Упражнение 7 - Преговор

Задачи за самостоятелна работа:

1. *Разгледайте Exam1.cpp. Пуснете програмата в средата на VC++. Обяснете си обектният модел на прозореца, който е реализиран в примера. Какво е предназначението на връщана стойност псевдоним към класа, използван в повечето методи? Как се реализира?*
2. *Създаите клас за работа с низове и предефинирайте операциите <<, >>, ==, !, +, += и = за него. Сравнете решението си с Exam2.cpp. Използвахте ли копи-конструктор и защо?*
3. *Разгледайте Exam3.cpp. Обяснете си наследяването и виртуалните методи. Какво ще се разпечати на екрана? Пуснете програмата и проверете правилността на отговорите си.*

Приложение

Exam1.cpp

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
const char BELL='\007';
class Screen {
      short height,width;
      char *cursor,*screen;
      Screen & home() { move(0,0); return *this; }
      void checkRange(int,int);
      row();
      remSpace();
public:
      Screen & move(int,int);
      Screen & reSize(int,int,char='#');
      Screen & lineX(int,int,int,char);
      Screen & display();
      Screen & down();
      Screen & forward();
      Screen & set(char *);
      Screen & set(char);
      isEqual(Screen &);
      void copy(Screen &);
      Screen (int=8,int=40,char='#');
      ~Screen();
};
void lineY(Screen & s,int row,int col,int len,char ch)
{     s.move(row,col);
      for(int i=0;i<len;i++) s.set(ch).down();
```

```
      }
      void Screen::checkRange(int row,int col)
      {     if (row<1 || row>height || col<1 || col>width)
            {     cerr<<"Coordinate  ("<<row<<','<<col<<")  is  out  of
range.\n";
               exit(1);
            }
      }
      inline Screen::row()
      {     int pos = cursor-screen+1;
            return ((pos+width-1)%width)+1;
      }
      inline Screen::remSpace()
      {     int sz = width*height;
            return (screen+sz-cursor-1);
      }
      Screen & Screen::move(int r,int c)
      {     checkRange(r,c);
            int row = (r-1)*width;
            cursor = screen+row+c-1;
            return *this;
      }
      Screen & Screen::reSize(int h,int w,char bk)
      {   Screen *ps = new Screen(h,w,bk);
            char *pNew = ps->screen;
            char *pOld = screen;
            while (*pOld && *pNew) *pNew++ = *pOld++;
            this->Screen::~Screen();
            *this = *ps;
            return *this;
      }
      Screen & Screen::lineX(int row,int col,int len,char ch)
      {   move(row,col);
            for(int i=0;i<len;i++) set(ch).forward();
            return *this;
      }
      Screen & Screen::display()
      {   char *p;
            for(int i=0;i<height;i++)
            {  cout<<'\n';
               int offset = width*i;
               for(int j=0;j<width;j++)
               {  p=screen+offset+j;
                     cout.put(*p);
               }
            }
            return *this;
      }
      Screen & Screen::down()
      {     if(row()>height)  cout.put(BELL);
            else  cursor += width;
            return *this;
      }
      Screen & Screen::forward()
      {     ++cursor;
            if(*cursor=='\0')  home();
            return *this;
```

```
}
Screen & Screen::set(char *s)
{     int space=remSpace();
      int len=strlen(s);
      if (space<len)
      {  cerr<<"String will be truncated from position "<<space<<
                  ". New string length will be "<<len<<".\n";
         len=space;
      }
      for (int i=0;i<len;i++)  *cursor++=*s++;
      return *this;
}
Screen & Screen::set(char ch)
{     if(ch=='\0')
        cerr<<"Symbol NULL is ignored.\n";
      else  *cursor=ch;
      return *this;
}
Screen::isEqual(Screen & s)
{   if(width!=s.width || height!=s.height)  return 0;
      char *p=screen,*q=s.screen;
      if(p==q)  return 1;
      while(*p && *q && *p++==*q++)  ;
      if(*p || *q)  return 0;
      return 1;
}
void Screen::copy(Screen & s)
{     delete screen;
      height=s.height;
      width=s.width;
      screen=cursor=new char[height*width+1];
      if(screen==NULL)
      {  cerr<<"Out of memory.\n";
         this->Screen::~Screen();
         exit(1);
      }
      strcpy(screen,s.screen);
}
Screen::Screen(int high,int widt,char bkgr)
{     int sz=high*widt;
      height=high;
      width=widt;
      cursor=screen=new char[sz+1];
      if(screen==NULL)
      {  cerr<<"Out of memory.\n";
         this->Screen::~Screen();
         exit(1);
      }
      char *ptr=screen,*endptr=screen+sz;
      while(ptr!=endptr)  *ptr++=bkgr;
      *ptr='\0';
}
Screen::~Screen()
{     delete screen;
}

void main(void)
```

```
{     Screen X(3,3),Y(3,3);
//    clrscr();
      cout<<"Objects are equal "<<X.isEqual(Y)<<'\n';
      Y.reSize(6,6);
      cout<<"Objects are not equal "<<X.isEqual(Y)<<'\n';
      lineY(Y,1,1,6,'*'); lineY(Y,1,6,6,'*');
      Y.lineX(1,2,4,'*').lineX(6,2,4,'*').move(3,3);
      Y.set("hi").lineX(4,3,2,'^').display();
      X.reSize(6,6);
      cout<<"\n\nObjects are not equal "<<X.isEqual(Y)<<'\n';
      X.copy(Y);
      cout<<"Objects are equal "<<X.isEqual(Y)<<'\n';
      cin.get();
}
```

Exam2.cpp

```
#include <string.h>
#include <iostream.h>

const SIZE=100;
class String {
      friend ostream& operator << (ostream&,String);
      friend istream& operator >> (istream&,String&);
      friend String operator + (String,String);
public:
      String(void) {len=0;};
      String(int);
      String(char *);
      String(String&);
      ~String(void);
      int operator ! (void);
      int operator == (String);
      String& operator += (String);
      String& operator = (String);
      String& operator = (char *);
private:
      char *str;
      int len;
};

String::String(int ln)
{
      len=ln;
      str=new char [len+1];
      str[0]='\0';
}

String::String(char *s)
{
      len=strlen(s);
      str=new char[len+1];
      strcpy(str,s);
}

String::String(String &st)
```

```
{
      len=st.len;
      str=new char[len+1];
      strcpy(str,st.str);
}

String::~String(void)
{ if (len)
      delete str;  }

int String::operator ! (void)
{     return (len==0);  }

int String::operator == (String st)
{     return (strcmp(str,st.str)==0);  }

String& String::operator += (String st)
{
      char *s = new char[len+1];
      strcpy(s,str);
      len+=st.len;
      String::~String();
      str=new char[len+1];
      strcpy(str,s);
      strcat(str,st.str);
      return *this;
}

String& String::operator = (String st)
{
      String::~String();
      len=st.len;
      str=new char[len+1];
      strcpy(str,st.str);
      return *this;
}

String& String::operator = (char *s)
{
      String::~String();
      len=strlen(s);
      str=new char[len+1];
      strcpy(str,s);
      return *this;
}


String operator + (String st1,String st2)
{
      String s=st1;
      s += st2;
      return s;
}

ostream& operator << (ostream &os,String st)
{     return (os<<st.str);  };
```

```
      istream& operator >> (istream &is,String &st)
      {
            char inBuf[SIZE];
            is>>inBuf;
            st=inBuf;
            return is;
      }

      void main(void)
      {
            String s1,s2,s3;
            cin>>s1>>s2;
            cout<<s1<<'\t'<<s2<<'\n';
            s3 = s1+s2;
            cout<<s3<<'\n';
            s3 = s1;
            s3 += s2;
            cout<<s3<<'\n';
      }
```

Exam3.cpp

```
#include <string.h>
#include <iostream.h>
enum ZooLocs{ZOOANIMAL,BEAR,PANDA};
static char *locTable[]={
"Whole zoo area",
"North B1: brown area",
"East B1,P area" };
class ZooAnimal {
      friend void print(ZooAnimal *);
public:
      ZooAnimal(char *s="ZooAnimal");
      virtual ~ZooAnimal() { delete name; }
      void link(ZooAnimal *);
      virtual void print();
      virtual void isA();
protected:
      char *name;
      ZooAnimal *next;
};
class Bear:public ZooAnimal {
public:
      Bear(char *s="Bear",ZooLocs loc=BEAR,char *sci="Ursidae");
      ~Bear() { delete sciName; }
      void print();
      void isA();
protected:
      char *sciName;
      ZooLocs ZooArea;
};
class Panda:public Bear {
public:
      Panda(char     *,int,char     *s="Panda",char     *sci="Ailuropoda
Melaoleuca",
```

```
            ZooLocs loc=PANDA);
      ~Panda() { delete indName; }
      void print();
      void isA();
protected:
      char *indName;
      int cell;
};
ZooAnimal::ZooAnimal(char *s):next(0) {
      name=new char[strlen(s)+1];
      strcpy(name,s);
}
void ZooAnimal::link(ZooAnimal *za)
{     za->next=next;
      next=za;
}
void ZooAnimal::isA()
{     cout<<"Animal name: "<<name<<'\n';
}
void ZooAnimal::print()
{     isA();
}
Bear::Bear(char *s,ZooLocs loc,char *sci):ZooAnimal(s),ZooArea(loc)
{       sciName=new char[strlen(sci)+1];
      strcpy(sciName,sci);
}
void Bear::isA()
{     ZooAnimal::isA();
      cout<<"\tSname: \t"<<sciName<<'\n';
}
void Bear::print()
{     ZooAnimal::print();
      cout<<"\tAdress: \t"<<locTable[ZooArea]<<'\n';
}
Panda::Panda(char *nm,int room,char *s,char *sci,ZooLocs loc):
           Bear(s,loc,sci),cell(room) {
      indName=new char[strlen(nm)+1];
      strcpy(indName,nm);
};
void Panda::isA()
{     Bear::isA();
      cout<<"\tCall our friend: \t"<<indName<<'\n';
}
void Panda::print()
{     Bear::print();
      cout<<"\tCell No: \t"<<cell<<'\n';
}
void print(ZooAnimal *pz)
{     while(pz)
      {  pz->print();
         cout<<'\n';
         pz=pz->next;
      }
}

ZooAnimal *headPtr=0,circus("Circus animal");
Bear yogi("Little bear",BEAR,"ursus cartoonus");
```

```
Panda yinYang("Yin Yang",1001,"Big Panda"),
      rocky("Rocky",943,"Red Panda","Ailurus fulgena");

ZooAnimal * makelist(ZooAnimal *ptr)
{     ptr=&yinYang;
      ptr->link(&circus);
      ptr->link(&yogi);
      ptr->link(&rocky);
      return ptr;
}

void main()
{     cout<<"Virtual Function Example\n";
      headPtr=makelist(headPtr);
      print(headPtr);
}
```