acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . OOP-Kont :

Контейнеры.

Контейнер - это структура данных для хранения и манипуляции с однородными объектами.

Самые простые известные контейнеры - это массивы. Как правило для большого количесва данных используются динамические массивы.

Контейнеры могут быть устроены и более сложным образом. Можно организовать, например, контейнеры-списки, множества, хеши и др. В принципе, большинство ихз них уже реализованы в C++. Однако рассморим "ручную реализацию" одного из видов контейнеров: кольцевого буфера.

Кольцевым буфером будем считать односвязный список, в котором последний элемент ссылается на первый элемент списка.

Вообще любой контейнер должен предоставлять следующие возможности для работы с ним:

1. конструирование контейнера
2. добавление элементов в контейнер
3. удаление элементов по заданному критерию
4. последовательный доступ к элементам
5. доступ к элементам по заданному критерию

Доступ к элементам обычно организовывается с помощью вспомогательного объекта, называемого **итератор**. Итератор должен иметь доступ к внутренним элементам класса-контейнера, поэтому он либо объявляется классом-другом контейнера, либо является внутренним доступным объектом класса-контейнера.

В примере контейнер **CBuff** содержит внутренний класс **Elem**. Этот класс доступен только для методов класса **CBuff** и друзей класса.

```
#include <cstdlib>
#include <iostream>


using namespace std;

//Класс Obj - элементы этого класса необходимо поместить в буфер
class Obj{
   int n;

public:
     Obj();
     Obj(int);
     void print();
};

// Класс-контейнер для элементов Obj
class CBuff{

/*
  Внутренний класс Elem,  описывающий каждый узел
  кольцевого буфера
*/
  class Elem{
  public:

// Объекты, помещаемые в буфер
    Obj n;
```

Поиск
Поиск
Раздел «Язык Си»
Главная
Зачем учить С?
Определения
Инструменты:
Поиск
Изменения
Index
Статистика
Разделы
Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C
Logon>>

```
// Указатель на следующий элемент буфера
    Elem *next;

  };

// Два указателя на кольцевой список: на первый элемент
// и  на последний
  Elem *frst,*last;

// Общее количество элементов
  int kolvo;

 public:

// Конструктор буфера
        CBuff();

/*
  Добавление элемента в буфер.
  Добвление будем производить после последнего элемента
*/
       void add(Obj);

/*
  Получить указатель на первый элемент кольцевого списка.
  Заметим, что возвращаемый тип данных (указатель на Elem)
  может быть доступен только функциям или друзьям класса CBuff
*/
       Elem* begin();

// Получить указатель на последний элемент списка
        Elem* end();

// Печать всего списка (для отладки)
        void print();

/*
  Класс Iterator - внутренний класс (НЕ ОБЪЕКТ!!) класса CBuff.
  Находится в области public, значит может быть доступен для
  конструирования объекта и доступа к его методам.
  Класс Iterator:
  1. получает указатель на конкретный элемент списка,
     состоящего из узлов типа Elem
  2. перемещается к следующему элементу в списке
  3. возвращает указатель на объект Obj, помещенный в буфер
  4. позволяет выполнить операцию "разыменовывания" для
     указателей на объект
  5. позволяет выпольнить сранение ("нет, это не он") для решения
     достигнут ли нужный элемент при просмотре списка
*/
        class Iterator{

 // Указатель на элемет списка
            CBuff::Elem *point;
            int idx; // для проверки конца обхода

          public:

// Конструктор
            Iterator();

// Оператор присваивание. В качестве параметра указатель на Elem
            Iterator* operator=(Elem*);

// Перемещение к следующему элементу в списке
            Iterator* operator++(int);

// Возвращение указателя на объект, помещенный в список
            Obj* operator->();
```

```
// Получение доступа к самому объекту. Оператор "разыменовывания"
            Obj operator*();

// Сравнение двух элементов ("нет, это не он")
            int operator!=(Elem*);

        };

};

//_____________________________________________________________________
// Реализация класса Obj
//_____________________________________________________________________

Obj::Obj(){
  n=0;
};
Obj::Obj(int a){
    n=a;
cout<<"obj param:"<<n<<endl;
};

void Obj::print(){
    cout<<n<<endl;
};

//_____________________________________________________________________
// Реализация класса CBuff
//_____________________________________________________________________

// Коструктор. Предполагаем, что вначале буфер пуст.
CBuff::CBuff(){
     frst=0;
     kolvo=0;
};

// Добавление элемента
void CBuff::add(Obj z){
    struct Elem *new_p;
    kolvo++;

// Выделяем память под новый элемент
    new_p = new Elem;

// Заполняем ее полезной информацией
    new_p->n=z;
    new_p->nm=kolvo;

// Проверяем, пуст буфер или нет
    if (!frst){

//  Если пуст - это элемент становится первым
//  и последним (ссылка на него же)
        frst=new_p;
        frst->next=frst;
        last=frst;

    } else{

// Если не пуст
// ему присваивается ссылка на первый элемент
         new_p->next=last->next;

// последний элемент теперь ссылается на новый
         last->next=new_p;

//  новый становится последним
         last=new_p;
```

```
    }

};

// Получение указателя на первый элемент
CBuff::Elem* CBuff::begin(){
    kolvo=0;
   return frst;
};

// Получение указателя на последний элемент
CBuff::Elem* CBuff::end(){
   return last;
};


// Печать всего буфера
void CBuff::print(){
  Elem *p=frst;
// Печатаем, если список не пуст
  if (frst!=0){
      (frst->n).print();
      p=frst->next;

     while (p!=frst){
         (p->n).print();
         p=p->next;
     };
  }
}

//_____________________________________________________________________
// Реализация класса CBuff::Iterator
//_____________________________________________________________________

// Конструктор. Указатель вначале 0
CBuff::Iterator::Iterator(){
    point=0;
};

// Операторприсваивания.
CBuff::Iterator* CBuff::Iterator::operator=(CBuff::Elem* z){
     idx=1;
     point=z;
     return this;
};

// Оператор ++
CBuff::Iterator* CBuff::Iterator::operator++(int){

// Если список не пуст, перемщаем указатель по списку
        if(point)
        point=point->next;
        return this;
};

// Оператор "разыменовывания".
// Обычно применаяется с try...catch, так как
//  при пустом списке объект выдать мы не можем
Obj CBuff::Iterator::operator*(){
     if(point)
      return point->n;
     else
       return 0;
};

// Оператор "укаатель на объект"
Obj* CBuff::Iterator::operator->(){
```

```
        return &(point->n);
};

// Оператор "неравенство"
/*
 Так как буфер - кольцевой, то начало и конец списка совпадают
 к тому же могут быть удалены некоторые элементы и начальный
 элемент может изменится
*/
int CBuff::Iterator::operator!=(CBuff::Elem* check){
 //   cout<<"check:"<<kolvo<<endl;
    bool ret;
//    cout<<"ret:"<<ret<<endl;

// Проверка в первый ли раз указатель "смотрит"
// на головной элемент
    ret=(idx);

// kolvo=0, значит перехода на следующий элемент не будет
    if(point==check)
        idx--;
    return ret;
};

// Тестирование буфера
int main(int argc, char *argv[])
{
    char c;
    CBuff a; //  список пуст
    a.add(7); // добавление элементов
    a.add(8);
    a.add(5);
    a.print();

// Создание итератора
    CBuff::Iterator t;
    cout<<"итератор\n";

// связывание итератора (это другой объект!!!) с буфером
    t=a.begin();
// t-> - указывает на первый элемент списка и, далее, вызов print()
    t->print();
     t++;

// Пример использования итератора в цикле
    for(t = a.begin(); t != a.end(); t++){
        t->print();
        (*t).print();
    }


    return 0;
}
```

Задачи

Задача 1.

Отладить, запустить и проверить эту программу.

Задача 2.

Память калькулятора состоит из 2K чисел типа float. Регистры x (k-тая ячейка) - и y (k-1 ячейка). Первые K ячеек - нумерованные регистры. Нумерация начинается с x. Последние k ячеек - кольцевой буфер, который можно вращать как вправо, так и влево. Значения всех ячеек можно посмотреть только если оно передано в x.

Описание работы с памятью в разделе "С для кофейником" "Калькулятор Электроника В3-21"

2.1 Написать интерфейс класса для работы с такой памятью. В классе должен быть объявлен итератор, который позволяет перемещаться по кольцевому буферу в обе стороны. Для класса переопределить операторы не скобки для доступа к нумерованные регистрам.

2.2 Реализовать и проверить работу функций класса

Задача 3.

Дана картинка в виде псевдографического файла. На белом фоне (белые клетки - ., черные - *) нарисованы черные прямоугольники, которые не пересекаются и не касаются ни в каких точках. Реализовать контейнер для работы с этими прямоугольниками. Все прямоугольники должны быть раскрашены в разные цвета и добавлены в контейнер прямоугольников. Требования к контейнеру:

1. конструирование контейнера
2. добавление прямоугольника в контейнер
3. удаление прямоугольника по заданному критерию
4. последовательный доступ к элементам (перегрузка операторных скобок)
5. итератор, который работает с множеством РАЗЛИЧНЫХ прямоугольников (предоставляет доступ к следующему или предыдущему ДРУГОМУ прямоугольнику).

-- TatyanaOvsyannikova2011 - 22 Sep 2015 (Замечания и вопросы присылайте по адресу: tat@jscc.ru

Attachment	Action	Size	Date	Who	Comment
image.tar	manage	60.0 K	15 Mar 2018 - 10:13	TatyanaOvsyannikova2011	
comp1	manage	0.1 K	15 Mar 2018 - 10:07	TatyanaOvsyannikova2011	
wxwidgets-install.sh	manage	0.3 K	16 Mar 2017 - 16:13	TatyanaOvsyannikova2011	
comp-mac	manage	0.1 K	16 Mar 2017 - 16:14	TatyanaOvsyannikova2011	