**Вопросы к экзамену по курсу “ЛИПО САПР” 4 семестр (часть 1).**

**(**группы А-6,9 -20 )

1. **Шаблоны функций. Определение. Пример. Использование для стандартных и пользовательских типов**

Параметрический полиморфизм

• Параметрический полиморфизм позволяет многократно использовать один и тот же код применительно к разным типам данных

- тип указывается как параметр функции или класса

• Шаблоны (templates) - средство для реализаций параметризированных классов и функций на языке C++

В языке **C++ шаблоны функций** — **это функции**, которые служат образцом для создания других подобных **функций**. Главная идея — создание **функций** без указания точного типа(ов) некоторых или всех переменных. Для этого мы определяем **функцию**, указывая тип параметра шаблона, который используется вместо любого типа данных.

|  |  |  |
| --- | --- | --- |
| //с одним параметром  template<class TYPE>  void swap(TYPE& x, TYPE& y)  {  TYPE temp;  temp = x;  x = y;  y = temp;  }  int main()  {  double x = 1, y = 5;  swap<double>(x,y);  } | //с двумя параметрами  template<class TYPE1, class TYPE2>  void swap(TYPE1& x, TYPE2& y)  {  TYPE1 temp;  temp = x;  x = y;  y = temp;  }  int main()  {  double x = 1;  int y = 3;  swap(x,y);  } | //c пользовательских типом  class complex  {  double x, y;  public:  complex()  {  x = 0;  y = 0;  }  complex(double a, double b)  {  x = a;  y = b;  }  };  template<class TYPE1, class TYPE2>  void swap(TYPE1& x, TYPE2& y)  {  TYPE1 temp;  temp = x;  x = y;  y = temp;  }  int main()  {  complex x;  complex y(4,3);  swap(x,y);  } |

1. **Шаблоны классов. Определение, пример.**

Шаблонный класс это шаблон для класса, который инстанцирован в зависимости от типа (типов) и других полученных шаблоном аргументов. Инстанциация сгенерирует шаблонный класс, или специализацию этого класса. Разберемся со следующим:

когда происходит наследование, что именно наследуетcя:

шаблон класса (Item< T >), или

специализацию (Item< int >) ?

#include <iostream>

#include <stdlib.h>

using namespace std;

// Описание шаблона Класса

template <class T, class T1>

class array

{

T \*data;

int size;

int index;

public:

array(int size);

T1 sum(void);

T average\_value(void);

void show\_array(void);

int add\_value(T);

};

//описание конструктора шаблона класса вне класса

template <class T, class T1>

array<T, T1>::array(int size)

{

data = new T[size];

if (data == NULL)

{

cerr << "Недостаточно памяти - программа завершается" << endl;

exit(1);

}

array::size = size;

array::index = 0;

}

//функция вычисления суммы

template <class T, class T1>

T1 array<T, T1>::sum(void)

{

T1 sum = 0;

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

sum += data[i];

return (sum);

}

//вычисление среднего

template <class T, class T1>

T array<T, T1>::average\_value(void)

{

T1 sum = 0;

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

sum += data[i];

return (sum / index);

}

//вывод массива

template <class T, class T1>

void array<T, T1>::show\_array(void)

{

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

cout << data[i] << ' ';

cout << endl;

}

//добавление элемента в массив

template <class T, class T1>

int array<T, T1>::add\_value(T Value)

{

if (index == size)

return (-1); // Массив полон

else

{

data[index] = Value;

index++;

return (0); //Успешно

}

}

int main(void)

{

// Массив из 100 элементов целого типа

array<int, long> numbers(100);

//Массив из 200 элементов вещественного типа

array<float, float> values(200);

int i;

for (i = 0; i < 50; i++)

numbers.add\_value(i);

numbers.show\_array();

std::cout << "Сумма чисел равна " << numbers.sum() << std::endl;

std::cout << "Среднее значение равно " << numbers.average\_value() << std::endl;

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

values.add\_value(i \* 100);

values.show\_array();

std::cout << "Сумма чисел равна." << values.sum() << std::endl;

std::cout << "Среднее значение равно " << values.average\_value() << std::endl;

}

1. **Шаблоны классов и дружественность. Пример.**

Для шаблонов классов также могут быть установлены отношения дружественности.

Дружественность может быть установлена:

• Между шаблоном класса и глобальной функцией,

• Методом другого класса (возможно, шаблонного класса) или даже целым классом (возможно, шаблонным классом).

Если внутри шаблона класса, объявленного как

template <…>

class X{…friend void f1( ); }

находится объявление дружественной функции , то функция f1 является дружественной для каждого класса, полученного как реализация из данного шаблона.

#include <iostream>

using namespace std;

template <class T>

class complex;

template <class T>

std::ostream &operator<<(std::ostream &out, const complex<T> &A)

{

if (A.y >= 0)

out << A.x << "+" << A.y << "i";

else

out << A.x << A.y << "i";

return out;

}

template <class T>

std::istream &operator>>(std::istream &in, complex<T> &A)

{

cout<<("введите действительную часть: ");

in >> A.x;

cout<<("введите мнимую часть: ");

in >> A.y;

return in;

}

template <class T>

class complex

{

T x, y;

public:

complex()

{

x = 0;

y = 0;

}

complex(T a, T b)

{

x = a;

y = b;

}

friend std::ostream &operator<<<T>(std::ostream &out, const complex &A);

friend std::istream &operator>><T>(std::istream &in, complex &A);

};

int main()

{

complex<double> A2,A3(3,4);

cin >> A2;

cout << A2;

cout << A3;

}

1. **Шаблоны классов и наследование. Пример.**

#include <iostream>

#include <string.h>

#include <typeinfo>

using namespace std;

template <class T>

class Myclass

{

protected:

T value;

public:

Myclass(T value1) { value = value1; }

void DataTypeSize() { cout << "razmer =" << sizeof(value) << endl; }

};

// наследование шаблона от шаблона

template <class T>

class TypeInfo : public Myclass<T>

{

public:

TypeInfo(T value1) : Myclass<T>(value1){};

void ShowTypeName()

{

cout << "imj tipa" << endl;

cout << typeid(Myclass<T>::value).name() << endl;

}

};

int main(void)

{

int a = 5;

TypeInfo<int> c(a);

c.DataTypeSize();

c.ShowTypeName();

double f = 5.345;

TypeInfo<double> c1(f);

c1.DataTypeSize();

c1.ShowTypeName();

return 0;

}

1. **Стандартная библиотека шаблонов. Контейнеры и итераторы. Примеры определения последовательных контейнеров. Заполнение контейнеров. Пример**
2. **Стандартная библиотека шаблонов Итераторы и примеры определения и работы с ними..**
3. **Стандартная библиотека шаблонов. Общие свойства контейнеров. Операции с контейнерами. Примеры (размер контейнера, проверка пустой или нет и др.)**
4. **Стандартная библиотека шаблонов. Алгоритмы. Примеры работы со стандартными алгоритмами)**
5. **Стандартная библиотека шаблонов. Последовательные контейнеры. Примеры vector.**
6. **Стандартная библиотека шаблонов. Последовательные контейнеры. Примеры list/**

Тематика задач:

1. Шаблоны функций.
2. Шаблоны классов.
3. Библиотека STL на примере использования последовательных контейнеров.(vector,list)

**Вопросы к экзамену по курсу ЛИПО САПР (2 часть)**

**Автоматизация конструкторского и технологического проектирования.**

**(**группы А-6,9 -18 )

1. Технологии разработки печатных плат (однослойные, двухсторонние, МПП).
2. Модели схем в конструкторском проектировании. Модели элементов, модели цепей. Примеры.
3. Задача компоновки, классификация, критерии. Используемые модели, в задаче компоновки.
4. Алгоритмы компоновки, классификация. Последовательный алгоритм разрезания мультиграфа.
5. Итерационные алгоритмы компоновки. Пример.
6. Гиперграф как модель схемы в алгоритмах разрезания. Сравнение с моделью мультиграфа.
7. Формализация задачи разрезания гиперграфа.
8. Алгоритм последовательного разрезания на блоки (модель схемы гиперграф).
9. Задача компоновки. Алгоритмы покрытия. Классификация.
10. Последовательно - приближенный алгоритм покрытия схемы по шагам.(связные модули).
11. Покрытие схемы модулями “россыпью”.
12. Размещение элементов, Классификация алгоритмов размещения и критерии качества..
13. Последовательно-итерационный алгоритм размещения. Последовательный этап. Пример.
14. Последовательно-итерационный алгоритм размещения. Итерационный этап. Пример.
15. Алгоритмы трассировки Общая характеристика задачи, зависимость критериев качества от технологии реализации элементов.
16. Алгоритм Прима. Компьютерный алгоритм , пример по шагам.
17. Формализация задачи трассировки. Коммутационное поле и размер дискрета, соседство, приоритет координат, функции качества алгоритма. Классификация алгоритмов.
18. Волновой алгоритм, исходные данные,основные шаги. Пример. Достоинства и недостатки волнового алгоритма.
19. Основные составляющие волнового алгоритма, позволяющие учитывать заданные ограничения разработчика схемы. Соседство, путевые координаты, весовые функции. Примеры. Способы улучшения волнового алгоритма.
20. Построение путей волновым алгоритмом по заданным параметрам(мин изгибов, мин пересечений и т.д.)
21. Топологическая и гибкая трассировка достоинства и недостатки. Модель схемы в гибкой трассировке.
22. Многослойная трассировка. МПП с открытыми контактными площадками. Основные задачи автоматизации. Отличия по сравнению с МПП.
23. Графо-теоретический подход к трассировке. Модели элементов. Основные решаемые задачи.

**Образцы задач для решения в билетах**

Моделью схемы является мультиграф, для заданного фрагмента, например, 4

Улучшить разбиение, если первый подграф содержит вершины v1,v2,v3,v4, а второй все остальные.

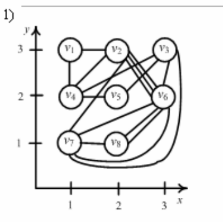


Моделью схемы является гиперграф, заданный фрагмент схемы разбить на две части с минимальным количеством цепей в разрезе Число элементов в каждой подсхеме равно трем.



Задача

Дан фрагмент схемы 1) или 2), надо разместить его на заданной подложке 3х3 по критерию минимальной суммарной длины соединений . Для подсчета длины соединений использовать формулу abs(xi-xj)+abs(yi-yj).



**1 2 3 4 5 6 7 8**

**1** 0 1 0 1 0 0 0 0

**2** 1 0 0 1 0 3 1 0

**3** 0 0 0 1 1 1 1 0

**4** 1 1 1 0 1 0 0 0

**5** 0 0 1 1 0 0 0 0

**6** 0 3 1 0 0 0 2 2

**7** 0 1 1 0 0 2 0 1

**8** 0 0 0 0 0 2 1 0

K1(V2) = 2S21 – P(V2) = 2 \*1 – 6 = -4

K1(V3) = 2S31 – P(V3) = 2 \*0 – 4 = -4

K1(V4) = 2S41 – P(V4) = 2 \*1 – 4 = -2

K1(V5) = 2S51 – P(V5) = 2 \*0 – 2 = -2

K1(V6) = 2S61 – P(V6) = 2 \*0 – 7 = -7

K1(V7) = 2S71 – P(V7) = 2 \*0 – 5 = -5

K1(V8) = 2S81 – P(V8) = 2 \*0 – 3 = -3

|  |  |  |
| --- | --- | --- |
| V1 | V4 |  |
| V5 |  |  |
|  |  |  |

K2(V2) = 2(S21+S24+S25) –P(V2) = 2\*(1+1+0) – 6 = -2

K2(V3) = 2(S31+S34+S35) – p(V3) = 2\*(0+1+1) - 4 = 0

K2(V6) = 2(S61+S64+S65) – p(V6) = 2\*(0+0+0) - 7 = -7

K2(V7) = 2(S71+S74+S75) – p(V7) = 2\*(0+0+0) - 5 = -5

K2(V8) = 2(S81+S84+S85) – p(V8) = 2\*(0+0+0) - 3 = -3

|  |  |  |
| --- | --- | --- |
| V1 | V4 |  |
| V5 | V3 |  |
|  |  |  |

K3(V2) = 2(S21+S24+S25+S23) –P(V2) = 2\*(1+1+0+0) – 5 = -2

K3(V6) = 2(S61+S64+S65+S63) –P(V6) = 2\*(0+0+0+0) – 7 = -7

K3(V7) = 2(S71+S74+S75+S73) –P(V7) = 2\*(0+0+0+1) – 5 = -3

K3(V8) = 2(S81+S84+S85+S83) –P(V8) = 2\*(0+0+0+0) – 3 = -3

|  |  |  |
| --- | --- | --- |
| V1 | V4 |  |
| V5 | V3 |  |
| V2 |  |  |

K4(V6) = 2(S61+S64+S65+S63+S62) –P(V6) = 2\*(0+0+0+0+3) – 7 = -1

K4(V7) = 2(S71+S74+S75+S73+S72) –P(V7) = 2\*(0+0+0+1+1) – 5 = -1

K4(V8) = 2(S81+S84+S85+S83+S82) –P(V8) = 2\*(0+0+0+0+0) – 3 = -3

|  |  |  |
| --- | --- | --- |
| V1 | V4 | V7 |
| V5 | V3 | V8 |
| V2 | V6 |  |

Для трассировки.

Задано дискретное поле и ограничения и контакты, принадлежащие разным цепям. Контакты, принадлежащие одной цепи , имеют одинаковый номер. Для варианта 3, например, соединить контакты 3-ей цепи. Использовать соседство по 4-м сторонам, приоритет путевых координат задать так, чтобы получить минимум поворотов. Функцию веса задать самостоятельно.

