Imię i Nazwisko	Maciej Korzeń
Numer grupy	IZ204
Nazwa przedmiotu	Programowanie Obiektowe
Tytuł projektu	Program do obliczania wartości wyrażenia arytmetycznego
Data wykonania	VI 2006

Spis treści

1. Wstęp

Program ma za zadanie obliczać wartość wyrażenia arytmetycznego podanego przez użytkownika. Obsługiwane operatory to **+, -, *, /, ^, (,)** oraz **minus** jednoargumentowy. Program pracuje w środowisku tekstowym. Za pomocą odpowiednich parametrów wywołania można zapisywać lub odczytywać dane z/do określonych plików.

2. Algorytm

Program wykorzystuje odwrotną notację polską do operacji na wyrażeniu. Pozwala to m.in. w łatwy sposób odnajdywać błędy składniowe w wyrażeniach.

A	StateStart
B	StateOperand
C	StateNumber
D	StateNumberPoint
E	StateNumberMantisa
F	StateNumberPowerSign
G	StateNumberPowerMinus
H	StateNumberPower
I	StateOperator
J	StateEnd

0	CmdNone
1	CmdSaveStartPos
2	CmdNumber
3	CmdMinus
4	CmdOpen
5	CmdClose
6	CmdNumber, CmdClose
7	CmdOperator
8	CmdNumber, CmdOperator
9	CmdEnd
10	CmdNumber, CmdEnd

Operatory i Nawiasy	Nazwa	Waga
-	1	Sub
+	1	Add
*	2	Mul
/	2	Div
^	3	Pow
jednoargumentowy -	4	Neg
(,)	+5, -5	

CmdNone - nic nie robimy.

CmdSaveStartPos - zapisujemy bieżącą pozycję do zmiennej.

CmdNumber - przyjmujemy liczbę zawartą pomiędzy zapisaną poprzednio pozycją a bieżącą, i zrzucamy ją na stos główny.

CmdMinus - zrzucamy operator (minus jednoargumentowy) z wagą zwiększoną o sumaryczną wagę otwartych nawiasów na stos dodatkowy.

CmdOpen - zwiększamy sumaryczną wagę otwartych nawiasów o wagę jednego nawiasu.

CmdClose - zmniejszamy sumaryczną wagę otwartych nawiasów o wagę jednego nawiasu.

CmdOperator - obliczamy wagę operatora jako wagę operatora w pozycji bieżącej zwiększoną o sumaryczną wagę otwartych nawiasów. Przerzucamy ze stosu dodatkowego na stos główny operatory o łącznej wadze większej lub równej łącznej wadze bieżącego operatora. Zrzucamy bieżący operator na stos dodatkowy.

CmdEnd - Przerzucamy ze stosu dodatkowego na stos główny wszystkie operatory;

	\n Eof	\x20 \t	0-9	.	E e	-	+	* / ^	(	)
A		A,0	C,1	D,1		B,3			A,4	
B		B,0	C,1	D,1					A,4	
C	J,10	I,2	C,0	E,0	F,0	B,8	B,8	A,8		I,6
D			E,0							
E	J,10	I,2	E,0		F,0	B,8	B,8	A,8		I,6
F			H,0			G,0				
G			H,0							
H	J,10	I,2	H,0			B,8	B,8	A,8		I,6
I	J,9	I,0				B,7	B,7	A,7		I,5

Kompilacja wyrażenia: Na początku stan kompilacji jest **StateStart**, natomiast sumaryczna waga otwartych nawiasów wynosi zero. Następnie, w zależności od bieżącego stanu i kolejnego symbolu z wiersza polecenia przechodzimy do odpowiedniego stanu według tabeli, oraz wykonujemy odpowiednie polecenie Cmd. Jeżeli w tabeli nie ma bieżącego znaku wiersza polecenia lub odpowiednia komórka w tabeli jest pusta, to bieżąca pozycja w wierszu polecenia jest pozycją błędu. Przy osiągnięciu stanu **StateEnd** kompilacja jest udana, jeżeli sumaryczna waga otwartych nawiasów wynosi zero. W przeciwnym wypadku koniec wiersza polecenia jest pozycją błędu.

Wydruk stosu: Stos jest drukowany od dołu do góry. Jeżeli kolejny element stosu jest liczbą - drukujemy „**Push** liczba”, w przeciwnym wypadku drukujemy nazwę operatora „**Sub**”, „**Add**”, „**Mul**”, „**Div**”, „**Pow**” lub „**Neg**”.

Wyliczenie wartości wyrazu: Przerzucamy wszystko na stos dodatkowy - odwrócenie „do góry nogami”. Następnie przerzucamy z powrotem jeżeli kolejny element jest liczbą. W przypadku operatora **Neg** zmieniamy znak wierzchołka stosu, w przypadku pozostałych operatorów wyciągamy ze stosu głównego wierzchołek jako **X**, wyciągamy ze stosu głównego wierzchołek jako **Y**, wykonujemy **X operator Y**, wynik zaś zrzucamy z powrotem na stos główny. Po wyczerpaniu zawartości stosu dodatkowego, w stosie głównym ma pozostać jedyny element - liczba, która jest wynikiem obliczanego wyrażenia.

3. Pliki zewnętrzne i biblioteki

Pliki nagłówkowe **iostream**, **iomanip**, **fstream** oraz **string** dostarczają podstawowych funkcji do operacji na zmiennych, odczytywania i zapisywania danych do plików (oraz standardowego wejścia i wyjścia), operacji na ciągach znaków.

Plik nagłówkowy **math.h** dostarcza funkcji arytmetycznych do wykonywania niezbędnych operacji (np. potęgowanie).

4. Zmienne globalne

enum State	Wylicza możliwe stany w analizowanym wyrażeniu.
enum Input	Wylicza możliwe rodzaje znaków w wyrażeniu.
enum Ops	Wylicza możliwe operacje, które mogą być wykorzystane w wyrażeniu.
bool ErrorPrintExpression	Jeśli w przypadku błędy należy wyświetlić błędne wyrażenie, to zmienna przyjmuje wartość **true**.
ostream * myout	Wskazuje, gdzie mają być wysyłane komunikaty (czyli np. standardowe wyjście lub plik).
ostream * myerr	Wskazuje, gdzie mają być zapisywane komunikaty o błędach (np. plik).

5. Funkcje

1. ErrorDivByZero

Zwracana wartość:	void
Parametry:	void (Brak)
Opis:	Wyświetla komunikat o błędzie.

*2. Cmd**

Funkcje te zostały opisane w części **Algorytm**.

3. StosWydruk

Zwracana wartość:	void
Parametry:	Stos &
Opis:	Interpretuje zawartość podanego stosu jako zapis odwrotnej notacji polskiej i wypisuje odpowiednio sformatowany zapis tego stosu.

4. PobierzWyrazenie

Zwracana wartość:	char *
Parametry:	void
Opis:	Odczytuje ze standardowego wejścia wyrażenie, którego wartość program ma obliczyć.

5. PoliczLinijki

Zwracana wartość:	unsigned
Parametry:	char *
Opis:	Oblicza ilość linijek z których składa się podany plik. Zwraca obliczoną wartość.

6. WartoscWyrazu

Zwracana wartość:	bool
Parametry:	Stos &, Stos &
Opis:	Analizuje zawartość stosów i na tej podstawie wykonuje wszystkie podane operacje arytmetyczne. W razie wystąpienia błędu zwraca **false**, w przeciwnym wypadku zwraca **true**.

7. ErrorOnPosition

Zwracana wartość:	unsigned
Parametry:	char *
Opis:	Oblicza ilość linijek z których składa się podany plik. Zwraca obliczoną wartość.

8. KonwertujWyrazenie

Zwracana wartość:	bool
Parametry:	char *, Stos &, Stos &, TabRecord[][]
Opis:	Wyrażenie podane jako pierwszy argument konwertuje na postać stosu zapisaną w zmiennych podanych jako drugi i trzeci argument wywołania. W razie napotkania

	błędu składniowego zwraca wartość **false**. W procesie konwersji wykorzystywana jest dwuwymiarowa tablica stanów podana jako ostatni z argumentów.

9. Koniec

Zwracana wartość:	void
Parametry:	void
Opis:	Wypisuje komunikat pożegnalny i kończy działanie programu. Jeśli program wyświetla komunikaty na standardowym wyjściu, to funkcja prosi o naciśnięcie klawisza **[Enter]**.

10. Pomoc

Zwracana wartość:	void
Parametry:	void
Opis:	Wyświetla dodatkowe polecenia, których można użyć.

11. OptionFind

Zwracana wartość:	char *
Parametry:	int, char **, char
Opis:	Sprawdza czy jeden z podanych w wierszu argumentów zaczyna się znakiem ukośnika (**/**) oraz czy następuje po nim znak podany jako trzeci argument wywołania funkcji. Jeśli tak, to jest zwracany adres znaku, który znajduje się zaraz za **/X** (gdzie **X** to szukany znak). Jeśli nie znaleziono szukanego argumentu, to funkcja zwraca **NULL**. Funkcja jest wykorzystywana do sprawdzenia opcji **/o**, **/i** oraz **/l**.

12. HaveArgument

Zwracana wartość:	bool
Parametry:	int, char **, char
Opis:	Sprawdza czy jeden z podanych w wierszu argumentów to **/X**, gdzie **X** to znak podany jako trzeci argument wywołania funkcji. Jeśli tak, to zwracana jest wartość **true**. W przeciwnym wypadku zwracane jest **false**.

13. PrintHelp

Zwracana wartość:	void
Parametry:	ostream *
Opis:	Wypisuje (do obiektu podanego jako argument) informacje na temat przełączników, których możemy użyć przy wywoływaniu programu.

6. Klasy

1. Stos

Klasa reprezentuje dynamicznie rosnący stos. Możemy na niego wkładać elementy i zdejmować je ze stosu. Elementy mogą być także usuwane ze środka stosu. Zawiera publiczną podklasę **Element**.

Posiada konstruktor domyślny, destruktor, oraz operatory: rzutowania na typ **bool**, przesunięć bitowych w lewo raz w prawo, nawiasów okrągłych.

Zmienne prywatne: rozmiar, pocztek (wskazuje na pierwszy element).

Metody: elementPtr (zwraca adres elementu o poadnym indeksie), remove (usuwa element o podanym indeksie), last (zwraca adres ostatniego elementu), vGet i wGet (zwracają odpowiednio wartość zmiennej **v** lub **w** z elementu o podanym indeksie).

Metody prywatne: rem (zdejmuje element ze stosu), add (wkłada element na stos).

2. Stos::Element

Klasa ta reprezentuje pojedynczy element w obrębie stosu. Służy głównie do przechowywania zmiennych **w** (od weight - ang. waga) oraz **v** (od value - ang. wartość). Posiada oczywiście zmienne prywatne **v** oraz **w** oraz dodatkowo zmienną **next** wskazującą na następny element na stosie.

W jej skład wchodzą publiczne metody **vGet**, **wGet**, **nextGet**, **wSet**, **vSet**, **nextSet**, które odpowiednio odczytują/ustawiają wartości zmiennych **v**, **w** oraz **next**.

Klasa posiada konstruktory domyślny oraz kopiujący, operator przypisania i metodę **print** (wyświetla ona zawartość dane egzemplarza klasy).

7. Instrukcja obsługi

Po skompilowaniu programu, uruchamiamy go poleceniem **projekt1** (lub **./projekt1** w systemach Uniksowych). Oczywiście pod warunkiem, że plik wykonywalny nazwaliśmy **projekt1**.

```
Witamy w programie 'projekt1'. Zyczymy milej pracy.
Przydatne komendy:
 ? - pomoc,
 ! - wyjscie z programu,
projekt1>
```

Możemy teraz wprowadzić wyrażenie arytmetyczne, kótrego wynik chcemy poznać. Dozwolone są operatory **+**, **-**, ***, */*, **^**, **(**, **)** oraz **minus** jednoargumentowy. Liczby możemy zapisywać w postaci dziesiętnej (np. **45.346**) oraz korzystając z liczby **E** (np. **10E13**).

```
projekt1> 0.5*(7-9)
Wynik: -1.000000
projekt1> 2e6*1/9
Wynik: 222222.222222
projekt1>
```

Jeśli popełnimy błąd w konstruowanym zapytaniu, to program poinformuje nas o tym, gdzie błąd wystąpił.

```
projekt1> 1+(*8-0)
Blad:       ^
projekt1> 5/a3
Blad:      ^
projekt1> 678.3*(6-(1/9)
Blad:                  ^
projekt1>
```

Zamiast podawać wyrażenia za pomocą klawiatury, możemy zapisać je wcześniej do pliku i uruchomić program z opcją **/iPLIK.TXT**. Wtedy program sam odczyta wyrażenia z tego pliku i obliczy ich wartości. Podobnych opcji możemy użyć do zapisywania wyników do pliku. Szczegółowe informacje na ten temat ujrzymy po wydaniu polecenia **projekt1 /?**:

```
C:\projekt1>projekt1 /?
Wywolywanie programu: projekt1 [/iPLIK] [/oPLIK] [/lPLIK] [/m]
[/?]

Wszystkie argumenty w nawiasach kwadratowych sa opcjonalne.
Znaczenie poszczegolnych argumentow:
/iPLIK  - pobiera dane z pliku PLIK zamiast z klawiatury
/oPLIK  - zapisuje wyjscie do pliku PLIK, zamiast wyswietlac
          je na ekranie
/lPLIK  - do pliku PLIK zapisywane sa wszystkie operacje
          nieprawidlowe oraz przebieg analizy operacji
          prawidlowych w odwrotnej notacji polskiej
/m      - w razie bledu na ekranie wyswietlane jest samo wyrazenie
/?      - wyswietlenie tego ekranu pomocy
```

Aby sprawdzić jakich dodatkowych poleceń możemy użyć w czasie pracy z programem, należy wydać polecenie **?**. Działanie programu kończymy wpisując **!** zamiast wyrażenia.

8. Listing programu

```
#include <iostream>

#include <iomanip>

#include <fstream>

#include <math.h>

#include <string>


using namespace std;


enum State {StateStart, StateOperand, StateNumber, StateNumberPoint,
StateNumberMantisa, StateNumberPowerSign, StateNumberPowerMinus,
StateNumberPower, StateOperator, StateEnd, StateError};

enum Input {Eof, Space, Digit, Dot, E, Minus, Plus, Mul, OBr, CBr};

enum Ops { OpSub, OpAdd, OpMul, OpDiv, OpPow, OpNeg };


bool ErrorPrintExpression = false;


ostream * myout = &cout;

ostream * myerr = (ostream *) NULL;


const unsigned BuforPliku = 1024;


class Stos

{
```

```
      public:
            class Element
            {
                  private:
                        double v;
                        int w;
                        Element * next;
                  public:
                        double vGet() const { return v;}
                        int wGet() const { return w;}
                        Element * nextGet() const { return next;}
                        void wSet(int a)    { w = a; }
                        void vSet(double a) { v = a; }
                        void nextSet(Element * a) { next = a; }
                        Element & operator=(const Element &a);
                        Element():v(0), w(0), next(0) {};
                        Element (const Element & a);
                        Element (double a, int b): v(a), w(b), next(NULL) {}
                        void print() const { *myout << "v: " << v << ", w: " <<
w; }
                        friend ostream &operator<<(ostream &s,const
Stos::Element &a);
            };
            Stos(): rozmiar(0), poczatek(NULL) {}
            class Empty {};
            ~Stos();
            operator bool() const { return(poczatek); }
            Stos &operator<<(const Element a) { add(a); return(*this); }
            Stos &operator>>(Element & a) { rem(a); return(*this); }
            friend ostream &operator<<(ostream &s,const Stos &S);
            Element * elementPtr (unsigned i) const;
            unsigned operator()() const { return rozmiar; }
            Element remove(unsigned i);
            Element * last() const;
            double vGet(unsigned a) const;
            int    wGet(unsigned a) const;
      private:
            unsigned rozmiar;
```

```
            Element * poczatek;
            void rem (Element &a);
            void add (const Element a);
};

double Stos::vGet(unsigned a) const
{
      Stos::Element * p = poczatek;
      for (unsigned c = 0; c != a; c++)
      {
            p = p->nextGet();
      }
      return p->vGet();
}

int Stos::wGet(unsigned a) const
{
      Stos::Element * p = poczatek;
      for (unsigned c = 0; c != a; c++)
      {
            p = p->nextGet();
      }
      return p->wGet();
}

Stos::Element * Stos::last() const
{
      if (!rozmiar)
      {
            return(NULL);
      } else {
            return elementPtr(operator()() - 1);
      }
}

Stos::Element Stos::remove (unsigned i)
{
      Element zwr;
```

```
	if (rozmiar == 0)
	{
		return zwr;
	}
	if (i == 0)
	{
		Element * a = poczatek;
		poczatek = a->nextGet();
		zwr = *a;
		delete a;
	} else if (i >= (operator()() - 1)) {
		Element * a = elementPtr(operator()() - 2);
		zwr = *(a->nextGet());
		delete a->nextGet();
		a->nextSet(NULL);
	} else {
		Element * toDelete = elementPtr(i);
		Element * before = elementPtr(i-1);
		before->nextSet(toDelete->nextGet());
		zwr = *toDelete;
		delete toDelete;
	}
	--rozmiar;
	return zwr;
}

Stos::Element * Stos::elementPtr (unsigned i) const
{
	Stos::Element * p = poczatek;
	for (unsigned c = 0; c != i; c++)
	{
		p = p->nextGet();
	}
	return p;
}

Stos::Element::Element(const Element & a): v(a.vGet()), w(a.wGet()),
next(a.nextGet())
```

```
{
}

Stos::Element & Stos::Element::operator=(const Element & a)
{
	w = a.w;
	v = a.v;
	next = a.next;
	return(*this);
}

Stos::~Stos()
{
	if (rozmiar == 0) return;
	return;
	while (poczatek)
	{
		Element *t = poczatek->nextGet();
		delete poczatek;
		poczatek = t;
	}
}

void Stos::add(const Element a)
{
	Element * N = new Element;
	(*N) = a;
	N->nextSet(NULL);
	if (rozmiar == 0)
	{
		poczatek = N;
	} else {
		Element * b = last();
		b->nextSet(N);
	}
	rozmiar++;
	return;
}
```

```
void Stos::rem(Element &a)
{
	if (!rozmiar) throw Empty();
	Element B = *(elementPtr(operator()() - 2));
	delete B.nextGet();
	B.nextSet(NULL);
	rozmiar--;
}

ostream &operator<<(ostream &s,const Stos::Element &a)
{
	return(s << "v: " << a.v << ", w: " << a.w);
}

ostream &operator<<(ostream &s,const Stos &S)
{
	s<<'{';
	Stos::Element *i=S.poczatek;
	if(i)
	{
		while(true)
		{
			s << '(' << (*i) << ')';
			i=i->nextGet();
			if(!i)break;
			s<<',';
		}
	}
	return(s<<'}');
}

struct UnivArgs
{
	Stos & stosA;
	Stos & stosB;
	unsigned i;
	unsigned & Position;
```

```
	int & nawiasy;
	char * wyrazenie;
};

struct TabRecord
{
	State NextState;
	void (*funkcja)(UnivArgs);
};

void ErrorDivByZero(void)
{
	*myout << "Blad: proba dzielenia przez zero!" << endl;
	return;
}

void CmdClose(UnivArgs a)
{
	a.nawiasy -= 5;
	return;
}

void CmdOperator(UnivArgs a)
{
	double v = 0;
	int w = a.nawiasy;
	switch ((a.wyrazenie)[a.i])
	{
		case '-': w += 1; v = OpSub; break;
		case '+': w += 1; v = OpAdd; break;
		case '*': w += 2; v = OpMul; break;
		case '/': w += 2; v = OpDiv; break;
		case '^': w += 3; v = OpPow; break;
	}

	for (unsigned i = (a.stosB() - 1); (int)i > -1;)
	{
		if ((a.stosB.wGet(i) != -1) && (a.stosB.wGet(i) >= w))
```

```
            {
                  a.stosA << a.stosB.remove(i--);
            } else {
                  --i;
            }
      }
      a.stosB << Stos::Element(v, w);
      return;
}

void CmdNumber(UnivArgs a)
{
      unsigned from = a.Position;
      char buf[a.i - from + 1];
      unsigned j = 0;
      while (from < a.i)
      {
            buf[j++] = *(a.wyrazenie+(from++));
      }
      buf[j] = '\0';
      a.stosA << Stos::Element(strtod(buf, (char**)'\0'), -1);
      return;
}

void CmdEnd(UnivArgs a)
{
      for (unsigned i = (a.stosB() - 1); (int)i > -1;)
      {
            if (a.stosB.wGet(i) != -1)
            {
                  a.stosA << a.stosB.remove(i--);
            } else {
                  --i;
            }
      }
      return;
}
```

```
void CmdOpen(UnivArgs a)
{
	a.nawiasy += 5;
	return;
}

void CmdMinus(UnivArgs a)
{
	a.stosB << Stos::Element(OpNeg, 4 + a.nawiasy);
	return;
}

void CmdSaveStartPos(UnivArgs a)
{
	a.Position = a.i;
	return;
}

void StosWydruk(Stos & S)
{
	unsigned rozm = S() - 1;
	for (unsigned i = 0; i <= rozm; ++i)
	{
		if (S.wGet(i) == -1)
		{
					*myerr << "push " << S.vGet(i) << endl;
		} else {
			if (S.vGet(i) == OpSub)
			{
						*myerr << "sub" << endl;
			}
			else if (S.vGet(i) == OpAdd)
			{
						*myerr << "add" << endl;
			}
			else if (S.vGet(i) == OpMul)
			{
						*myerr << "mul" << endl;
```

```
                }
                else if (S.vGet(i) == OpDiv)
                {
                                        *myerr << "div" << endl;
                }
                else if (S.vGet(i) == OpPow)
                {
                                        *myerr << "pow" << endl;
                }
                else if (S.vGet(i) == OpNeg)
                {
                                        *myerr << "neg" << endl;
                }
        }
    }
    return;
}

char * PobierzWyrazenie(void)
{
    char * wyrazenie = new char[1];
    unsigned rozmWyr = 0;

    while(true)
    {
        char buf[10];

        cin.getline(buf, sizeof(buf));
        unsigned rozmBuf = (unsigned) strlen(buf);

        char * noweWyr = new char [rozmWyr + rozmBuf + 1];

        memcpy(noweWyr, wyrazenie, rozmWyr);
        memcpy(noweWyr + rozmWyr, buf, rozmBuf);
        rozmWyr += rozmBuf;
        delete[] wyrazenie;
        wyrazenie = noweWyr;
        if((signed)rozmBuf < cin.gcount()) break;
```

```
            cin.clear();
      }

      wyrazenie[rozmWyr] = '\0';
      return(wyrazenie);
}

unsigned PoliczLinijki(char * Plik)
{
      fstream fi;
      fi.open(Plik,ios::in);
      unsigned i = 0;
      while(fi)
      {
            char Bufor[BuforPliku];
            fi.getline(Bufor,BuforPliku);
            i++;
      }
      fi.close();
      return --i;
}

bool WartoscWyrazu(Stos & A, Stos & B)
{
      for (unsigned i = (A() - 1); (int)i > -1 ; --i)
      {
            B << A.remove(i);
      }
      for (unsigned i = (B() - 1); (int)i > -1; B.remove(i), --i)
      {
            if (B.wGet(i) == -1)
            {
                  A << *(B.elementPtr(i));
            } else {
                  unsigned j = A() - 1;
                  if (B.vGet(i) == OpNeg)
                  {
                        A.elementPtr(j)->vSet(-(A.elementPtr(j)->vGet()));
```

```
                } else {
                        double wynik;
                        double b = A.vGet(j);
                        double a = A.vGet(j-1);
                        if (B.vGet(i) == OpSub) { wynik = a - b; }
                        else if (B.vGet(i) == OpAdd) { wynik = a + b; }
                        else if (B.vGet(i) == OpMul) { wynik = a * b; }
                        else if (B.vGet(i) == OpDiv) {
                                if ( b == 0 )
                                {
                                        ErrorDivByZero();
                                        return(false);
                                }
                                wynik = a / b;
                        }
                        else if (B.vGet(i) == OpPow) { wynik = pow(a, b); }
                        A.remove(j--);
                        A.remove(j--);
                        A << Stos::Element(wynik, -1);
                }
        }
    }
    return(true);
}

void ErrorOnPosition(unsigned i, char * wyrazenie)
{
    if (!ErrorPrintExpression)
    {
        ostream * out;
        if (myerr != NULL)
        {
            out = myerr;
            *out << "projekt1> " << wyrazenie << endl;
        } else {
            out = &cout;
        }
        *out << "Blad:     ";
```

```
            for (unsigned j = 0; j < i; j++) *out << " ";
            *out << "^" << endl;
      } else {
            cout << wyrazenie << endl;
      }
}

void CmdNone(UnivArgs a)
{
      return;
}

void CmdNumberAndEnd(UnivArgs a)
{
      CmdNumber(a);
      CmdEnd(a);
      return;
}

void CmdNumberAndOperator(UnivArgs a)
{
      CmdNumber(a);
      CmdOperator(a);
      return;
}

void CmdNumberAndClose(UnivArgs a)
{
      CmdNumber(a);
      CmdClose(a);
      return;
}

bool KonwertujWyrazenie(char * wyrazenie, Stos & stosA, Stos & stosB, TabRecord
Tablica[][CBr + 1])
{
      State stan = StateStart;
      unsigned rozmWyr = (unsigned) strlen(wyrazenie);
```

```
int nawiasy = 0;
unsigned Position = 0;
unsigned i = 0;

for (; i <= rozmWyr; i++)
{
	Input in;
	switch (wyrazenie[i])
	{
		case '\0':
			in = Eof; break;
		case ' ':
		case '\t':
			in = Space; break;
		case '0':
		case '1':
		case '2':
		case '3':
		case '4':
		case '5':
		case '6':
		case '7':
		case '8':
		case '9':
			in = Digit; break;
		case '.':
			in = Dot; break;
		case 'E':
		case 'e':
			in = E; break;
		case '-':
			in = Minus; break;
		case '+':
			in = Plus; break;
		case '*':
		case '/':
		case '^':
			in = Mul; break;
```

```
                case '(':
                        in = OBr; break;
                case ')':
                        in = CBr; break;
                default:
                        ErrorOnPosition(i, wyrazenie);
                        return(false);
                        break;
            }

            UnivArgs argumenty = {stosA, stosB, i, Position, nawiasy,
wyrazenie};
            Tablica[stan][in].funkcja(argumenty);
            stan = Tablica[stan][in].NextState;
            if ((nawiasy < 0) || (stan == StateError))
            {
                  ErrorOnPosition(i, wyrazenie);
                  return(false);
            }
      }
      if (nawiasy != 0)
      {
            ErrorOnPosition(i - 1, wyrazenie);
            return(false);
      }
      return(true);
}

void Koniec(void)
{
      *myout << endl << "Dziekuje za korzystanie z programu." << endl;
      *myout << "Maciej Korzen <maciek@korzen.org>. Grupa IZ204." << endl;
      if (myout == &cout)
      {
            *myout << endl << "Nacisnij [Enter], aby zamknac okno." << endl;
            cin.get();
      }
      exit(0);
```

```
}

void Pomoc(void)
{
	*myout << "Przydatne komendy:" << endl;
	*myout << " ? - pomoc," << endl;
	*myout << " ! - wyjscie z programu," << endl;
	return;
}

char * OptionFind(int argc, char * argv [], char c)
{
	for (unsigned i = 1; i < (unsigned)argc; ++i)
	{
		if (argv[i][0] == '/' && argv[i][1] == c && argv[i][2] != '\0')
		{
			return &(argv[i][2]);
		}
	}
	return NULL;
}

bool HaveArgument(int argc, char * argv [], char c)
{
	for (unsigned i = 1; i < (unsigned)argc; ++i)
	{
		if (argv[i][0] == '/' && argv[i][1] == c && argv[i][2] == '\0')
		{
			return true;
		}
	}
	return false;
}

void PrintHelp(ostream * out)
{
	*out << "Wywolywanie programu: projekt1 [/iPLIK] [/oPLIK] [/lPLIK] [/m]
[/?]" << endl << endl
```

```
            << "Wszystkie argumenty w nawiasach kwadratowych sa opcjonalne." <<
endl
            << "Znaczenie poszczegolnych argumentow:" << endl
            << "/iPLIK  - pobiera dane z pliku PLIK zamiast z klawiatury" << endl
            << "/oPLIK  - zapisuje wyjscie do pliku PLIK, zamiast wyswietlac" <<
endl
            << "          je na ekranie" << endl
            << "/lPLIK  - do pliku PLIK zapisywane sa wszystkie operacje" << endl
            << "          nieprawidlowe oraz przebieg analizy operacji" << endl
            << "          prawidlowych w odwrotnej notacji polskiej" << endl
            << "/m      - w razie bledu na ekranie wyswietlane jest samo
wyrazenie" << endl
            << "/?      - wyswietlenie tego ekranu pomocy" << endl;
      return;
}

int main(int argc, char * argv[])
{
      fstream wejscie, wyjscie, blad;
      char * PlikZapis  = OptionFind(argc, argv, 'o');
      if (PlikZapis != NULL)
      {
            wyjscie.open(PlikZapis, ios::out);
            if (!wyjscie)
            {
                  *myout << "Blad! Nie moge pisac do pliku wyjsciowego!" <<
endl;
                  exit(1);
            }
            myout = &wyjscie;
      }
      char * PlikBlad  = OptionFind(argc, argv, 'l');
      if (PlikBlad != NULL)
      {
            blad.open(PlikBlad, ios::out);
            if (!blad)
            {
                  *myout << "Blad! Nie moge pisac do pliku bledu!" << endl;
                  exit(1);
```

```
            }
            myerr = &blad;
      }
      if (HaveArgument(argc, argv, '?'))
      {
            PrintHelp(myout);
            exit(0);
      }
      *myout << "Witamy w programie \'projekt1\'. Zyczymy milej pracy." << endl;
      Pomoc();

      char prompt[] = "projekt1> ";
      char * PlikOdczyt = OptionFind(argc, argv, 'i');
      ErrorPrintExpression = HaveArgument(argc, argv, 'm');
      if (PlikOdczyt != NULL)
      {
            wejscie.open(PlikOdczyt, ios::in);
            if (!wejscie)
            {
                  *myout << "Blad! Nie moge odczytac pliku wejsciowego!" <<
endl;
                  exit(1);
            }
      }

      TabRecord Tablica[StateError + 1][CBr + 1] =
      {
            { {StateError, &CmdNone},     {StateStart, &CmdNone},
      {StateNumber, &CmdSaveStartPos},    {StateNumberPoint, &CmdSaveStartPos},
      {StateError, &CmdNone},             {StateOperand, &CmdMinus},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateStart, &CmdOpen}, {StateError, &CmdNone} },
            { {StateError, &CmdNone},     {StateOperand, &CmdNone},
      {StateNumber, &CmdSaveStartPos},    {StateNumberPoint, &CmdSaveStartPos},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateStart, &CmdOpen}, {StateError, &CmdNone} },
            { {StateEnd, &CmdNumberAndEnd},     {StateOperator, &CmdNumber},
      {StateNumber, &CmdNone},            {StateNumberMantisa, &CmdNone},
      {StateNumberPowerSign, &CmdNone},   {StateOperand, &CmdNumberAndOperator},
      {StateOperand, &CmdNumberAndOperator},    {StateStart,
&CmdNumberAndOperator}, {StateError, &CmdNone}, {StateOperator,
```

```
&CmdNumberAndClose} },

            { {StateError, &CmdNone},     {StateError, &CmdNone},
      {StateNumberMantisa, &CmdNone},           {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone}, {StateError, &CmdNone} },

            { {StateEnd, &CmdNumberAndEnd},     {StateOperator, &CmdNumber},
      {StateNumberMantisa, &CmdNone},           {StateError, &CmdNone},
      {StateNumberPowerSign, &CmdNone},   {StateOperand, &CmdNumberAndOperator},
      {StateOperand, &CmdNumberAndOperator},    {StateStart,
&CmdNumberAndOperator}, {StateError, &CmdNone}, {StateOperator,
&CmdNumberAndClose} },

            { {StateError, &CmdNone},     {StateError, &CmdNone},
      {StateNumberPower, &CmdNone},       {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateNumberPowerMinus, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone}, {StateError, &CmdNone} },

            { {StateError, &CmdNone},     {StateError, &CmdNone},
      {StateNumberPower, &CmdNone},       {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone}, {StateError, &CmdNone} },

            { {StateEnd, &CmdNumberAndEnd},     {StateOperator, &CmdNumber},
      {StateNumberPower, &CmdNone},       {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateOperand, &CmdNumberAndOperator},
      {StateOperand, &CmdNumberAndOperator},    {StateStart,
&CmdNumberAndOperator}, {StateError, &CmdNone}, {StateOperator,
&CmdNumberAndClose} },

            { {StateEnd, &CmdEnd},        {StateOperator, &CmdNone},
      {StateError, &CmdNone},             {StateError, &CmdNone},
      {StateError, &CmdNone},             {StateOperand, &CmdOperator},
      {StateOperand, &CmdOperator},       {StateStart, &CmdOperator},
      {StateError, &CmdNone}, {StateOperator, &CmdClose} }

      };


      while(true)

      {

            unsigned Precyzja = 6;

            *myout << prompt;

            if (PlikZapis != NULL) { *myout << endl; }

            char * wyrazenie = NULL;

            if (PlikOdczyt != NULL)

            {

                  wyrazenie = new char[BuforPliku];

                  memset(wyrazenie, 0, sizeof(char) * BuforPliku);

                  wejscie.getline(wyrazenie, BuforPliku);

                  if (wejscie.eof()) Koniec();
```

```
                *myout << wyrazenie << endl;
        } else {
                wyrazenie = PobierzWyrazenie();
        }
        if (wyrazenie[1] == '\0')
        {
                if (wyrazenie[0] == '!')
                {
                        Koniec();
                } else if (wyrazenie[0] == '?') {
                        Pomoc();
                        continue;
                }
        }
        Stos stosA = Stos();
        Stos stosB = Stos();
        if (KonwertujWyrazenie(wyrazenie, stosA, stosB, Tablica) == false)
                continue;
        if (myerr != NULL) { *myerr << wyrazenie << endl; }
        myout->setf(ios::fixed);
        if (myerr != NULL) StosWydruk(stosA);
        if (WartoscWyrazu(stosA, stosB) == false) continue;
        double w = stosA.vGet(0);
        *myout << setprecision(Precyzja) << "Wynik: " <<  w << endl;
        if (myerr != NULL) { *myerr << '=' << endl << w << endl; }
    }
    return(0);
}
```