }
  cout<<endl;
}
```

1. Виводиться масив з елементами в межах від 0 до 120.
2. Виводиться масив з елементами в межах від 20 до 100.
3. Будується та виводиться масив з елементами в межах від 100 до –20.
4. **Будується та виводиться масив з елементами в межах від 100 до 120.**
5. Будується та виводиться масив з елементами в межах від 100, які поділені на 20.

49. Чи може узагальнена функція мати аргументом об’єкт узагальненого класу?
    1. Не може.
    2. Може, за певних умов.
    3. **Може.**
50. Чи може узагальнена функція мати аргументом змінну узагальненого типу?
    1. Не може.
    2. Може, за певних умов.
    3. **Може.**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Народный учебник по OpenGL. Уроки по OpenGL с сайта NeHe. Иркутск, 2005. – 357 с.
2. Ю.М. Баяковский, А.В. Игнатенко, А.И. Фролов. Графическая библиотека OpenGL, учебно-методическое пособие. Москва, 2003. – 132 с.
3. Власюк А.П., Мартинюк П.М., Прищепа О.В., Філатова І.А., Філатов М.С., Рощенюк А.М., Демчук О.С., Демчук М.Б., Мічута О.Р., Цвєткова Т.П., Федорчук Н.А. Лабораторний практикум з програмування. Навч. посібник / За загальною редакцією проф. Власюка А.П. – Рівне: НУВГП, 2011. – 495 с.
4. Креневич, А.П. С у задачах і прикладах : навчальний посібник із дисципліни "Інформатика та програмування" / А.П. Креневич, О.В. Обвінцев. – К. : Видавничо-поліграфічний центр "Київський університет", 2011. – 208 с.
5. Васильев А. Н.Самоучитель C++ с примерами и задачами. 4-е издание (переработанное). Книга + виртуальный CD. — СПб.: Наука и Техника, 2016. – 480 с.
6. КАК рисовать на СИ++. URL: http://radiofront.narod.ru/htm/prog/htm/paint.html
7. Тема 2. Графика Cpp. URL: https://www.youtube.com/watch?v=Wmh8uBph6FM (дата звернення 15.04.2021)
8. GitHub. Graphics-Library. URL: https://github.com/SagarGaniga/Graphics-Library (дата звернення 15.04.2021)
9. Справочник по C/C++. Графический режим. URL: http://mycpp.ru/cpp/scpp/cppd_graphics.h.html (дата звернення 15.04.2021)

10. Справочник по функціям C/C++. Функции по модулям. URL: http://www.codenet.ru/progr/cpp/spr/mod.php (дата звернення 15.04.2021)
11. Функции для работы с текстом и графикой. URL: http://www.c-cpp.ru/funkcii/funkcii-dlya-raboty-s-tekstom-i-grafikoy (дата звернення 15.04.2021)
12. Graphics.h. Рисование Дома, Дерева, Озера, Солнца. URL: http://ci-plus-plus-snachala.ru/?p=104 (дата звернення 15.04.2021)
13. C/C++ Graphics Tutorial 9 | Car animation. URL: https://www.youtube.com/watch?v=9a6vCSC5cbs&list=PL5UFsTza4wWSNhe0xuO6ELw7ORU-UHNDO&index=9&ab_channel=VCRGames (дата звернення 15.04.2021)
14. Using OpenGL & GLUT in Code::Blocks. URL: http://www.sci.brooklyn.cuny.edu/~goetz/codeblocks/glut/ (дата звернення 16.04.2021)
15. Народный учебник по OpenGL. Урок 4. Вращение полигонов. URL: http://pmg.org.ru/nehe/nehe04.htm (дата звернення 16.04.2021)
16. C++. Лекция 2. Указатели. Статистические и динамические массивы. URL: http://www.itmathrepetitor.ru/s-lekciya-2-ukazateli-staticheskie-i-din/
17. Задачі з вкладеними циклами. URL: https://sites.google.com/site/infinschool1/dla-ucniv/10-ti-klasi-profil/vkladeni-cikli (дата звернення 15.04.2021)
18. #9 Работа с файлами C++. URL: https://www.youtube.com/watch?app=desktop&v=_VL_5v14a_Q
19. Работа с файлами c++. Запись в файл c++ ofstream. Изучение C++ для начинающих. Урок #115. URL: https://www.youtube.com/watch?app=desktop&v=CBnB2fvfu_I&ab_channel=%23SimpleCode
20. Работа с файлами c++. Запись в файл c++ ofstream. Изучение C++ для начинающих. Урок #116. URL: https://www.youtube.com/watch?app=desktop&v=aUP0eAEIxog&ab_channel=%23SimpleCode

21. Работа с файлами с++. Запись в файл с++ ofstream. Изучение C++ для начинающих. Урок #118. URL: https://www.youtube.com/watch?app=desktop&v=bTysglLJ8No&list=PLQOaTSbfxUtCrKs0nicOg2npJQYSPGO9r&index=137&ab_channel=%23SimpleCode
22. Работа с файлами с++. Запись в файл с++ ofstream. Изучение C++ для начинающих. Урок #117. URL: https://www.youtube.com/watch?app=desktop&v=imYmUAR4QBs&ab_channel=%23SimpleCode
23. Работа с файлами с++. Запись в файл с++ ofstream. Изучение C++ для начинающих. Урок #119. URL: https://www.youtube.com/watch?app=desktop&v=z9m5XpOAxm0&ab_channel=%23SimpleCode
24. C++ для начинающих. Работа с бинарными файлами. URL: https://ciplusplussnachala.wordpress.com/2012/08/26/c-%D0%B4%D0%BB%D1%8F-%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D1%81-%D0%B1%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%BC%D0%B8-%D1%84%D0%B0/
25. Программирование на C++. Урок 64. Бинарный доступ к файлу. URL: https://www.youtube.com/watch?app=desktop&v=0kOMg3YLrEg&ab_channel=%D0%9C%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%92%D0%BE%D0%BB%D0%BA%D0%BE%D0%B2
26. Чтение/запись в бинарный файл C++. URL: https://ru.stackoverflow.com/questions/636262/%d0%a7%d1%82%d0%b5%d0%bd%d0%b8%d0%b5-%d0%b7%d0%b0%d0%bf%d0%b8%d1%81%d1%8c-%d0%b2-%d0%b1%d0%b8%d0%bd%d0%b0%d1%80%d0%bd%d1%8b%d0%b9-%d1%84%d0%b0%d0%b9%d0%bb-%d0%a1?rq=1
27. Народный учебник по OpenGL. Урок 5. Создание фигур в 3D. URL: http://pmg.org.ru/nehe/nehe05.html

28. Уроки OpenGL. Часть 4. Создание 3D-объектов. URL: https://microtechnics.ru/opengl-urok-4-sozdanie-3d-obektov/
29. Як відобразити коло в OpenGL. URL: https://uk.know-net.org/renderizar-circulo-opengl-como-330563-559
30. Программирование графики OpenGL. Сечение/Пунктирные линии для полигонов Delphi + OpenGL. URL: https://www.cyberforum.ru/opengl/thread1281970.html
31. ВикиЧтение. Стандартные геометричиские примитивы. URL: https://it.wikireading.ru/24714
32. Задачі з вкладеними циклами. URL: https://sites.google.com/site/infinschool1/dla-ucniv/10-ti-klasi-profil/vkladeni-cikli

УДК 004.42(075.8)

Г 54

Електронне мережне навчальне видання

Глинчук Л. Я., Гришанович Т.О.

ПІДРУЧНИК

ПРОГРАМУВАННЯ

Для студентів спеціальностей

122 «Комп’ютерні науки»

113 «Прикладна математика»

014 «Середня освіта (Інформатика)»