Объектно-ориентированное программирование на алгоритмическом языке C++

МИРЭА, Институт Информационных технологий,
кафедра Вычислительной техники

Автор: доцент, канд. физ.-мат. наук,
Путуридзе Зураб Шотаевич

Контейнеры

Контейнеры – это объекты, которые содержат другие объекты

bitset	Множество битов	<bitset>
deque	Двусторонняя очередь	<deque>
list	Линейный список	<list>
map	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано только одно значение	<map>
multimap	Ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано два или более значений	<map>
multiset	Множество, в котором каждый элемент не обязательно уникален	<set>
priority_queue	Очередь с приоритетом	<queue>
queue	Очередь	<queue>
set	Множество, в котором каждый элемент уникален	<set>
stack	Стек	<stack>
vector	Динамический массив	<vector>

Итераторы

Итераторы – это объекты, которые действуют подобно указателям и реализуют инструменты доступа к элементам контейнеров.

Итератор	Описание
Произвольного доступа (random access)	Используется для считывания и записи значений. Доступ к элементам произвольный
Двунаправленный (bidirectional)	Используется для считывания и записи значений. Может проходить контейнер в обоих направлениях
Однонаправленный (forward)	Используется для считывания и записи значений. Может проходить контейнер только в одном направлении
Ввода (input)	Используется только для считывания значений. Может проходить контейнер только в одном направлении
Вывода (output)	Используется только для записи значений. Может проходить контейнер только в одном направлении

Векторы

Векторы представляют собой динамические массивы

#include <vector>

vector < тип > «имя вектора»

vector < тип > :: iterator «имя итератора»

Item_0	Item_1	Item_2	Item_3	….	Item_n

Некоторые методы вектора

Метод	Описание
at (i)	Возвращает ссылку на элемент, заданный параметром i
back ()	Возвращает ссылку на последний элемент вектора
begin ()	Возвращает итератор первого элемент вектора
capacity ()	Возвращает текущую емкость вектора
clear ()	Удаляет все элементы вектора
empty ()	Возвращает истинное значение если вектор пуст, иначе ложь
end ()	Возвращает итератор для конца вектора
erase (iterator it)	Удаляет элемент, на который указывает итератор it. Возвращает итератор элемента, который расположен следующим за удаленным
erase (iterator start, iterator end)	Удаляет элементы, заданные между итераторами start и end. Возвращает итератор элемента, который расположен следующим за последним удаленным
front ()	Возвращает ссылку на первый элемент вектора
insert (iterator it, const T & val = T ( ))	Вставляет параметр val перед элементом, заданным итератором it. Возвращает итератор элемента
pop_back ( )	Удаляет последний элемент вектора
push_back (const T & val)	Добавляет в конец вектора элемент, значение которого равно параметру val

Пример использования вектора

```
#include <iostream>
#include <vector>
using namespace std;
int main ( ) {
  vector < char > v, v2;
  unsigned int    i;

  for ( i = 0; i < 10; i++ ) v.push_back ( 'A' + i );
  cout << "Contents of v :\n";
  for ( i = 0; i < v.size (); i++ ) cout << v [ i ] << " ";
  cout << endl << endl;

  char str[] = "-STL Power-";       // инициализация второго вектора
  for ( i = 0; str [ i ]; i++ ) v2.push_back ( str [ i ] );

  vector < char > :: iterator p       = v.begin  () + 5;
  vector < char > :: iterator p2start = v2.begin ();
  vector < char > :: iterator p2end   = v2.end   ();
  v.insert ( p, p2start, p2end );

  cout << "Contents of v :\n";
  for ( i = 0; i < v.size (); i++ ) cout << v [ i ] << " ";
  return 0;
}
```

Ответ примера

```
Contents of v :
A B C D E F G H I J

Contents of v :
A B C D E - S T L   P o w e r - F G H I J
```

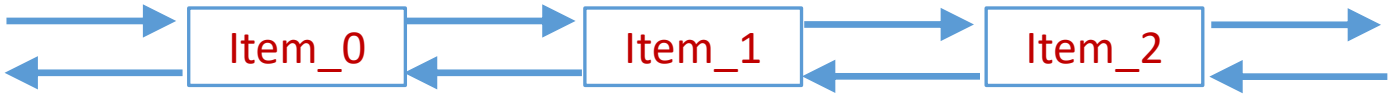
Список

Список - это контейнер с двунаправленным последовательным доступом к его элементам

```
#include <list>

list < тип >  «имя списка»

list < тип >  :: iterator «имя итератора»
```



Некоторые методы списка

Метод	Описание
assign (m, const T & значение = T())	Присваивает списку m элементов, каждый элемент равно параметру значение
back ( )	Возвращает ссылку на последний элемент списка
begin ( )	Возвращает итератор первого элемент списка
clear ( )	Удаляет все элементы списка
empty ( )	Возвращает истинное значение если список пуст, иначе ложь
end ( )	Возвращает итератор конца списка
Erase (iterator it)	Удаляет элемент, на который указывает итератор it. Возвращает итератор, указывающий на элемент, который расположен после удаленного
erase (iterator start, iterator end)	Удаляет элементы, заданные между итераторами start и end. Возвращает итератор, указывающий на элемент, который расположен за последним удаленным
front ( )	Возвращает ссылку на первый элемент списка
insert (iterator it, const T & val = T ( ))	Вставляет параметр val перед элементом, заданным итератором it. Возвращает итератор элемента
pop_back ( )	Удаляет последний элемент списка
push_back (const T & val)	Добавляет в конец списка элемент, значение которого равно параметру val

Пример использования списка

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
    list < char > lst; // создание пустого списка
    int  i;
    for ( i = 0; i < 10; i++ ) lst.push_back ( 'A' + i );
    cout << "Размер = " << lst.size () << endl;

    list < char > :: iterator p;
    cout << "Содержимое: ";
    while ( ! lst.empty ( ) ) {
        p = lst.begin ( );
        cout << *p;
        lst.pop_front ( );
    }
    return 0;
}
```

```
Размер = 10
Содержимое: ABCDEFGHIJ
```

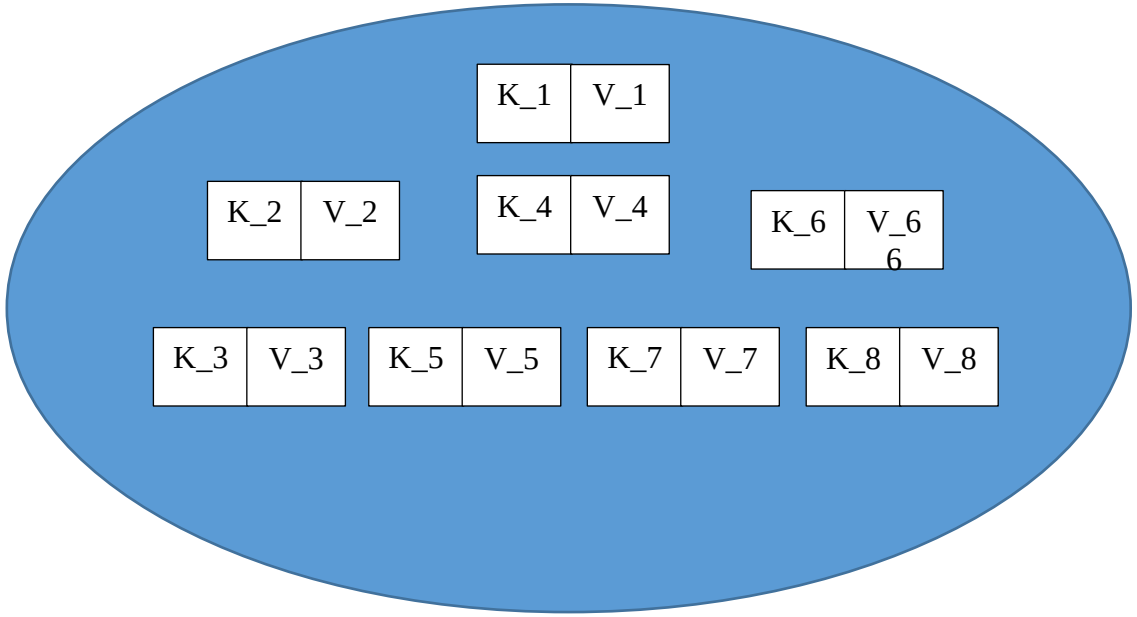
Ассоциативный список

Ассоциативный список - контейнер, в котором уникальным ключам соответствуют определенные значения

```
#include <map>
```

```
map < тип key, тип val >  «имя ассоциативного списка»
```

```
map < тип key, тип val >  ::  iterator «имя итератора»
```



Некоторые методы ассоциативного списка

Метод	Описание
assign (m, const T & значение = T())	Присваивает списку m элементов, каждый элемент равно параметру значение
back ( )	Возвращает ссылку на последний элемент списка
begin ( )	Возвращает итератор первого элемент списка
clear ( )	Удаляет все элементы списка
empty ( )	Возвращает истинное значение если список пуст, иначе ложь
end ( )	Возвращает итератор конца списка
Erase (iterator it)	Удаляет элемент, на который указывает итератор it. Возвращает итератор, указывающий на элемент, который расположен после удаленного
erase (iterator start, iterator end)	Удаляет элементы, заданные между итераторами start и end. Возвращает итератор, указывающий на элемент, который расположен за последним удаленным
front ( )	Возвращает ссылку на первый элемент списка
insert (iterator it, const T & val = T ( ))	Вставляет параметр val перед элементом, заданным итератором it. Возвращает итератор элемента
pop_back ( )	Удаляет последний элемент списка
push_back (const T & val)	Добавляет в конец списка элемент, значение которого равно параметру val

Полиморфизм

При конструировании новых версии изделий, многие элементы сохраняют форму (наименование, способ активизации), но меняют функционал поведения.

Полиморфизм – механизм обеспечивающий возможность определения различных реализации метода одним названием для классов различных уровней иерархии.

Наследование – конструирование иерархии классов.

Полиморфизм – изменение функционала в иерархии классов.

Виртуальный метод

Виртуальнаый метод – метод который объявляется в базовом классе с использованием ключевого слова virtual и реализацию которой можно переопределить в производном классе.

```
class base {
public:
      virtual void who() { cout << «Базовый класс \n"; }
};
class first_d : public base {
public:
      void who() { cout << «Первый производный класс \n"; }
};
class second_d : public base {
public:
       void who() { out << " Второй производный класс  \n"; }
};
```

Виртуальный метод (пример)

```
int main ( ) {
    base      base_obj;
    base    * p;
    first_d   first_obj;
    second_d  second_obj;

    p = & base_obj;
    p -> who ();   // вызов who из объекта базового класса

    p = & first_obj;
    p -> who ();   // вызов who из объекта класса first_d

    p = & second_obj;
    p -> who ();   // вызов who из объекта класса second_d

    return 0;
}
```

Наследование виртуального метода

Если метод объявляется как виртуальный, он остается такой независимо от того, через сколько уровней производных классов он может пройти.

Если виртуальный метод в классе не переопределен, то отрабатывает переопределенный ближайший по иерархии классов.

Наследование виртуального метода (пример)

```
class base {
public:
      virtual void who() { cout << «Базовый класс \n"; }
};
class first_d : public base {
public:
      void who() { cout << «Первый производный класс \n"; }
};

class second_d : public first_d {
public:
};
```

Наследование виртуального метода (пример)

```
int main ( ) {
  base       base_obj;
  base     * p;
  first_d    first_obj;
  second_d   second_obj;

  p = & base_obj;
  p -> who ();    // вызов who из объекта базового класса

  p = & first_obj;
  p -> who ();    // вызов who из объекта класса first_d

  p = & second_obj;
  p -> who ();    // вызов who из объекта класса first_d

  return 0;
}
```

Чисто виртуальные методы и абстрактные классы

Чисто виртуальный метод – это виртуальный метод, который не имеет реализации в базовом классе.

```
virtual «тип» «имя метода» ( параметры ) = 0;
```

Абстрактный класс – класс, который содержит хотя бы один чисто виртуальный метод.

Перегрузка функции

Перегрузка функции – это механизм, который позволяет двум родственным функциям иметь одинаковые имена.

Несколько функций могут иметь одинаковые имена, но при условии, что их параметры будут различными.

Это один из способов реализации полиморфизма.

Перегрузка функции (пример)

```
void f ( int i);          // один целочисленный параметр
void f ( int i, int j); // две целочисленных параметра
void f ( double k);       // один действительный параметр

int main()
{
   f ( 10 );      // вызов f ( int )
   f ( 10, 20 ); // вызов f ( int, int )
   f ( 12.23 );  // вызов f ( double )

   return 0;
}
```