**Лабораторна робота №4**

**Використання шаблонів при розробці класів**

**Мета роботи**: Ознайомитись з базовими механізмами використання шаблонів функцій. Навчитись створювати та використовувати шаблони класів.

**Загальні вказівки**

Лабораторна робота складається з одного завдання.

Необхідно розробити клас згідно варіанту, використовуючи перевантаження зазначених операторів. Перевірити правильність роботи програми.

Перед виконанням завдання необхідно ознайомитись з теоретичними відомостями та матеріалами лекцій.

Для перевірки надається звіт та файл з текстом розробленої програми.

Структура звіту:

1. Титульний аркуш

2. Текст завдання.

3. Хід виконання завдання з детальними поясненнями, що включають:

3.1. Опис вхідних та вихідних даних; алгоритм розв'язання завдання з доцільними поясненнями.

3.2. Лістинги програм.

3.3. Розробка тестів та результати тестування. Скріншоти вхідних та результатів розв'язання завдання на ЕОМ.

4. Висновки по роботі.

**Теоретичні відомості .**

**Шаблони функцій**

Шаблони функцій – це потужний інструмент у С++, який суттєво спрощує роботу програміста. Наприклад, нам потрібно запрограмувати функцію, яка виводила б на екран елементи масиву. При цьому ми хочемо, щоб функція виводила масиви типу int, double, float і char. Тобто, нам потрібно запрограмувати 4 функції, які виконують одні й ті самі дії, але для різних типів даних. Скористаємося перевантаженням функцій.

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;

}

Таким чином, ми маємо 4 перевантажені функції, для різних типів даних. Ввони відрізняються тільки заголовком, тіло у них абсолютно однакове.

А що якщо, нам знадобиться запрограмувати алгоритм сортування у вигляді функції. Виходить, для кожного типу даних доведеться свою функцію створювати. Тобто, один і той же код буде в декількох примірниках, що дуже нераціонально. Тому в С ++ придуманий такий механізм – шаблони функцій.

**Шаблони функцій** – це інструкції, згідно з якими створюються локальні версії функції для певного набору параметрів і типів даних.

Синтаксис:

**template <class T>**

**template <typename T>**

**template <typename T1, typename T2>**

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

Ключове слово typename говорить про те, що у шаблоні буде використовуватися вбудований тип даних, такий як: int, double, float, char і т. д. А ключове слово class повідомляє компілятору, що в шаблоні функції як параметр будуть використовуватися типи даних користувача, тобто класи.

Ми створюємо один шаблон, в якому описуємо всі типи даних. Таким чином код не буде захаращуватися.

Приклад 1. Програма виводить на екран елементи масивів різних типів, використовуючи шаблон функції.

#include <iostream>

using namespace std;

    // шаблон функції printArray

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 iSize = 10, dSize = 7, fSize = 10, cSize = 5;

       // масиви різних типів даних

  int    iArray[iSize]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  double dArray[dSize]={1.2345,2.234,3.57,4.67876,5.346, 6.1545, 7.7682};

  float  fArray[fSize]={1.34,2.37,3.23,4.8,5.879,6.345,73.434,8.82,9.33,10.4};

  char cArray[cSize] = {"MARS"};

cout << "\n\t Використання шаблонiв функцiй:\n";

cout << "\n\tМасив типу int:\n\t";  printArray(iArray, iSize);

cout << "\n\tМасив типу double:\n\t"; printArray(dArray, dSize);

cout << "\n\tМасив типу float:\n\t"; printArray(fArray, fSize);

cout << "\n\tМасив типу char:\n\t"; printArray(cArray, cSize);

system("pause");

return 0;

}

Результат роботи програми:



**Шаблони класів**

У міру того, як кількість створюваних класів зростає, виявляється, що деякий клас, створений для однієї програми, необхідний в іншій програмі. У багатьох випадках класи можуть відрізнятися тільки типами. Наприклад, один клас працює з цілочисельними значеннями, в той час як в іншій програмі він повинен працювати зі значеннями типу float.

Щоб збільшити ймовірність повторного використання існуючого коду, C++ дозволяє програмам визначати шаблони класів.

**Шаблон класу** визначає тіпонезалежний клас, який надалі служить для створення об'єктів необхідних типів.

Якщо компілятор C++ зустрічає оголошення об'єкта, засноване на шаблоні класу, то для побудови класу необхідного типу він буде використовувати типи, зазначені при оголошенні. Дозволяючи швидко створювати класи, що відрізняються тільки типом, шаблони класів скорочують обсяг програмування, що, в свою чергу, заощаджує час.

Пояснимо використання шаблонів класів на простому прикладі. Нехай необхідно створити простий клас "Масив", який буде виконувати прості дії: Додавання і Відображення елементів.

Приклад. Створення класу "Масив" без шаблону.

#include <iostream>

using namespace std;

         /\*НАШ КЛАСС\*/

 class Massiv

 {

   int Аrray[10]; //Масив цiлочислених значень з 10 елементiв

   int count;        //Лiчильник елементiв масиву

  public:

    void Add(int );   //Метод для додавання елементiв в масив

    void Show();  //Метод для вiдображення масиву на екранi

 };

 void Massiv::Add(int x)

 {

   static int pos=0;

   Array[pos]=x;

   pos++;

   count=pos;

 }

 void Massiv::Show()

 {

  for (int i=0;i<count;i++)

      cout<<Array[i]<<"\t";

  cout<<endl;

 }

int main()

{

  Massiv Arr;

  Arr.Add(100.555);

  Arr.Add(200);

  Arr.Add(300);

  Arr.Show();

  system("pause");

  return 0;

}

Це приклад створення звичайного класу. Але у програмістів іноді виникає необхідність створення такого ж класу, в якому відрізняється тільки тип даних. Наприклад, може знадобитися створення класу, в якому потрібно створення масиву, який буде зберігати в собі і обробляти не цілочисельні змінні, а рядкові. Як варіант, можна дописати купу класів для кожного з типів змінних, але це не раціонально. Код вийде більшим, громіздким. Чим більше коду, тим простіше в ньому помилятися і тим складніше шукати. Ось тут і приходять на допомогу шаблони класів.

Приклад 2. Створення класу "Масив" з використанням шаблону.

#include <iostream>

using namespace std;

int pos=0; //Позицiя в масивi

template <class T> //Шаблон с класу з параметром T

class Massiv

{

T Array[10]; //Масив цiлочислених значень з 10 елементiв

int count; //Лiчильник елементiв масиву

public:

void Add(T ); //Метод для додавання елементiв в масив

void Show(); //Метод для вiдображення масиву на екранi

};

template <class T> void Massiv<T>::Add(T x)

{ static int pos=0;

Array[pos]=x;

pos++;

count=pos;

}

template <class T> void Massiv<T>::Show()

{ cout<<"\t"<<endl;

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

cout<<"\t"<<Array[i];

cout<<endl;

}

int main()

{

setlocale(0,"ukr");

Massiv<int> Arr;

Arr.Add(100.555);

Arr.Add(200);

Arr.Add(300);

Arr.Show();

Massiv<char \*> Arr2;

Arr2.Add("Рядок");

Arr2.Add("Цiкавий проект");

Arr2.Add("Вийшло");

Arr2.Show();

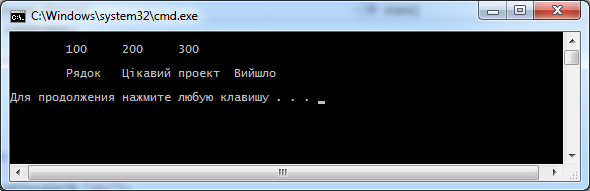
cout<<endl;

system("pause");

return 0;

}

Результат роботи програми:



*Приклад 3*. Шаблон класу, конструктор якого породжує об’єкти з двома параметризованими полями.

#include <iostream>

using namespace std;

template <class T1>

class X

{protected:

T1 a, b;

public:

X(T1 i, T1 j);

void Print(void) {

cout << " a = " << a << " b = " << b << endl;

}

};

template <class T1> X <T1> :: X(T1 i, T1 j) // Конструктор

{

a = i; b = j;

cout << " Object created, size = " << sizeof(T1) << endl;

}

void main(void) {

X <int> x1(2, 3); // цілочисельні

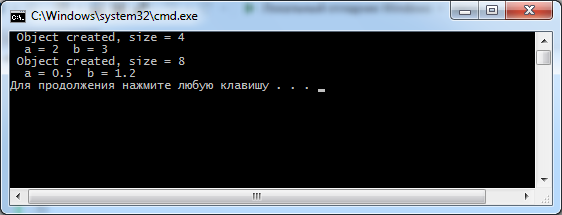
x1.Print();

X <double> x2(0.5, 1.2); // дійсні числа

x2.Print();

}

Результат роботи програми



Завдання на лабораторну роботу

**Завдання 1.**

Загальна постановка. Дано: число N і послідовність a1, a2, ... aN

Створити шаблон класу, що породжує динамічні одномірні масиви з елементами різних типів (речові, цілочисельні, символьні тощо). Тип даних і результат є параметрами по відношенню до класу. Програма повинна мати методи ініціалізації, конструктор, деструктор, перегляд значень створеного масиву, а також для вирішення задачі формування нового масиву за такими алгоритмами:

Варіант 1. a1, (a1+a2), … ,(a1+a2+…+aN)

Варіант 2. (a1\*a1), (a1\*a2), …, (a1\*aN)

Варіант 3. |a1|, |a1+a2|, …, |a1+a2+…aN|

Варіант 4. a1, -a1\*a2, +a1\*a2\*a3, … ,(-1)N\*a1\*a2\*…aN

Варіант 5. -a1, +a2, -a3, … , (-1)N\*aN

Варіант 6. (a1+1), (a2+2) , (a3+3), …, (aN+N)

Варіант 7. a1\*1, a2\*2, a3\*3,     , aN\*N

Варіант 8. a1\*a2, a2\*a3, … , aN-1\*aN

Варіант 9. a1/1, a2/2, a3/3, …,aN/N

Варіант 10. (a1+a2), (a2+a3),… ,(aN-1+aN)

Варіант 11. (a1+a2+a3), (a2+a3+a4), (a3+a4+a5), … (aN-2+aN-1+aN)

Варіант 12. (N+a1), ( N-1+a2),   ,(1+aN)

Варіант 13. (N\*a1), ( (N-1)\*a2),   ,(1\*aN)

Варіант 14. a1/N, a2/N, … ,aN/1

Варіант 15. a1, (a1-a2), … ,(a1-a2-…-aN)

Варіант 16. (a1/a1), (a1/a2), …, (a1/aN)

Варіант 17. |a1|, |a1-a2|, …, |a1-a2-…-aN|

Варіант 18. a1, -a2\*a2, a3\*a3\*а3, … , (-1)N\*aN\*aN\*…\*aN

Варіант 19. (N\*a1), ( (N+1)\*a2),   ,((N+N)\*aN)

Варіант 20. a1\*N, a2\*N\*N, … ,aN\*NN

Варіант 21. -a1, +a2, -a3, … , (-1)N\*aN

Варіант 22. (a1-1), (a2-2) , (a3-3), …, (aN-N)

Варіант 23. (a1)1, (a2)2, (a3)3,     , (aN)N

Варіант 24. (a1-a2-a3), (a2-a3-a4), (a3-a4-a5), … , (a(N-2)-a(N-1)-aN)

Варіант 25. a1/(a2\*a3), a2/(a3\*a4), … , a(N-2)/(a(N-1)\*aN)

Варіант 26. a1N, a2(N-1), a3(N-2), …,aN1

Варіант 27. (a1-a2), (a2-a3),… ,(a(N-1)-aN)

Варіант 28. (N2+a1), (( N-1)2+a2),   ,(12+aN)

Варіант 29. a1/(aN\*a(N-1)), a2/(a(N-1)\*a(N-2)), a3/(a(N-2)\*a(N-3)), … a(N-2)/(a2\*a1)

Варіант 30. |a1N|, |a1(N-1)|, …, |aN1 |

Варіант 31. Na1, Na2,   ,NaN

**Завдання 2**

Створити клас, який описує та забезпечує дії над даними параметризованого двовимірного масиву, розмірність якого визначається під час роботи програми. Завдання щодо обробки масиву взяти з лабораторної роботи 1. Усі обчислення і перетворення реалізувати у вигляді функцій-членів класу.