acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . OOP-KontSM :

Пример контейнера для игры в крестики-нолики

Мы будем рассматривать поле для игры в крестики-нолики на доске ***n x n***.

Реализуем класс, позволяющий работать с таким полем. В этом классе предусмотрем создание специального объекта – ***итератора***. Итератор позволяет иметь доступ к отдельным элементам сущностей, хранимых в контйнере так как нам будет удобно для решения конкретной задачи. Итератор – часть класса-контейнера. Но создается как ОТДЕЛЬНЫЙ объект, поэтому его нужно СВЯЗАТЬ с конкретным объектом-контейнером.

Для одного контейнера можно одновременно импользовать несколько различных иетраторов.

Заголовочный файл *tictac.h*

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <iostream>
#include <cstdlib>
#include <fstream>

using namespace std;

// Будем считать крестик(X) - 1, а нолик(0) - -1
class TicTac{
  string s;
  int len; // размер поля
  int *a;  // указатель на область памяти  для игрового поля
public:
// инициализирующий конструктор (нужен размер)
   TicTac(int);
// Деструктор для удаления выделенной памяти под поле
   ~TicTac();
// доступ к ряду поля по его номеру
   int* operator[](int);
// Класс Iteraror является классом в классе TicTac
// Функции Iterator имеют доступ ко всем атрибутам и функциям
// класса TicTac
   class Iterator{
      int start; // для подсчета количества итераций
      int ln;   // размер поля
      int current;
// указатели на начало и конец области памяти для поля
      int *begin, *end;
     public:
// Конструктору нужен конкретный контейнер
       Iterator(TicTac&);
// Оператор присваивания (присваивается номер элемента в поле)
       Iterator& operator=(int);
// обход ряда поля по циклу: (0,0)->(0,1)->(0,1)->(0,0)
```

```
        Iterator& operator++(int);
// обход столбца по циклу
        Iterator& operator++();
// обход левой диагонали по циклу
        Iterator& operator--(int);
// обход правой диагонали по циклу
        Iterator& operator--();
// досуп к значению элемента поля,
// на который указывает итератор
        int operator*();
// оператор сравнения с числом (для циклов)
        int operator<(int);
    };
};
```

Реализация *tictac.cpp*

```
#include "tictac.h"

TicTac::TicTac(int n){
// установить размер поля
    len = n;
// выделить память из кучи для поля
    a = new int[len*len];
// очистить поле.
// Но лучше использовать функцию bzero()
    for(int i=0;i<len*len;i++){
       a[i]=0;
    }
};

// деструктор
TicTac::~TicTac(){
//  дописать код для реализации деструктора
};

// оператор [] возвращает УКАЗАТЕЛЬ на начало СТРОКИ поля
// по номеру этой строки
int* TicTac::operator[](int n){
    return a + (n)*len;
};

//===== Реализация класса TicTac::Iterator =============
TicTac::Iterator::Iterator(TicTac& t){
// указатель на начало памяти для поля
     begin = t.a;
// размер поля
     ln = t.len;
// указатель на конец памяти для поля
     end = t.a + ln*ln - 1;

     start = 0;
};

// Присваивание. Присваиваем НОМЕР элемента памяти
TicTac::Iterator& TicTac::Iterator::operator=(int n){
     start= 0; // для цикла ставим 0
     current = n-1;
     return *this;
};

// Изменяем указатель на номер элемента в текущей
// строке по циклу
TicTac::Iterator& TicTac::Iterator::operator++(int){
     int n = current / ln;
     int k = (current + 1) % ln;
     current = n * ln + k;
     start++;
     return *this;
```

```
};

// Изменяем указатель на номер элемента в текущем
// столбце по циклу
TicTac::Iterator& TicTac::Iterator::operator++(){
     int n = current % ln;
     int k = current / ln;
     cout<<"n="<<n<<" k="<<k<<endl;
     current = ((k+1)*ln)%(ln*ln) + n;
      cout<<"Vcur="<<current<<endl;
     start++;
     return *this;
};

// сравниваем с числом
int TicTac::Iterator::operator<(int n ){
     if(start<n) return 1;
     return 0;
};
// возвращаем значение элемента, на
// указывает итератор
int TicTac::Iterator::operator*(){
    return *(begin+current);
};
```

Использование классов *tictac.h*

```
#include "tictac.h"

int main(){
  TicTac pole(3);
  pole[0][0] = 1;
  pole[0][1] = 2;
  pole[0][2] = 3;
  pole[1][0] = 4;
  pole[1][1] = 5;
  pole[1][2] = 6;
  pole[2][0] = 7;
  pole[2][1] = 8;
  pole[2][2] = 9;
  cout<<pole[1][1]<<endl;
  cout<<pole[2][2]<<endl;
// объявление итератора и связь его с полем
  TicTac::Iterator it(pole);
// указываем на элемет 5
  it = 5;
// печатаем значение элемента, на
// который указывает итератор
  cout<<"ti5:"<<*it<<endl;
// обходим ряд (в которос 5 элемент) по циклу три раза
  for(it = 5; it< 3 ; ++it){
    cout<<(int)(*it)<<" ";
  }
  cout<<endl;
 }
```

Модернизация контейнера для совместной работы с разделяемой памятью

Для разделения доступа к ресурсам могут использоваться СЕМАФОРЫ.

Семафор в UNIX – системное устройство. Все операции, связанные с изменением сстояния семафора – АТОМАРНЫЕ. То есть, выполнение программы не может быть прервано и отложено пока операция с семафором не будет полностью завершена.

Каждый семафор имеет некоторое значение, которое при создании семафора сразу утанавливается в 0.

Над семафором можно выполнять следующие операции:

1. Увеличить значение семафора
2. Дождаться когда значение семафора станет равным 0
3. Дождаться когда значение семафора станет равным числу N

При этом понятно, что первая операция безусловная, а две последние требуют проверки значения семафора.

Рассмотрим пример работы с семафорами для игры в крестики-нолики.

Имеем два игрока, которые используют одно и тоже поле. Необходимо обеспечить очередность ходов игроков.

Мы не знаем какой из игроков первый предложит игру.

Самое первое действие, которое необходимо сделатть при организации игры – это обеспечить чтобы первый игрок дождался второго и не начинал игру один.

Для этого нужно использовать семафор номер 1

Далее доступ к полю (будем блокировать поле целиком) будем обеспечивать семафором номер 0.

Заголовочный файл для поля с семафорами *tictacSM.h*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <iostream>
#include <cstdlib>
#include <fstream>

#define PERM 0666

using namespace std;


class TicTac{
// ключ для создания семафоров и разделяемой памяти
  key_t key;
// описание операций с семафором (ожидание партнера)
// процесс - в ожидание
  struct sembuf waitYou[1];
// первое поле - номер семафора (1)
// второе - операция с семафором. Здесь:
// дождаться когда занчение семафора будет равно 1
// ={ 1,-1,0};

// описание операций с семафором (партнер пришел)
// партнер может действовать
  struct sembuf ImHere[1];
// первое поле - номер семафора (1)
// второе - операция с семафором. Здесь:
// увеличить значение семафора на 1
//={ 1,1,0};

// описание операций с семафором (записает поле)
// используется 2 операции
  struct sembuf myTurn[2];
//  Значение семафора 0 - ресурс свободен, 1 - занят
// Первая операция - дождаться когда семафор будет 0
// вторая операция - ставим семафор в 1, то есть запираем ресурс
// = {
//    0,0,0,
//    0,1,0 };
```

```
// описание операций с семафором (записает поле)
// используется 1 операция
struct sembuf canWork[1];
//  Дождаться когда семафор будет 1 и обнулить его
// = { 0,-1,0 };

  int lng;
// очередность определяется по тому кто создал очередь семафоров
  int range;
  char stop;
  int set;
// дескрипторы для семафоров и разделяемой памяти
  int semid,shmid;
// указатель на разделяемую память
  int *msgptr;
  string s;
  int len; // размер поля

public:
// Для ключа нужны имя файла и символ
   TicTac(string, char, int);
// необходимо удалять семафоры после работы
   ~TicTac();
   int* operator[](int);
// запирание ресурса
   void myWork();
// освобождение ресурса
   void youWork();
// получить очередность
   int getRange();
//  печать поля
   void print();
// Все операции с полем выполняются только если ресурс "захвачен"
   class Iterator{
      int start,ln;
      int current,*begin, *end;
     public:
       Iterator(TicTac&);
       Iterator& operator=(int);
       Iterator& operator++(int);
       Iterator& operator++();
       Iterator& operator--(int);
       Iterator& operator--();
       int operator*();
       int operator<(int);
   };
};
```

Реализация *tictacSM.cpp*

```
#include "tictacSM.h"

TicTac::TicTac(string s, char c, int n){

   len = n; // размер поля
// получим ключ
   if ((key=ftok(s.c_str(),c))<0){
      printf("Can't get key\n");
      exit(1);
   }
// Установим операции для семафоров:

// Захватить ресурс
// Эти операции будут выполняться сразу обе
   myTurn[0].sem_num = 0; // для семафора номер 0
   myTurn[0].sem_op = 0; //  дождаться 0
   myTurn[0].sem_flg = 0;
   myTurn[1].sem_num = 0; // для семафора номер 0
   myTurn[1].sem_op = 1; // увеличить значение на 1
```

```
   myTurn[1].sem_flg = 0;

// Освободить ресурс
   canWork[0].sem_num = 0; // для семафора номер 0
   canWork[0].sem_op = -1; //  дождаться 1 и поставить значение семафора в 0
   canWork[0].sem_flg = 0;

// Ожидание партнера
   waitYou[0].sem_num = 1; // для семафора номер 1
   waitYou[0].sem_op = -1; // дождаться 1 и поставить значение семафора в 0
   waitYou[0].sem_flg = 0;

// Уведомление "Я пришел"
   ImHere[0].sem_num = 1; // для семафора номер 1
   ImHere[0].sem_op = 1; // увеличить значение на 1
   ImHere[0].sem_flg = 0;

// Создать разделяемую память
   shmid = shmget(key, sizeof(int)*len*len, PERM|IPC_CREAT);
   switch (errno){
      case EEXIST:{
                   if((shmid = shmget(key, sizeof(int)*len*len, 0))<0){
                  printf("Can't create shared mem\n");
                  exit(1);
               }

                  break;
                }
      case 0:    {

                  break;
                }
      default:   {
                  printf("Can't create shared mem\n");
                  exit(1);
                }

   }
//  получить указатель на разделяемую паамять
   if ((msgptr = (int*)shmat(shmid, 0, 0))<0){
      perror(":(");
      exit(1);
   }
//  создать семафоры (2 семафора)
   semid = semget(key,2,PERM|IPC_CREAT|IPC_EXCL);
   switch (errno){
      case EEXIST:{
                   if ((semid = semget(key, 2, PERM))<0){
                       perror(":(");
                        exit(1);
                   }
                  cout<<"Семафор уже есть\n";
                   range = 1;
                   break;
                }
      case 0:    {
                  range = 0;
                  break;
                }
      default:   {
                  printf("Can't create sema\n");
                  exit(1);
                }

   }
   set = 0;
// получить свою очередь
   if(range == 0){
    cout<<"Ваш знак: 0\n";
```

```
    cout<<"Ставлю семафор на встречу в -1\n";

// semop - функция операции с семафорами
// тот, кто создал очередь семафоров ждет партнера
     if (semop(semid, &waitYou[0], 1)<0){
         printf(":(\n");
         exit(1);
     }
   }

   if(range == 1 ){

     cout<<"Ваш знак: X\n";
     cout<<"Чищу память\n";
     bzero(msgptr, sizeof(int)*len*len);
     cout<<"Я пришел\n";
// Уведомление "Я пришел"
     if (semop(semid, &ImHere[0], 1)<0){
      printf(":(\n");
      exit(1);
     }

   }

};
// Получить свою очередь
int TicTac::getRange(){
  return range;
};

// Удаление семафорров и разделяемой памяти
TicTac::~TicTac(){
   if (shmctl(shmid,IPC_RMID,0)<0){
      printf(":(\n");
   }

   if (semctl(semid,2,IPC_RMID)<0){
     printf(":(\n");
   }
};

void TicTac::print(){

  for(int j = 0;j<3;j++)
     {
      for(int i = 0;i<3;i++)
//  разделяемая память может быть использована как обычный массив
        cout << msgptr[j*3 + i]<<" ";

      cout<<endl;
     }
};

int* TicTac::operator[](int n){

    return msgptr + (n)*len;
};
// Запираем ресурс
void TicTac::myWork(){
   cout<<"Запираем ресурс\n";
// указываем, что выполняться будут сразу 2 операции (атомарно)
    if (semop( semid, &myTurn[0], 2)<0){
       printf(":(\n");
       exit(1);
    }
};

// освобождаем ресурс
void TicTac::youWork(){
```

```
    cout<<"Освобождаем ресурс\n";
    if (semop(semid, &canWork[0], 1)<0){
        printf(":(\n");
        exit(1);
   }      cout<<"Vcur="<<current<<endl;

};

TicTac::Iterator::Iterator(TicTac& t){
     begin = t.msgptr;
     ln = t.len;
     end = t.msgptr + ln*ln - 1;
     start = 0;
};

TicTac::Iterator& TicTac::Iterator::operator=(int n){
     start= 0;
     current = n-1;
     return *this;
};

TicTac::Iterator& TicTac::Iterator::operator++(int){
     int n = current / ln;
     int k = (current + 1) % ln;
     current = n * ln + k;
     cout<<"cur="<<current<<endl;
     start++;
     return *this;
};

TicTac::Iterator& TicTac::Iterator::operator++(){
     int n = current % ln;
     int k = current / ln;
     current = ((k+1)*ln)%(ln*ln) + n;
     start++;
     return *this;
};


int TicTac::Iterator::operator<(int n ){
     if(start < n) return 1;
     return 0;
};
int TicTac::Iterator::operator*(){
    return *(begin+current);
};
```

=Использование = ***tic***

```
#include "tictacSM.h"

int main(){
  TicTac pole("tic",'a',3);
  string s;
  int range = pole.getRange();

  int sgn = (range==1)?1:-1;
  int x, y;
// обеспечиваем очередность 3 ходов с каждой стороны
  for(int k= 0; k<3;k++){

      pole.myWork();
      pole.print();
      cout<<"Ваш ход:";
      cin>>y>>x;
     pole[y-1][x-1]=sgn;
      cout<<"ok\n";
     pole.print();
```

```
      pole.youWork();
  }
  return 0;
 }
```

Задачи

Задача 1.

Дописать функции итератора для класса TicTac? (с расширяемой памятью)

Задача 2.

Полностью реализовать игру в крестики нолики.

Задача 3.

В ящике летает шарик по прямой. Вектор движения шарика перпендикулярен стенкам ящика. Начальная скорость шарика - **V**. Положение шарика рассматривается раз в 3 секунды.

Написать две программы: первая вычисляет движения шарика и записывает из в разделяемую память, вторая - выясняет есть ли периодичность в движении шарика. Если периодичность есть, то в файл здля рассмотрения аписывается только период, если нет, все данные пока программа не закончит свою работу. Придумать и описать классы для реализации процесса.

-- TatyanaOvsyannikova2011 - 16 Nov 2016