Московский ордена Ленина, ордена Октябрьской Революции
и ордена Трудового Красного Знамени.
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н. Э. БАУМАНА

Факультет: Информатики и систем управления
Кафедра: Проектирование и технология производства электронной аппаратуры (ИУ4)

О Т Ч Е Т

По лабораторной работе №7
"Разбор выражений методом
рекурсивного спуска с оптимизацией
под сопроцессор 80x87"

По курсу: Программирование на языке C/C++

Студент: Афанасьев А.В. ИУ4-22
(фамилия, инициалы) (индекс группы)

Руководитель: Дединский И.Р.
(фамилия, инициалы)

отметка о защите	дата защиты

Москва
2002

Содержание

Техническое задание

Наименование разработки: Разбор выражений методом рекурсивного спуска с оптимизацией под сопроцессор 80x87.

Цель работы – оптимизировать алгоритм разбора выражений методом рекурсивного спуска под сопроцессор 80x87.

Решаемые задачи:

Преобразовать существующую программу для разбора математических выражений с учетом работы по алгоритму рекурсивного спуска согласно алфавиту, указанному в разделе «Математическое обеспечение» данного отчета.

Оптимизировать разбор выражения с учетом работы сопроцессора 80x87.

Этапы выполнения:

1. Исправление алфавита метода рекурсивного спуска (в предыдущих лабораторных работах была замечена ошибка в алфавите).
2. Оптимизация под работу с сопроцессором 80x87.

Форма отчетности:

По результатам разработки предоставляется: отчет (файл otchet.doc), исходный текст программы (файлы stack.h, stack.cpp, parser.h, parser.cpp, cmd_list.h, cmd_list.cpp function.cpp, function.h, my_types.h, nametable.h, nametable.cpp, makefile), исполняемый файл (parser.exe).

1. Математическое обеспечение

Алфавит языка, понимаемого программой:

+, -, /, *, (,), числа в пределах типа double, символьные строки, начинающиеся с латинской буквы и содержащие буквы латинского алфавита, цифры и знак подчеркивания.

Правила, по которым программа анализирует входное выражение:

!!!! алфавит немного неправильный !!!! надо переделать, а то бред

выражение -> слагаемое ост.выражение

ост.выражение -> + выражение

ост.выражение -> - слагаемое ост.выражение **

ост.выражение ->

слагаемое -> множимое ост.слагаемое

*ост.слагаемое -> * слагаемое*

*ост.слагаемое -> * множимое ост.слагаемое* *

ост.слагаемое ->

множимое -> число

множимое -> (выражение)

Мнемоники команд сопроцессора

Арифметические операции

FABS	FADD/FIADD	FADDP
FCHS	FDIV/FIDIV	FDIVP
FDIVR/FIDIVR	FDIVRP	FMUL/FIMUL
FMULP	FPREM	FPREM1
FRNDINT	FSCALE	FSQRT
FSUB/FISUB	FSUBP	FSUBR/FISUBR
FSUBRP	FXTRACT	

Операции сравнения

FCOM/FICOM	FCOMP/FICOMP	FCOMPP
FSTSW/FNSTSW	FTST	FUCOM
FUCOMP	FUCOMPP	FXAM

Команды загрузки

FLD/FILD/FBLD	FLDCW	FLDENV
FRSTOR	FXCH	

Загрузка констант

FLD1	FLDL2E	FLDL2T

FLDLG2	FLDLN2	FLDPI
FLDZ		

Команды контроля процессором

FCLEX/FNCLEX	FDECSTP	FDISI/FNDISI
FENI/FNENI	FFREE	FINCSTP
FINIT/FNINIT	FLDCW	FNOP
FRSTOR	FSAVE/FNSAVE	FSETPM
FSTCW/FNSTCW	FSTENV/FNSTENV	FSTSW/FNSTSW
FWAIT		

Команды сохранения

FSAVE/FNSAVE	FST/FIST	FSTCW/FNSTCW
FSTENV/FNSTENV	FSTP/FISTP/FBSTP	FSTSW/FNSTSW

Трансцендентные операции

F2XM1	FCOS	FPATAN
FPREM	FPREM1	FPTAN
FSIN	FSINCOS	FYL2P1
FYL2X		

---

* исправления алфавита по сравнению с предыдущими лабораторными работами

1. Алгоритмического обеспечение

Обобщенная схема алгоритма программы

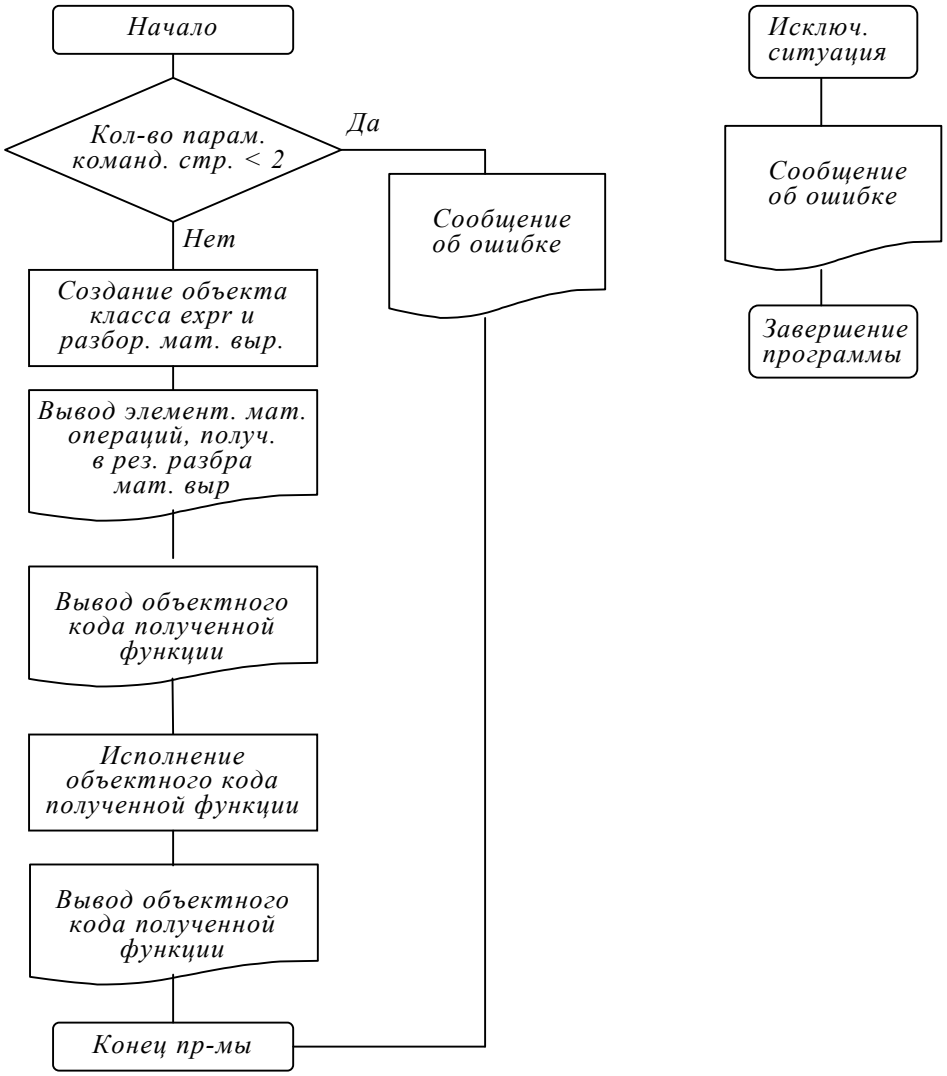
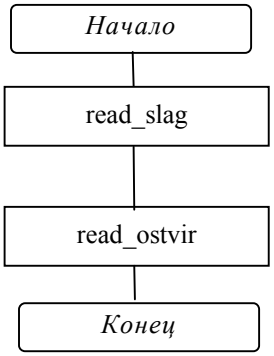
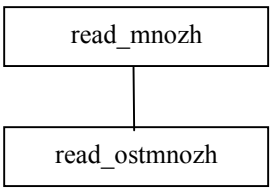


Схема анализатора математического выражения.

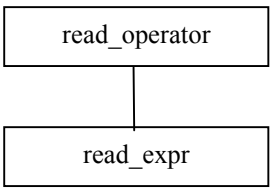
read_expr:



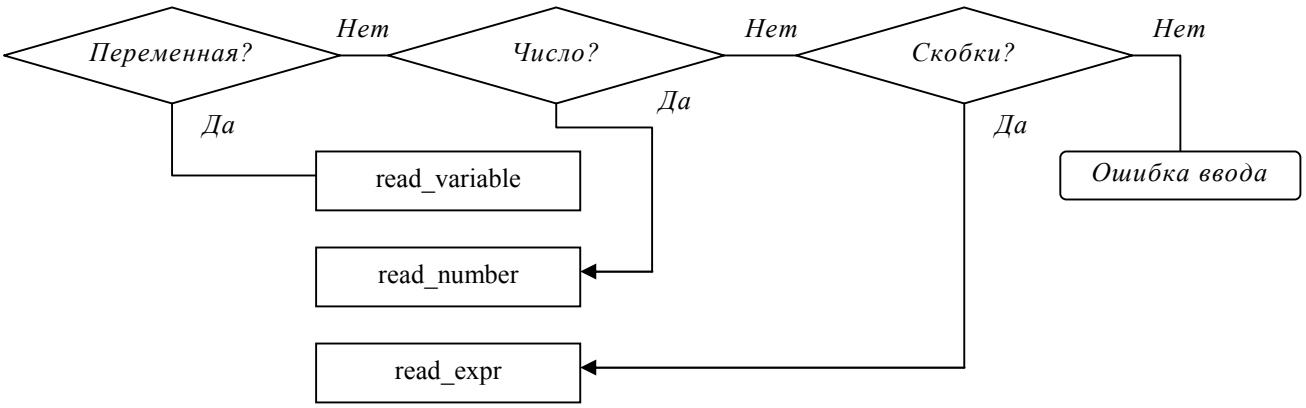
read_slag:



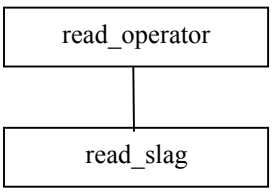
read_ostvir:



read_ostvir:



read_ostmnozh:



2. Лингвистическое обеспечение

В качестве лингвистического обеспечения использован язык C++.

3. Программное обеспечение

3.1. Использование библиотечных функций

Для реализации данной программы были использованы следующие стандартные библиотеки:

1. iostream.h - модуль с текстовыми потоками данных
2. ctype.h – модуль, объявления некоторых необходимых в программе макросов.
3. map – модуль, включающий определения STL шаблона класса map.
4. vector - модуль, включающий определения STL шаблона класса vector.

3.2. Формализация предметной области и допущения

В ходе реализации программы был сделан ряд допущений:

- выражение подающееся на вход анализатора берется из командной строки.
- в программе отключена возможность работы с переменными
- из-за особенностей работы сопроцессора, более 8-ми подряд идущих операций равного приоритета вызовут ошибку программы.

Таблица 1. Исходный листинг программы.

```
////////////////////////////////////////////////////////////////////////////
// cmd_list.h
//
// (c) 2002 Alexander Afanasyev


#ifndef MY_CMD_LIST_H
#define MY_CMD_LIST_H

#include <vector>
#include "my_types.h"
#include "nametable.h"

const int COPROCESSOR_STACK_MAX = 8;

const cmd_type PTR  = 128; /* 10000000 */

const cmd_type FLD  = 11;  /* 00001011 */
const cmd_type FLD7 = 12;

const cmd_type FST  = 13;
const cmd_type FST7 = 14;


const cmd_type FXCH = 1;

const cmd_type FADD = 3;   /* 00000011 */
const cmd_type FSUB = 4;   /* 00000100 */
const cmd_type FMUL = 5;   /* 00000101 */
const cmd_type FDIV = 6;   /* 00000110 */


class cmd_list
{
private:
    nametable myNameTable;
    vector<val_cmd_ptr_type> myList;

    int mycoprocessor_stack;
```

```
    int mystack_saved;
    int mycount_saved;
public:
    cmd_list( const nametable &table ): myNameTable(table),
                                                        mycoprocessor_stack(0),
                                        mystack_saved(0)        {};

    cmd_list( const cmd_list &cmds ):   myNameTable(cmds.myNameTable),
                                                        mycoprocessor_stack(0),
                                        mystack_saved(0)        {};

    cmd_list( ):mycoprocessor_stack(0),mystack_saved(0)         {};

    cmd_list( int offset ): myNameTable(offset),
                            mycoprocessor_stack(0),
                            mystack_saved(0)                    {};
//    cmd_list( const map<var_type,ptr_val_type> &m ): myMap(m),myOffset(0) {;};

    void add( cmd_type cmd, const ptr_type &ptr );
    void add( cmd_type cmd, const var_type &var );
    void add( cmd_type cmd );

    void pop( );

    int  offset( ) { return myNameTable.offset(); }
    int  realoffset( ) { return myNameTable.realoffset(); }

    void offset( int offs ) {  myNameTable.offset( offs ); }

    void operator+=( const cmd_list &cmds )
    {
        myNameTable.insert( cmds.myNameTable );
        myList.insert( myList.begin(),
                       cmds.myList.begin(),
                       cmds.myList.end() );
    }

    bool operator!( );

    //return - if stack full: false; if not: true
    bool push( const var_type &var );

    nametable& NameTable() { return myNameTable; };
    int NameTableSize() const { return myNameTable.size(); };

    nametable::const_iterator NameTableBegin() const
    {
        return myNameTable.begin();
    };
    nametable::const_iterator NameTableEnd() const
    {
        return myNameTable.end();
    };

public:
    typedef vector<val_cmd_ptr_type>::iterator iterator;
    typedef vector<val_cmd_ptr_type>::const_iterator const_iterator;

    iterator begin() { return myList.begin(); };
    const_iterator begin() const { return myList.begin(); };

    iterator end()   { return myList.end();   };
    const_iterator end() const   { return myList.end();   };
};

#endif

////////////////////////////////////////////////////////////////////////
// cmd_list.cpp
//
// (c) Alexander Afanasyev, 2002

#include "cmd_list.h"


void cmd_list::add( cmd_type cmd, const ptr_type &ptr )
{
    myList.push_back( cmd|PTR );
    myList.push_back( ptr );
}

void cmd_list::add( cmd_type cmd, const var_type &var )
{
    myList.push_back( cmd|PTR );
    myList.push_back( myNameTable[var] );
}

void cmd_list::add( cmd_type cmd )
{
    myList.push_back( cmd );
/*    if( mystack_saved>0 )
    {
        ptr_type loadptr=myNameTable["@@@@"+mystack_saved];
        mystack_saved--;
        add( FLD7, loadptr );
    }
    else
        mycoprocessor_stack--;*/
}
```

```
void cmd_list::pop( /*8cmd_type cmd*/ )
{
//    myList.push_back( cmd|PTR );
//    myList.push_back( myNameTable.pop() );
}

//return - if stack full: false; if not: true
bool cmd_list::push( const var_type &var )
{
    ptr_type dataptr=myNameTable.push( var );
    mycoprocessor_stack++;
    add( FLD, dataptr );

/*    if( mycoprocessor_stack<COPROCESSOR_STACK_MAX-1 )
        mycoprocessor_stack++;
    else
    {
        mystack_saved++;
        ptr_type saveptr=myNameTable["@@@@"+mystack_saved];
        add( FST7, saveptr );
    }*/

    return mycoprocessor_stack<COPROCESSOR_STACK_MAX-2;
}


bool cmd_list::operator!( )
{
    return ! (mycoprocessor_stack<COPROCESSOR_STACK_MAX-2);
}

////////////////////////////////////////////////////////////////////////
// function.h : header for my function generation/using class
//
// (c)2002 Alexander Afanasyev
////////////////////////////////////////////////////////////////////////
// comment: do not use it if you are unsure what this class is doing
//          it probaly can damage your computer, if compiler
//          is different than GNU C++ of Delorie DJGPP project

#ifndef MY_FUNCTION_H_01010101010101002927436463524173848585746362
#define MY_FUNCTION_H_01010101010101002927436463524173848585746362


#include "my_types.h"
#include "nametable.h"
#include "cmd_list.h"


const int FUNCTION_BUFFER_MAX=256;

// some coprocessor commands
const int FMULP_ST1_ST      = 0xDEC9;
const int FMULP_ST1_ST_SIZE = 2;

const int FDIVP_ST1_ST      = 0xDEF9;
const int FDIVP_ST1_ST_SIZE = 2;

const int FADDP_ST1_ST      = 0xDEC1;
const int FADDP_ST1_ST_SIZE = 2;

const int FSUBP_ST1_ST      = 0xDEE9;
const int FSUBP_ST1_ST_SIZE = 2;


const int FXCH_ST1_ST       = 0xc3;
const int FXCH_ST1_ST_SIZE  = 1;


class function
{
private:
    typedef double (*myfunc_type)(double);

    char *myFuncBuf; //if myFuncBuf != NULL <=> init function was run already
                         //and will throw an exception if somebody try to run
                     //function::init(...) one more time
    char *myPointer;

    void add( int  dw, int numbytes=1 ); //dw - cmd, numbutes - butes to write
    void add( void *s );
public:
    function( ): myFuncBuf( NULL ) { };
    function( cmd_list &cmds ): myFuncBuf( NULL ) { init( cmds ); }; //never used :)

    ~function( )
    {
        if( myFuncBuf ) delete [] myFuncBuf;
        myFuncBuf=NULL;
    }

    void init( cmd_list &cmds );

    double call( double );

public: //public exceptions
    class CannotRunInitTwice { };
};
```

```
#endif

//////////////////////////////////////////////////////////////////////
// function.cpp : my function generation/using class definition
//
// (c)2002 Alexander Afanasyev
//////////////////////////////////////////////////////////////////////
// comment: do not use it if you are unsure what this class is doing
//          it probaly can damage your computer, if compiler
//          is different than GNU C++ of Delorie DJGPP project

#include "function.h"

//#define UBRAT_NENUZHNIY_COUT

void function::add( void *ptr )
{
    *((unsigned *)(myPointer))=(unsigned)(ptr)/* & 0xFFFF*/;
    myPointer+=4;
}

void function::add( int dw, int bytesnum=1 )
{
    for( ; bytesnum>0; bytesnum-- )
    {
        *myPointer=( dw >> 8*(bytesnum-1) ) & 0xFF;
        *myPointer++;
    }
}


bool isnumber( const var_type &var )
{
    return ('0'<=var[0] && var[0]<='9') || var[0]=='.' ||
           var[0]=='-' || var[0]=='+';
}


void function::init( cmd_list &cmds )
{
    if( myFuncBuf ) throw CannotRunInitTwice();
    myFuncBuf=new char[FUNCTION_BUFFER_MAX];
    myPointer=myFuncBuf;

    //some preparing code
    add( 0x55 );     //push
    add( 0x89e5, 2); //mov %esp,%ebp

    //allocate place in memory for all our data and constants
    int datasize=cmds.NameTableSize();
    int dataoffset = FUNCTION_BUFFER_MAX - datasize*sizeof(double);
    if( dataoffset<FUNCTION_BUFFER_MAX/4 )
    {
        cerr << "So big expression will support in future program versions"
             << endl << "It is not a bug, it is a feature ;) " << endl;
        throw 1234;
    }

    int ds=dataoffset;
    for( nametable::const_iterator i =cmds.NameTableBegin();
                                   i!=cmds.NameTableEnd();
                                   i++ )
    {
#ifndef UBRAT_NENUZHNIY_COUT
        cout << i->first << ' ';
#endif
        if( isnumber(i->first) )
        { //it is a constant value and need to be initialized
#ifndef UBRAT_NENUZHNIY_COUT
            cout << i->second.myVal << endl;
#endif
            *(double*)
            ( myFuncBuf+ds+
             (i->second.myPtr-1)*sizeof(double) ) = i->second.myVal;
        }
        else
        {
             cerr << "Unfortunately, in this program temporaly variables "
                  << "are disabled. Please check for update. check();" << endl;
             throw 12345;
        }
    }


    for( cmd_list::const_iterator i=cmds.begin(); i<cmds.end(); i++ )
    {
        switch (i->myCmd & 31)
        {
        case FLD:
            add( 0xDD05, 2 );
            i++;
            add( myFuncBuf+ds+(i->myPtr-1)*sizeof(double) );
            break;
        case FLD7:
            throw 1;
            cout << "FLD ";
            if( i->myCmd & PTR )
                cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
```

```
            cout << "FXCH ST(7)" << endl;
            break;
        case FST:
            throw 1;
            cout << "FST ";
            if( i->myCmd & PTR )
                cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
            break;
        case FST7:
            throw 1;
            cout << "FXCH ST(7)" << endl
                 << "FST ";
            if( i->myCmd & PTR )
                cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
            cout << "FXCH ST(7)" << endl
                 << "FFREE ST(7)" << endl;
            break;
        case FADD:
            add( FADDP_ST1_ST, FADDP_ST1_ST_SIZE );
            break;
        case FSUB:
            add( FSUBP_ST1_ST, FSUBP_ST1_ST_SIZE );
            break;
        case FMUL:
            add( FMULP_ST1_ST, FMULP_ST1_ST_SIZE );
            break;
        case FDIV:
            add( FDIVP_ST1_ST, FDIVP_ST1_ST_SIZE );
            break;
        case FXCH:
            add( FXCH_ST1_ST, FXCH_ST1_ST_SIZE );
            break;
        default:
                    break;
        }
    }

    //some function ending code
       add( 0x89ec, 2); //mov %ebp,%esp
    add( 0x5d );     //pop
    add( 0xC3 );     //ret
}


double function::call( double someval )
{
    if( myFuncBuf )
    {
        const char *HEX="0123456789ABCDEF";
        cout << "Dump of function generated object code:" << endl;
        for( int i=0; i<FUNCTION_BUFFER_MAX; i++ )
        {
            cout << HEX[(myFuncBuf[i]>>4) & 0xF] << HEX[myFuncBuf[i]&0xF];
            if( (i+1) % 2 == 0 ) cout << '\t'; else cout << ' ';
        }
        cout << endl;
        myfunc_type brr = (myfunc_type) myFuncBuf;
        return brr(someval);
    }
    else
    {
        cerr << "Using function::call(...) before initialisation. Check()."
             << endl;
        throw 12222;
    }
}

//////////////////////////////////////////////////////////////////////////
// my_types.h
//
// (c) Alexander Afanasyev

#ifndef MY_MY_TYPES_H
#define MY_MY_TYPES_H

class BadOperation {};
class BadSyntax {};
class NullExpression {};

// type definitions
typedef char          oper_type;
typedef double        val_type;

#include <string>
typedef std::string   var_type;

typedef char   cmd_type;
typedef int    ptr_type;



struct ptr_val_type
{
    ptr_val_type():               myPtr(NULL),myVal(-1)  { };
    ptr_val_type( ptr_type ptr ): myPtr(ptr), myVal(-1)  { };
    ptr_val_type( val_type val ): myPtr(NULL),myVal(val) { };
```

```
    ptr_val_type( ptr_type ptr, val_type val ): myPtr(ptr),myVal(val) { };

    ptr_type myPtr;
    val_type myVal;

    bool val() { return myPtr==NULL; };
    bool ptr() { return myPtr!=NULL; };
};

struct var_val_type
{
    var_val_type(): myVar() {};
    var_val_type( var_type var ): myVar(var)             { };
    var_val_type( val_type val ): myVar(   ), myVal(val) { };
    var_val_type( val_type val,var_type var ): myVar( var ), myVal(val) { };

    var_type myVar;
    val_type myVal;

    bool var() { return myVar.size(); };
};

union val_cmd_ptr_type
{
    val_cmd_ptr_type( val_type val ): myVal(val) { };
    val_cmd_ptr_type( cmd_type cmd ): myCmd(cmd) { };
    val_cmd_ptr_type( ptr_type ptr ): myPtr(ptr) { };

    val_type   myVal;
    cmd_type   myCmd;
    ptr_type   myPtr;
};



#endif

////////////////////////////////////////////////////////////////////////
// nametable.h
//
// (c) Alexander Afanasyev

#ifndef MY_NAMETABLE_H
#define MY_NAMETABLE_H

#include "my_types.h"
#include <map>
#include <stack>


class nametable
{
public:
    typedef map<var_type,ptr_val_type>::iterator iterator;
    typedef map<var_type,ptr_val_type>::const_iterator const_iterator;

    iterator begin() { return myMap.begin(); };
    const_iterator begin() const { return myMap.begin(); };

    iterator end()   { return myMap.end();   };
    const_iterator end() const   { return myMap.end();   };

private:
    map<var_type,ptr_val_type> myMap;
    stack<iterator> myStack;
    int myOffset;
public:
    nametable( ): myOffset(0) {;};
    nametable( const nametable &nt ): myMap( nt.myMap ), myOffset(0) {;}
    nametable( const map<var_type,ptr_val_type> &m ): myMap(m),myOffset(0) {;};
    nametable( int offset ) : myOffset(offset) {;}

    int  offset( ) { return myMap.size(); }
    int  realoffset( ) { return myOffset; }
    void offset( int offs ) { myOffset=offs; }

    void insert( const nametable &table )
    {
        myMap.insert( table.begin(), table.end() );
    }

    ptr_type operator[]( const var_type &var );
    ptr_type push( const var_type &var );
    ptr_type pop( );

    int size() const { return myMap.size(); };
public: //exeption types
    class StackUnderflow{ };

};


#endif

////////////////////////////////////////////////////////////////////////
// nametable.cpp: my values/variables nametable definition
//
// (c)2002 Alexander Afanasyev

#include "nametable.h"
```

```
#include <map>


bool isnumber( const var_type &var );
/*{
    return ('0'<=var[0] && var[0]<='9') || var[0]=='.' ||
           var[0]=='-' || var[0]=='+';
} */


val_type tovalue(const var_type &var )
{
    const char *s=var.c_str();
    val_type val=0;
    int isdrob=0;

    val_type delitel=10.0;
    bool isminus=false;
    if( *s=='-' ) { isminus=true; s++; }
    else
    if( *s=='+' ) s++;


    for( ; *s && (isdigit(*s) || *s=='.'); s++ )
    {
        if( *s=='.' )
        {
            if( isdrob ) throw BadSyntax();
            isdrob=1;
            continue;
        }
        if( isdrob )
        {
            val+=(*s-'0')/delitel;
            delitel*=10.0;
        }
        else
            val=val*10+(*s-'0');
    }
    return ((isminus) ? -1 : +1 )*val;

       return 0;
}


ptr_type nametable::operator[]( const var_type &var )
{
    iterator ii=myMap.find( var );
    if( ii==myMap.end() )
    {
        if( !isnumber(var) )
        {
            pair<var_type,ptr_val_type> pp( var, ptr_type(myMap.size( )+1+myOffset) );
            myMap.insert( pp/*pair(var, ptr_type(myMap.size( )+1))*/ );
        }
        else
        {
            pair<var_type,ptr_val_type> pp(var,
                              ptr_val_type( ptr_type(myMap.size( )+1+myOffset),
                                                    tovalue(var) ));
            myMap.insert( pp );
        }
        return myMap.size()+myOffset;
    }
    else
        return ii->second.myPtr;
}


ptr_type nametable::push( const var_type &var )
{
    iterator ii=myMap.find( var );
    if( ii==myMap.end() )
    {
        if( !isnumber(var) )
        {
            pair<var_type,ptr_val_type> pp(var, ptr_type(myMap.size( )+1+myOffset));
            myStack.push(myMap.insert( myMap.begin(), pp ));
        }
        else
        {
            pair<var_type,ptr_val_type> pp(var,
                              ptr_val_type( ptr_type(myMap.size( )+1+myOffset),
                                                    tovalue(var) ));
            myStack.push(myMap.insert( myMap.begin(), pp ));
        }
        return myMap.size()+myOffset;
    }
    else
    {
        myStack.push( ii );
        return ii->second.myPtr;
    }
}

ptr_type nametable::pop( )
{
    if( myStack.size()==0 ) throw StackUnderflow();

    iterator ii=myStack.top();
```

```
    myStack.pop();
    return ii->second.myPtr;
}

////////////////////////////////////////////////////
// parser.h   -  header file for parser project
// (c) 2002 Alexander Afanasyev

#ifndef MY_PARSER_H
#define MY_PARSER_H

#include <vector>
#include <string>

#include "my_types.h"
#include "nametable.h"
#include "cmd_list.h"
#include "function.h"

#include "stack.h"


class expr
{
private:
    enum id { NUMBER, VARIABLE, _SK_EXPR_SK_ };
    cmd_list  myCmdList;
    function  myFunc;

//    void *myFunc; // pointer to point future generated function
//    bool myisfirstoperand;

    var_type  read_number  ( char *&s );
    var_type  read_variable( char *&s );
    oper_type read_operator( char *&s );

       void read_expr    ( char *&s );
       void read_ostvir  ( char *&s );
       void read_slag    ( char *&s );
       void read_ostmnozh( char *&s );
       void read_mnozh   ( char *&s );

    bool next_add_or_sub( const char *s );
    bool next_mul_or_div( const char *s );

    void read_skobka  ( char *&s );

    id probe_mnozh_type( const char *s );
public:
       expr( char *&s ) { read_expr(s); };
    expr( char *&s, int offset ): myCmdList(offset) { read_expr(s); };
    expr( char *&s, const cmd_list &cmds ): myCmdList(cmds) { read_expr(s); };

    void init()
    {
        myFunc.init( myCmdList );
    }

       ~expr() { };

    double call( double someval ) { return myFunc.call( someval ); };
    void dump_cmdlist( ostream &os );
};

#endif

#include <iostream.h>
#include <ctype.h>
#include <assert.h>
#include <map>

//#include "stack.cpp"
#include "parser.h"

#define MY_DEBUG 1

/*
V programme delayetsya korennaya zamena. Vse chisla s
tekuschego momenta imeyut absolutno takoye zhe predstavleniye,
chto i peremenniye! Vot. Absolutno takoye zhe predstavleniye.
Takim obrazom, 2.1 i 2.1, napisanniye v raznix mestax
vxodnoy stroki budut ukazivat na odnu i tu zhe oblast dannih.
Yedinstvennoye otlichiye ot prostix peremennih, oni budut
initializirovani znacheniyami tipa double, dlya sobsno ix posleduyu-
schego soxraneniya.
*/


/*
Alfavit yazika

expr     -> slag ostvir
ostvir   -> + expr
ostvir   -> - expr
ostvir   ->

slag     -> mnozh ostmnozh
ostmnozh -> * slag
ostmnozh -> / slag
ostmnozh ->
```

```
mnozh    -> number
mnozh    -> variable
mnozh    -> ( expr )

*/

//expr     -> slag ostvir    :)
void expr::read_expr( char *&s )
{
    assert( s!=NULL );
    while( isspace(*s) ) s++;

    read_slag(s);
    read_ostvir(s);
}

//ostvir   -> + expr        :)
//ostvir   -> - slag ostvir        :)
//ostvir   ->               :)
void expr::read_ostvir( char *&s )
{
    assert( s!=NULL );
    while( isspace(*s) ) s++;
    if( !*s || !next_add_or_sub(s) ) return;

    oper_type oper=read_operator(s);

    switch( oper )
    {
    case '+':
        read_expr(s);
        myCmdList.add( FADD );
            break;
    case '-':
        read_slag(s);
        myCmdList.add( FSUB );
        read_ostvir(s);
            break;
    default:
        throw BadSyntax();
    }
    myCmdList.pop( );
}

//slag     -> mnozh ostmnozh
void expr::read_slag( char *&s )
{
    assert( s!=NULL );
    while( isspace(*s) ) s++;

    read_mnozh(s);
    read_ostmnozh(s);
}

//ostmnozh -> * slag
//ostmnozh -> / mnozh ostmnozh
//ostmnozh ->               :)
void expr::read_ostmnozh( char *&s )
{
    assert( s!=NULL );
    while( isspace(*s) ) s++;
    if( !*s || !next_mul_or_div(s) ) { return; }

    oper_type oper=read_operator(s);
    switch( oper )
    {
    case '*':
        read_slag(s);
        myCmdList.add( FMUL );
            break;
    case '/':
        read_mnozh(s);
        myCmdList.add( FDIV );
        read_ostmnozh(s);
            break;
    default:
        throw BadSyntax();
    }

    myCmdList.pop( );

/*    if( !myCmdList )
    {
        expr next_level( s );
        myCmdList+=next_level.myCmdList;
    }*/

}

//mnozh    -> number
//mnozh    -> variable
//mnozh    -> ( expr )
void expr::read_mnozh( char *&s )
{
    assert( s!=NULL );
    while( isspace(*s) ) s++;
    switch( probe_mnozh_type(s) )
    {
    case NUMBER:
```

```
        if( !myCmdList )
        {
            expr next_level( s, myCmdList );
            myCmdList+=next_level.myCmdList;
        }
        else
            myCmdList.push( read_number(s) );
        break;
    case VARIABLE:
        if( !myCmdList )
        {
            expr next_level( s, myCmdList );
            myCmdList+=next_level.myCmdList;
        }
        else
            myCmdList.push( read_variable(s) );
        break;
    case _SK_EXPR_SK_:
        {
       read_operator(s); //'('
        read_expr(s);
       read_operator(s); //')'
        }
        break;
    default:
       throw BadSyntax();
    }
}


bool expr::next_add_or_sub( const char *s )
{
    return *s=='-' || *s=='+';
}

bool expr::next_mul_or_div( const char *s )
{
    return *s=='*' || *s=='/';
}


oper_type expr::read_operator( char *&s )
{
    return *s++;
}


expr::id expr::probe_mnozh_type( const char *s )
{
       if( !*s ) throw BadSyntax();
    if( ('0'<=*s && *s<='9') || *s=='.' ) return NUMBER;
    if( ('a'<=*s && *s<='z') ||
        ('A'<=*s && *s<='Z')            ) return VARIABLE;
    if( *s=='-' || *s=='+' )
    {
           if( ('0'<=*(s+1) && *(s+1)<='9') || *(s+1)=='.' ) return NUMBER;
//      if( ('a'<=*s && *s<='z') ||
//            ('A'<=*s && *s<='Z')            ) return VARIABLE;
//   expressions like '-x'  are forbidden!
    }
    if( *s=='(' )                          return _SK_EXPR_SK_;
    throw BadSyntax();
}

var_type expr::read_number( char *&s )
{
    var_type var;

    if( *s=='-' || *s=='+' ) { var+=*s; s++; }
    for( ; *s && (isdigit(*s) || *s=='.'); s++ ) { var+=*s; };

    return var;
}


var_type expr::read_variable( char *&s )
{
    var_type var;

    for( ; *s && (isdigit( *s ) || isalpha( *s ) || *s=='_'); s++ )
        var+=*s;
    return var;
}


void expr::dump_cmdlist( ostream &os )
{
    for( cmd_list::iterator i=myCmdList.begin();
         i<myCmdList.end();
         i++
       )
    {
        switch (i->myCmd & 31)
        {
        case FLD:
            cout << "FLD ";
            if( i->myCmd & PTR )
                cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
```

```
            break;
        case FLD7:
            cout << "FLD ";
            if( i->myCmd & PTR )
               cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
            cout << "FXCH ST(7)" << endl;
            break;
        case FST:
            cout << "FST ";
            if( i->myCmd & PTR )
               cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
            break;
        case FST7:
            cout << "FXCH ST(7)" << endl
                 << "FST ";
            if( i->myCmd & PTR )
               cout << '[' << (++i)->myPtr << ']' << endl;
            else
               cout << (++i)->myVal << endl;
            cout << "FXCH ST(7)" << endl
                 << "FFREE ST(7)" << endl;
            break;
        case FADD:
               cout << "FADDP ST(1),ST" << endl;
            break;
        case FSUB:
               cout << "FSUBP ST(1),ST" << endl;
            break;
        case FMUL:
               cout << "FMULP ST(1),ST" << endl;
            break;
        case FDIV:
               cout << "FDIVP ST(1),ST" << endl;
            break;
        case FXCH:
               cout << "FXCH ST(1),ST" << endl;
            break;
        default:
                     break;
        }
    }
}


int main( int argc, char **argv )
{
try
{
//    cerr << "Nexren zapuskat, do okonchaniya raboti !!! " << endl;
//    return 1;

    if( argc<2 )
    {
        cerr << "Usage: parser.exe <math_expression> " << endl;
        return 1;
    }
    cout << "--== NEW EXPRESSION: `" << argv[1] << "` ==---" << endl;
    expr my_expr( argv[1] );
    my_expr.init();
//    my_expr.dump_cmdlist( cout );
    cout << "After object code execution value is " << my_expr.call(2.1) << endl;
    my_expr.dump_cmdlist( cout );
}
/*catch( StackUnderflow )
{
#ifndef MY_DEBUG
    cerr << "Error: incorrect syntax, please check" << endl;
#else
    cerr << "Error: catching exeption BlankStack()" << endl;
#endif
       return 7;
}
catch( StackOverflow )
{
#ifndef MY_DEBUG
    cerr << "Error: too big expression. Please update program version"
         << endl;
#else
    cerr << "Error: catching exeption StackOverflow()" << endl;
#endif
       return 8;
}
*/
catch( BadOperation )
{
#ifndef MY_DEBUG
    cerr << "Error: undefinied operation" << endl;
#else
    cerr << "Error: catching exeption BadOperation()" << endl;
#endif
       return 16;
}
catch( BadSyntax )
{
#ifndef MY_DEBUG
```

```
    cerr << "Error: error in expression" << endl;
#else
    cerr << "Error: catching exeption BadSyntax()" << endl;
#endif
       return 16;
}
catch( ... )
{
#ifndef MY_DEBUG
    cerr << "Error: unknown error" << endl;
#else
    cerr << "Error: catching exeption <unknown>" << endl;
#endif
       return 255;
}
}

#ifndef MY_STACK_H
#define MY_STACK_H

#include <iostream.h>

const int MAX_BUF_SIZE=256;

class StackUnderflow {};
class StackOverflow {};


template<class T>
class my_stack
{
private:
    T   myBuf[MAX_BUF_SIZE];
    int mySize;
public:
    my_stack( ) : mySize(0) {;};


    int size() const { return mySize; };

    inline void push( const T &val );
    inline T    pop();
    inline T    top();

    inline T    operator[]( const int &i ) const
    {
        if( i>mySize-1 ) throw StackUnderflow();
        return myBuf[mySize-i-1];
    }

    bool ok()   const { return 0<=mySize && mySize<=MAX_BUF_SIZE; };
    void dump( ostream &os ) const
    {
        for( int i=0; i<MAX_BUF_SIZE; i++ )
           os << myBuf[i] << "\t";
    }
};


template<class T>
inline void my_stack<T>::push( const T &val )
{
    if( mySize==MAX_BUF_SIZE ) throw StackOverflow();
    myBuf[mySize++]=val;
};

template<class T>
inline T    my_stack<T>::pop()
{
    if( mySize==0 ) throw StackUnderflow();
    return myBuf[--mySize];
};

template<class T>
inline T    my_stack<T>::top()
{
    if( mySize==0 ) throw StackUnderflow();
    return myBuf[mySize-1];
};


#endif
```

4. Методическое обеспечение.

Программа предназначена для использования в среде MS DOS, Windows, Linux. Для компиляции исходного текста программы использовался компилятор GNU C++ (в среде DOS из комплекта GNU DJGPP http://www.delorie.com/djgpp/). Для начала

работы с программой необходимо выполнить исполняемый файл parser.exe. Для работы в среде Linux необходимо распаковать поставляемый дистрибутив в какой-нибудь каталог и запустить команду `make`. Для запуска программы использовать `./parser.exe`

В командной строке при запуске программы указывается математическое выражение для разбора его программой.

Состав комплекта:

- Документация по разработке (отчет) - файл otchet.doc.
- Исходный текст программы - файлы parser.cpp, parser.h, stack.cpp, stack.h, makefile.

Список использованных источников

1. Б.Страуструп, Язык программирования C++, 3-е изд./Пер. с англ. – СПб.; М.: «Невский Диалект» - «Издательство БИНОМ», 1999 г. – 991 с., ил.
2. Intel Architecture Software Developer’s Manual, Floating-Point Unit (fpu.pdf)
3. Mike Eddy, Coprocessor (09LMARFC05.pdf)

Московский ордена Ленина, ордена Октябрьской Революции
и ордена Трудового Красного Знамени.
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н. Э. БАУМАНА

Факультет: Информатики и систем управления
Кафедра: Проектирование и технология производства электронной аппаратуры (ИУ4)

О Т Ч Е Т

По лабораторной работе №1
"Разработка диагностирующе-обучающего класса, работающего аналогично встроенному классу ***int***"

По курсу: Программирование на языке С/С++

Студент: Афанасьев А.В. ИУ4-22
(фамилия, инициалы) (индекс группы)

Руководитель: Дединский И.Р.
(фамилия, инициалы)

отметка о защите	дата защиты

Москва
2002

Содержание

Техническое задание

Наименование разработки: создание диагностирующе-обучающего класса, аналогичного классу *int* с выводом диагностирующих сообщений.

Цель работы – ознакомиться с основными синтаксическими конструкциями, обеспечивающими адекватное реагирование объектов пользовательский класса на стандартные операции:

- сложения, умножения, деления;
- приведения типов;
- логические операции;
- работы с потоками (stream) данных.

Решаемые задачи:

Реализовать класс и предоставить к нему интерфейс, включающий в себя поддержку стандартных математических, и логических операций, операций присваивания, операций преобразования типов и других.

Для обеспечения диагностики, встроить в класс вывод сообщений на экран, при проведении любых операций с объектами этого класса.

Дополнительное задание – реализация неадекватного поведения всех объектов данного класса в случайные моменты времени, с возможностью задания процента неправильной работы во времени.

Написать пример использования данного класса.

Разработанная программа должна работать как в системе Windows, так и Linux и компилироваться следующими компиляторами:

- GNU C++
- Microsoft Visual C++ 6.0
- Borland C++ 3.1
- Borland C++ 5.02

Этапы выполнения:

1. Разработать класс, реализующий описанные выше функции.
2. Разработать пример программы, использующей разработанный класс.

Форма отчетности:

По результатам разработки предоставляется: отчет (файл otchet.doc), исходный текст программы (файлы superint.h, superint.cpp, main.cpp, makefile), исполняемый файл (superint.exe).

3. Лингвистическое обеспечение

Согласно ТЗ в качестве лингвистического обеспечения использован язык C++.

4. Программное обеспечение

4.1. Использование библиотечных функций

Для реализации данной программы были использованы следующие стандартные библиотеки:

1. iostream.h - модуль с текстовыми потоками данных
2. stdio.h, stdlib.h – модули стандартных функций
3. time.h – модуль, содержащий прототипы функций обрабатывающих системное время

4.2. Формализация предметной области и допущения

В ходе реализации программы был сделан ряд допущений:
процент неадекватного поведения объектов класса (с выводом сообщения об ошибке) определяется общим для всех объектов параметра.

4.3. Структурно-функциональное построение программы

Интерфейс класса представлена в файле superint.h.

Таблица 1. Исходный листинг программы.

Файл superint.h

```
#ifndef SUPERINT_H
#define SUPERINT_H

#include <iostream.h>

const int default_bug_percentage=25;

#if !(__TURBOC__) || (__TURBOC__ > 0x0410)
typedef bool BOOL;
#else
typedef int BOOL;
#endif

class Int
{
private:
  int val;
  static int buggy;

  static BOOL bug();
public:
  static void setbuggy(int i=25) { Int::buggy=i%101; };

  Int();
  Int( const Int& i );
  Int( const int& i );

#if !(__TURBOC__) || (__TURBOC__ > 0x0410)
  operator bool();
#endif
  operator short();
  operator int();
  operator long();
#if !(__TURBOC__) && !(_MSC_VER)
  operator long long();
#endif
  operator float();
  operator double();

  Int& operator=( const Int& i );

  Int operator+( const Int& i ) const;
  Int& operator+=( const Int& i );

  Int& operator+();
  Int& operator-();

  Int operator-( const Int& i ) const;
  Int& operator-=( const Int& i );

  Int operator*( const Int& i ) const;
  Int& operator*=( const Int& i );
```

```
  Int operator/( const Int& i ) const;
  Int& operator/=( const Int& i );

  Int operator%( const Int& i ) const;
  Int& operator%=( const Int& i );

  Int operator&( const Int& i ) const;
  Int& operator&=( const Int& i );

  Int operator|( const Int& i ) const;
  Int& operator|=( const Int& i );

  Int operator^( const Int& i ) const;
  Int& operator^=( const Int& i );

  Int operator>>( const int& i ) const;
  Int& operator>>=( const int& i );

  Int operator<<( const int& i ) const;
  Int& operator<<=( const int& i );

  BOOL operator||( const Int& i ) const;
  BOOL operator&&( const Int& i ) const;
  BOOL operator!() const;
  BOOL operator==( const Int& i ) const;
  BOOL operator!=( const Int& i ) const;
  BOOL operator>=( const Int& i ) const;
  BOOL operator<=( const Int& i ) const;
  BOOL operator>( const Int& i ) const;
  BOOL operator<( const Int& i ) const;

  friend ostream& operator<<( ostream &os, const Int& i );
  friend istream& operator>>( istream &is, Int& i );
};

ostream& operator<<( ostream &os, const Int& i );
istream& operator>>( istream &is, Int& i );

#endif
```

Файл superint.cpp

```
#include "superint.h"
#include <iostream.h>
#include <stdlib.h>

int Int::buggy=default_bug_percentage;

inline int my_rand()
{
#ifdef _MSC_VER
    return rand();
#else
#ifdef __TURBOC__
    return random(10000);
#else
    return random();
#endif
#endif
}

Int::Int()
{
    cout << "Исполняю: конструктор по умолчанию" << endl;
};

Int::Int( const Int& i ) : val(i.val)
{
    cout << "Исполняю: конструктор Int::Int( const Int& i ): "
         << "i.val=" << i.val << endl;
    if( Int::bug() ) val=my_rand();
}

Int::Int( const int& i ) : val(i)
{
        cout << "Исполняю: конструктор Int::Int( const int& i ): "
                << "i.val=" << i << endl;
        if( Int::bug() ) val=my_rand();
}
```

```
BOOL Int::bug()
{
        if( my_rand()%100<buggy )
        {
                int k=my_rand()%100;
                if( k%100<40 )
                        cout << "Error: lame error, maybe your programmer "
                                << "need more money?" << endl;
                else
                if( k%100<60 )
                        cout << "Error: memory error, you don't think, that "
                                << "that programmer wants eat too?" << endl;
                else
                if( k%100<80 )
                        cout << "Error: please check programmer's wade. "
                                << "He wants more :)" << endl;
                else
                        cout << "Error: this operation is forbidden in this "
                                << "trial beta version. Please contact to somebody" << endl;

                return 1;
        }
        return 0;
}

#if !(__TURBOC__) || (__TURBOC__ > 0x0410)
Int::operator bool()
{
        cout << "Исполняю: operator bool(): val=" << val << endl;
        if( Int::bug() )
                return my_rand()%2;
        else
                return val;
}
#endif

Int::operator short()
{
        cout << "Исполняю: operator short(): val=" << val << endl;
        if( Int::bug() )
                return my_rand();
        else
                return val;
}

Int::operator int()
{
    cout << "Исполняю: operator int(): val=" << val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val;
}

Int::operator long()
{
    cout << "Исполняю: operator long(): val=" << val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val;
}

#ifndef __TURBOC__
#ifndef _MSC_VER
Int::operator long long()
{
    cout << "Исполняю: operator long long(): val=" << val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val;
```

```
}
#endif
#endif

Int::operator float()
{
    cout << "Исполняю: operator float(): val=" << val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val;
}

Int::operator double()
{
    cout << "Исполняю: operator double(): val=" << val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val;
}

Int& Int::operator=( const Int& i )
{
    cout << "Исполняю: operator=( const Int& i ): "
            "скопируем-ка " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val=i.val;
    return *this;
}

Int Int::operator+( const Int& i ) const
{
    cout << "Исполняю: operator+( const Int& i ) const: " <<
            "Сложим " << val << " с " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val+i.val;
}

Int& Int::operator+=( const Int& i )
{
    cout << "Исполняю: operator+=( const Int& i ): " <<
            "Сложим (+=)" << val << " с " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val+=i.val;
    return *this;
}

Int Int::operator-( const Int& i ) const
{
    cout << "Исполняю: operator-( const Int& i ) const: " <<
            "Отнимем " << val << " от " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val-i.val;
}

Int& Int::operator-=( const Int& i )
{
    cout << "Исполняю: operator-=( const Int& i ): " <<
            "Отнимем (-=)" << val << " от " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val-=i.val;
```

```
    return *this;
}

Int Int::operator*( const Int& i ) const
{
    cout << "Исполняю: operator*( const Int& i ) const: " <<
            "Помножим " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val*i.val;
}

Int& Int::operator*=( const Int& i )
{
    cout << "Исполняю: operator*=( const Int& i ): " <<
            "Помножим (*=) " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val*=i.val;
    return *this;
}

Int Int::operator/( const Int& i ) const
{
    cout << "Исполняю: operator/( const int& i ): " <<
            "Поделим " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val/i.val;
}

Int& Int::operator/=( const Int& i )
{
    cout << "Исполняю: operator/=( const int& i ): " <<
            "Поделим (/=) " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val/=i.val;
    return *this;
}

Int Int::operator%( const Int& i ) const
{
    cout << "Исполняю: operator%( const Int& i ): " <<
            "Возьмем остаток от деления " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val%i.val;
}

Int& Int::operator%=( const Int& i )
{
    cout << "Исполняю: operator%=( const Int& i ): " <<
            "Возьмем остаток от деления " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val%=i.val;
    return *this;
}

Int Int::operator&( const Int& i ) const
{
    cout << "Исполняю: operator&( const Int& i ): " <<
            "Битовое И: " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
```

```
    else
        return val&i.val;
}

Int& Int::operator&=( const Int& i )
{
    cout << "Исполняю: operator&=( const Int& i ): " <<
            "Битовое И: " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val&=i.val;
    return *this;
}

Int Int::operator|( const Int& i ) const
{
    cout << "Исполняю: operator|( const Int& i ): " <<
            "Битовое ИЛИ: " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val|i.val;
}

Int& Int::operator|=( const Int& i )
{
    cout << "Исполняю: operator|=( const Int& i ): " <<
            "Битовое ИЛИ: " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val|=i.val;
    return *this;
}

Int Int::operator^( const Int& i ) const
{
    cout << "Исполняю: operator^( const Int& i ): " <<
            "Битовое исключающее ИЛИ: " << val << " на " << i.val << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val^i.val;
}

Int& Int::operator^=( const Int& i )
{
    cout << "Исполняю: operator^=( const Int& i ): " <<
            "Битовое исключающее ИЛИ: " << val << " на " << i.val << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val^=i.val;
    return *this;
}

Int Int::operator>>( const int& i ) const
{
    cout << "Исполняю: operator>>( const Int& i ): " <<
            "Битовый сдвиг вправо " << val << " на " << i << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val>>i;
}

Int& Int::operator>>=( const int& i )
{
    cout << "Исполняю: operator>>=( const Int& i ): " <<
            "Битовый сдвиг вправо " << val << " на " << i << endl;
    if( Int::bug() )
```

```
        val=my_rand();
    else
        val>>=i;
    return *this;
}

Int Int::operator<<( const int& i ) const
{
    cout << "Исполняю: operator<<( const Int& i ): " <<
            "Битовый сдвиг влево " << val << " на " << i << endl;
    if( Int::bug() )
        return my_rand();
    else
        return val<<i;
}

Int& Int::operator<<=( const int& i )
{
    cout << "Исполняю: operator<<=( const Int& i ): " <<
            "Битовый сдвиг влево " << val << " на " << i << endl;
    if( Int::bug() )
        val=my_rand();
    else
        val<<=i;
    return *this;
}


BOOL Int::operator||( const Int& i ) const
{
    cout << "Исполняю: operator||( const Int& i ): " <<
            "Logic OR " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val||i.val;
}

BOOL Int::operator&&( const Int& i ) const
{
    cout << "Исполняю: operator&&( const Int& i ): " <<
            "Logic AND " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val&&i.val;
}


BOOL Int::operator!() const
{
    cout << "Исполняю: operator!( ): " <<
            "NOT operation to " << val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return !val;
}

BOOL Int::operator==( const Int& i ) const
{
    cout << "Исполняю: operator==( const Int& i ): " <<
            "Compare " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val==i.val;
}

BOOL Int::operator!=( const Int& i ) const
{
    cout << "Исполняю: operator!=( const Int& i ): " <<
```

```
            "Compare (if not) " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val!=i.val;
}

BOOL Int::operator>=( const Int& i ) const
{
    cout << "Исполняю: operator>=( const Int& i ): " <<
            "Compare (more or equal) " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val>=i.val;
}

BOOL Int::operator<=( const Int& i ) const
{
    cout << "Исполняю: operator<=( const Int& i ): " <<
            "Compare (less or equal) " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val<=i.val;
}

BOOL Int::operator>( const Int& i ) const
{
    cout << "Исполняю: operator==( const Int& i ): " <<
            "Compare (more) " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val>i.val;
}

BOOL Int::operator<( const Int& i ) const
{
    cout << "Исполняю: operator<( const Int& i ): " <<
            "Compare (less) " << val << " with " << i.val << endl;
    if( Int::bug() )
        return my_rand()%2;
    else
        return val<i.val;
}


ostream& operator<<( ostream &os, const Int& i )
{
    cout << "Исполняю: operator<<( ostream &os, const Int& i )" << endl
         << "Выводим на экран " << i.val << endl;
    if( Int::bug() )
        return os << my_rand();
    else
        return os << i.val;
}

istream& operator>>( istream &is, Int& i )
{
    cout << "Исполняю: operator>>( istream &is, const Int& i )" << endl
         << "Вводим с экрана" << endl;
    int temp;
    is >> temp;
    if( Int::bug() )
    {
        i.val=my_rand();
        return is;
    }
    else
    {
        i.val=temp;
```

```
        return is;
    }
}
```

Файл main.cpp

```
int main( int argc, char** argv )
{
    if( argc>1 )
        Int::setbuggy( atoi(*(argv+1)) );
    clrscr();
#ifndef __TURBOC__
    srandom( rawclock() );
#else
    randomize();
#endif
    Int i=5,j;
    j=i/Int(5.0);
    cout << i << endl;
    cout << j << endl;
    Int t=1.0;
    t<<=2;
    bool bool_=t;
    short short_=t;
    int int_=t;
    long long_=t;
#ifndef __TURBOC__
    long long long__=t;
#endif
    float float_=t;
    double double_=t;
}
```

Файл makefile

```
WARN=-Wall

all: superint.exe

clean:
       del *.o
       del *.obj

superint.exe: superint.o main.o
       gpp superint.o main.o -o superint.exe
       strip superint.exe

superint.o: superint.cpp
       gcc $(WARN) -c superint.cpp

main.o: main.cpp
       gcc $(WARN) -c main.cpp
```

5. Методическое обеспечение.

Программа предназначена для использования в среде MS DOS, Windows, Linux. Для начала работы с программой необходимо выполнить исполняемый файл superint.exe. В командной строке можно указать процент неадекватности работы (0-100).

Состав комплекта:

- Документация по разработке (отчет) - файл otchet.doc.
- Исходный текст программы - файлы superint.h, superint.cpp, main.cpp, makefile.
- Исполняемый файл - файл superint.exe.

Список использованных источников

1. Б.Страуструп, Язык программирования C++, 3-е изд./Пер. с англ. – СПб.; М.: «Невский Диалект» - «Издательство БИНОМ», 1999 г. – 991 с., ил.