ALGORYTMY I STRUKTURY DANYCH

PROJEKT 1 - ODWROTNA NOTACJA POLSKA

Artur Guniewicz

Projekt 1. (25 pkt)

Odwrotna notacja Polska (ONP):

-konwersja wyrażenia z notacji tradycyjnej do ONP
(na wejściu mamy wyrażenie zapisane w notacji tradycyjnej (nawiasowej), na wyjściu: wyrażenie zapisane w ONP;
np.: Wejście: 2*(3+3) Wyjście: 2 3 3 + *)

-obliczanie wartości wyrażenia w ONP.
(na wejściu mamy wyrażenie zapisane w ONP, na wyjściu: wynik obliczeń;
np.: wejście: 2 3 3 + * wyjście: 12)

Dla uproszczenia zakładamy, ze w wyrażeniach mogą pojawiać się tylko liczby całkowite, nawiasy "(" (otwierający), ")" (zamykający) oraz operatory "+" (dodawanie), "-" (odejmowanie), "*" (mnożenie), "/" , (dzielenie całkowite), "^" (potęgowanie),"~" minus unarny.
Składniki wyrażeń oddzielone są pojedynczą spacją. Wyrażenia zapisane są bezbłędnie. Kolejność wykonywania działań (priorytet operatorów):
1. potęgowanie
2. mnożenie, dzielenie całkowite
3. dodawanie i odejmowanie

Odwrotna Notacja Polska (ONP) - to sposób zapisu wyrażenia arytmetycznego w taki sposób, aby operator występował po argumentach (operandach). Nie występują w niej także nawiasy.

Algorytmy obliczania wartości wyrażenia w ONP oraz konwersji z notacji tradycyjnej do odwrotnej notacji polskiej oparłem na strukturze danych jaką jest stos (ang. stack). Jest to tzw lista lifo, co oznacza, że wymiana elementów, czyli ich dodawanie, bądź usuwanie odbywa się w obrębie jej wierzchołka, czyli na początku listy.

W moim programie zaimplementowałem stos tablicowo tworząc klasę Stos z następującymi metodami:

- Stos() - konstruktor tworzący pusty stos
- Push() - umieszcza zadany element na szczycie stosu
- Pop() - usuwa element na szczycie stosu
- TopElem() - zwraca element ze szczytu stosu
- Empty() - sprawdza czy stos jest pusty
- Makenul() - czyści stos
- Print() - wypisuje wszystkie elementy stosu w kolejności

Kod programu:

1. projekt.cpp - plik główny

```
/*
************************************************************
*                Algorytmy i Struktury Danych              *
*                                                          *
*                      Artur Guniewicz                     *
************************************************************
*/

#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <stdlib.h>
#include <string>

#include "klasaStos.cpp"
#include "convertToRPN.cpp"
#include "calculateRPN.cpp"

using namespace std;

int main()
{
menu:

                                                cout                 <<
"************************************************************" << endl
              <<  "                                ODWROTNA  NOTACJA  POLSKA
" << endl
```

```
                                                                    <<
"************************************************************" << endl
        << endl;

    cout << "Co chcesz zrobić?" << endl
         << "1. Przekonwertować wyrażenie z notacji tradycyjnej do ONP"
<< endl
        << "2. Obliczyć wyrażenie w ONP" << endl
        << "3. Zakończyć program" << endl
        << "Twój wybór: ";

    int choice;
    char choice2;

    cin >> choice;

    cout << endl;

    string expression;

    switch (choice)
    {
    case 1:
        cin.ignore();
        cout << "Podaj wyrażenie w notacji tradycyjnej: ";
        getline(cin, expression);
        cout << "To samo wyrażenie w ONP: ";
        convertToRPN(expression);
        break;

    case 2:
        cin.ignore();
        cout << "Podaj wyrażenie w ONP: ";
        getline(cin, expression);
        cout << "Wynik: ";
        calculateRPN(expression);
        break;

    case 3:
        cout << "Miłego dnia!" << endl;
```

```
        exit(1);
        break;

    default:
        system("clear");
         cout << "Błędnie podana cyfra!" << endl << "Proszę podać cyfrę
1, 2 albo 3..." << endl << endl;
        goto menu;
        break;
    }


    cout << endl << "Zakończyć program? (T/N): ";

end:
    cin >> choice2;

    switch(choice2)
    {
    case 'T':
    case 't':
        cout << endl << "Miłego dnia!" << endl;
        exit(1);

    case 'N':
    case 'n':
        cout << endl;
        system("clear");
        goto menu;

    default:
        cout << "Wpisz <T> aby zakończyć lub <N> aby wrócić do menu: ";
        goto end;
    }

    return 0;
}
```

2. klasaStos.cpp - zawiera implementację stosu

```cpp
#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

const int maxlegth = 100;

typedef int elementtype;
typedef int position;

class Stos
{

protected:
   int S[maxlegth];
   position Top; //szczyt stosu

public:
   Stos();
   ~Stos();

   void Push(elementtype x);
   void Pop();
   elementtype TopElem();
   bool Empty();
   void Makenul();
   void Print();
};

Stos::Stos(void)
{
   Top = -1;
}

void Stos::Push(elementtype x)
{
```

```
    if (Top < maxlegth - 1)
    {
        Top++;
        S[Top] = x;
    }
}

void Stos::Pop()
{
    if (Top >= 0)
        Top--;
}

elementtype Stos::TopElem()
{
    if (Top >= 0)
        return S[Top];
}

bool Stos::Empty()
{
    if (Top == -1)
        return true;
    else
        return false;
}

void Stos::Makenul()
{
    Top = -1;
}

void Stos::Print()
{
    int x = Top;
    while (x != -1)
    {
        cout << S[x] << " ";
        x--;
    }
```

```
    cout << endl;
}

// poza klasą - sprawdza czy to operator
bool IsOperator(char c)
{
    switch (c)
    {
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
    case '~':
        return true;
        break;

    default:
        return false;
        break;
    }
}
```

3. convertToRPN.cpp - konwersja notacji tradycyjnej do ONP

```
#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cctype>
#include <string>

using namespace std;

int priority(elementtype el)
{
    switch((char)el)
    {
    case '+':
```

```
    case '-':
        return 1;
        break;

    case '*':
    case '/':
        return 2;
        break;

    case '^':
        return 3;
        break;

    case '(':
        return 0;
        break;

    case '~':
        return 4;
        break;

    default:
        return -1;
        break;
    }
}

void convertToRPN(string expression)
{
    int i = 0;

    Stos *S = new Stos;

    string digit;

    while (i < expression.length())
    {
        // ignoruje spacje
        while(isspace(expression[i]))
            i++;
```

```
        // sprawdza czy to liczba
        if(isdigit(expression[i]))
        {
            while(isdigit(expression[i]))
            {
                digit += expression[i];
                i++;
            }

            cout << digit << " ";
            digit = "";
            i--;
        }

        // sprawdza czy to operator
        else if(IsOperator(expression[i]))
        {
            // kontrola minusa unarnego
              if ((expression[i] == '-' && i == 0) || (expression[i] ==
'-' && isdigit(expression[i + 1]) && expression[i - 1] == '('))
            {
                int j = i + 1;
                int counter = 0;

                while (isdigit(expression[j]))
                {
                    digit += expression[j];
                    j++;
                    counter++;
                }

                cout << digit << " ~ ";
                digit = "";

                i += counter;
            }

            else
            {
```

```
                while (priority(S->TopElem()) >= priority(expression[i])
&& !(S->Empty()))
                {
                    cout << (char)S->TopElem() << " ";
                    S->Pop();
                }

                S->Push(expression[i]);
            }
        }

        else if(expression[i] == '(')
        {
            S->Push(expression[i]);
        }

        else if(expression[i] == ')')
        {
            while((char)S->TopElem() != '(')
            {
                cout << (char)S->TopElem() << " ";
                S->Pop();
            }

            S->Pop();
        }

        i++;
    }

    // czyści stos z pozostałych operatorów
    while (!(S->Empty()))
    {
        cout << (char)S->TopElem() << " ";
        S->Pop();
    }

    cout << endl;
}
```

4. calculateRPN.cpp - oblicza wartość wyrażenia zapisanego w ONP

```
#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cctype>
#include <string>

using namespace std;

int calculateRPN(string expression)
{
   int i = 0;
   elementtype a, b, result;

   Stos *S = new Stos;

   string digit = "";

   while(i < expression.length())
   {
       // ignoruje spacje
       while(isspace(expression[i]))
           i++;

       // sprawdza czy to liczba
       if (isdigit(expression[i]))
       {
           while (isdigit(expression[i]))
           {
               digit += expression[i];
               i++;
           }

           S->Push(stoi(digit.c_str()));
           digit = "";
       }
```

```
        // sprawdza czy i jaki to operator
        else if(IsOperator(expression[i]))
        {
            if (expression[i] == '+')
            {
                a = S->TopElem();
                S->Pop();
                b = S->TopElem();
                S->Pop();

                result = (b + a);
            }

            if (expression[i] == '-')
            {
                a = S->TopElem();
                S->Pop();
                b = S->TopElem();
                S->Pop();

                result = (b - a);
            }

            if (expression[i] == '*')
            {
                a = S->TopElem();
                S->Pop();
                b = S->TopElem();
                S->Pop();

                result = (b * a);
            }

            if (expression[i] == '/')
            {
                a = S->TopElem();
                S->Pop();
                b = S->TopElem();
                S->Pop();
```

```
                result = (b / a);
            }

            if (expression[i] == '^')
            {
                a = S->TopElem();
                S->Pop();
                b = S->TopElem();
                S->Pop();

                result = pow(b, a);
            }

            if (expression[i] == '~')
            {
                a = S->TopElem();
                S->Pop();

                result = -a;
            }

            i++;

            // dopisuje wynik na szczyt stosu
            S->Push(result);
        }

        else
        {
            cout << "Błędne wyrażenie!" << endl;
            break;
        }
    }

    cout << S->TopElem() << endl;
    S->Makenul();
}
```

Kompilacja:

g++ projekt.cpp -o projekt && ./projekt

(Warunek: wszystkie 4 pliki muszą znajdować się w tym samym folderze)

Wyniki:

Przykłady konwersji:

```
Podaj wyrażenie w notacji tradycyjnej: ( 15 - 3 ) ^ ( 3 + 2 ) * 6 /  3
To samo wyrażenie w ONP: 15 3 - 3 2 + ^ 6 * 3 /
```

```
Podaj wyrażenie w notacji tradycyjnej: (-1+2)/3*(4-5)
To samo wyrażenie w ONP: 1 ~ 2 + 3 / 4 5 - *
```

```
Podaj wyrażenie w notacji tradycyjnej: ( ( 5 - 2 ) ^ ( 3 + 1 ) / ( 2 + 1 ) + 12 ) * 21
To samo wyrażenie w ONP: 5 2 - 3 1 + ^ 2 1 + / 12 + 21 *
```

Przykłady obliczania wartości:

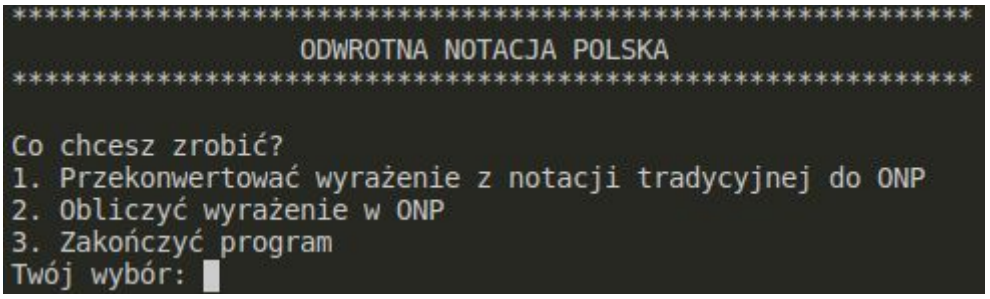
```
Podaj wyrażenie w ONP: 7 3 + 5 2 - 2 ^ *
Wynik: 90
```

```
Podaj wyrażenie w ONP: 12 2 3 4 * 10 5 / + * +
Wynik: 40
```

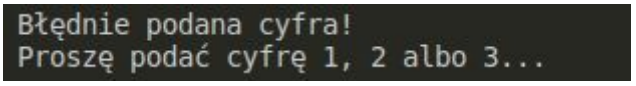
```
Podaj wyrażenie w ONP: 1 ~ 2 + 8 ~ 2 - *
Wynik: -10
```

Działanie programu:

Po uruchomieniu programu głównego “projekt.cpp” ukazuje się menu wyboru pozwalające zdecydować czy chcemy obliczyć wartość wyrażenia w ONP, dokonać konwersji z notacji tradycyjnej do ONP, czy też zakończyć program.

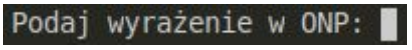


Podanie błędnej cyfry (innej niż 1, 2 lub 3) spowoduje wyświetlenie menu ponownie z komunikatem o błędnie wprowadzonych danych.



Wybór 3 zakończy program z komunikatem “Miłego dnia!”.

Wybór 2 umożliwi wpisanie z klawiatury wyrażenia. Można je wpisać z dowolną ilością spacji, a także i bez ich użycia, gdyż program posiada kontrolę znaków białych i nie powinien popełniać błędów.



Wpisanie błędnego wyrażenia, spowoduje wyświetlenie o tym komunikatu i umożliwi ponowne wprowadzenie danych.

Metoda umożliwiająca wykonanie obliczeń znajduje się w pliku “calculateRPN.cpp”. Działa ona następująco:
Na początku tworzony jest pusty stos S, zmienna digit, która będzie zapamiętywać liczbę oraz parę zmiennych pomocniczych ułatwiających poruszanie się po stosie. Pętla while przechodzi po kolei po każdym znaku wprowadzonego przez nas wyrażenia i kończy się gdy skończą się znaki. Najpierw sprawdzane jest czy element nie jest spacją, jeśli tak -> pomijamy. Następnie sprawdzane jest czy element jest liczbą, jeśli tak to w zmiennej digit zapamiętywane są wszystkie cyfry danej liczby, a następnie po konwersji z typu string na int (stoi) wrzucana jest ona na stos, a digit czyszczony. Jeśli element jest operatorem (specjalna metoda w pliku “klasaStos.cpp” IsOperator()) to sprawdzane jest jaki to operator i wykonywane jest odpowiednie działanie arytmetyczne, a jego wynik wrzucany jest na stos.

Po przejściu całego wyrażenia na szczycie stosu znajduje się wynik, który wysyłany jest na wyjście, a stos jest czyszczony.

Następnie program wysyła zapytanie czy ma się zakończyć, czy chcemy kontynuować.

Wybór T (lub t) zakończy program, N (lub n) wyczyści konsolę i ponownie wyświetli menu, a wybór czegoś innego wyświetli komunikat z prośbą o podanie poprawnej litery.

Wybór 1 w menu także umożliwi wpisanie wyrażenia, ale tym razem w notacji tradycyjnej. Również można je wpisać używając spacji, bądź też nie.

Plik “convertToRPN.cpp” zawiera dwie metody: priority() oraz convertToRPN(). Ta pierwsza przyporządkowuje operatorom priorytety, co umożliwi późniejsze operowanie nimi na stosie.

OPERATOR	PRIORYTET
~	4
^	3
/ *	2
+ -	1
(	0

Natomiast druga dokonuje konwersji z notacji tradycyjnej do ONP. Działa ona tak:
Najpierw tworzy pusty stos S oraz zmienną digit jak wyżej. Początek metody jest taki sam jak metody calculateRPN() ale z większą kontrolą przechodzenia między elementami wyrażenia. Liczby zostają przepisane na wyjście, natomiast z operatorami jest sprawa bardziej złożona. Najpierw kontrolowany jest minus unarny: w momencie kiedy znak ‘-’ występuje jako pierwszy bądź pomiędzy znakiem ‘(‘, a liczbą program przepisuje na wyjście liczbę i ‘~’. Kiedy jest to inny operator porównywane są ich priorytety oraz sprawdzane jest to czy stos zawiera jakieś elementy. I teraz w zależności od warunków operatory przepisywane są na wyjście bądź dokładane na stos. Znak ‘(‘ jest od razu umieszczany na stosie, natomiast znak ‘)’ powoduje przepisanie ze stosu na wyjście wszystkich operatorów aż do

natrafienia na znak ‘(‘, który jest potem ze stosu usuwany. Następnie pobierany jest kolejny element i tak aż do końca wyrażenia. Na koniec, jeżeli w stosie zostały jakieś operatory, zostają one po kolei przepisane na wyjście.

Kiedy konwersja przebiegnie poprawnie program wyświetla zapytanie o zakończenie programu, gdyż ta część jest już napisana w pliku “projekt.cpp”.

Po zakończeniu wykonywania żądanego polecenia stos jest pusty, a zmienne gotowe do nadpisania, więc każdorazowo wybierając <N> żeby program się nie zakończył możemy teoretycznie w nieskończoność wykonywać kolejne operacje.

Komentarz:

Wszystkie 4 pliki zostaną dołączone do tego pliku. Program zawiera pewne opcje kontroli błędów, jednak z premedytacją da się go zawiesić. Dlatego prosiłbym więc o wykonywanie poleceń programu oraz wpisywanie poprawnych wyrażeń.