Odwrotna notacja polska w języku C++

Źródło: Guillaume Techer, domena publiczna.

Część języków programowania, a także kalkulatorów, używa do swoich obliczeń odwrotnej notacji polskiej. Wiesz już, na czym ona polega i czym różni się od tradycyjnego zapisu wyrażeń.

W tym e-materiale zaimplementujesz algorytm konwersji z notacji klasycznej do ONP w języku C++.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Odwrotna notacja polska w języku Java,
- Odwrotna notacja polska w języku Python.

Więcej zadań? Sięgnij do: Odwrotna notacja polska – zadania maturalne.

> **Twoje cele**
>
> - Wykorzystasz w praktyce sposób zapisywania wyrażeń arytmetycznych z zastosowaniem odwrotnej notacji polskiej.
> - Zaimplementujesz w języku C++ algorytm przekształcania wyrażenia arytmetycznego zapisanego z wykorzystaniem notacji infiksowej do postaci w ONP.
> - Rozwiążesz przykładowe zadania wymagające wykazania się znajomością ONP.

Film samouczek

Polecenie 1

Napisz program, który przekonwertuje wyrażenie arytmetyczne zapisane w notacji infiksowej na jego odpowiednik w odwrotnej notacji polskiej. Zauważ, że wyrażenie nie musi zawierać pełnego nawiasowania, zatem istotna jest tu kolejność wykonywania działań.

Przetestuj działanie programu dla wyrażenia arytmetycznego:

```
wyrazenie = "(a+b*c)/d="
```

Specyfikacja problemu:

Dane:

- `wyrazenie` – ciąg znaków, wyrażenie arytmetyczne zapisane w notacji infiksowej, gdzie dozwolonymi znakami są jednoliterowe nazwy zmiennych, operatory: dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/), potęgowania (^) oraz nawiasy okrągłe: otwierające (( ) i zamykające ( )); znak równości (=) należy interpretować jako koniec wyrażenia arytmetycznego

Wynik:

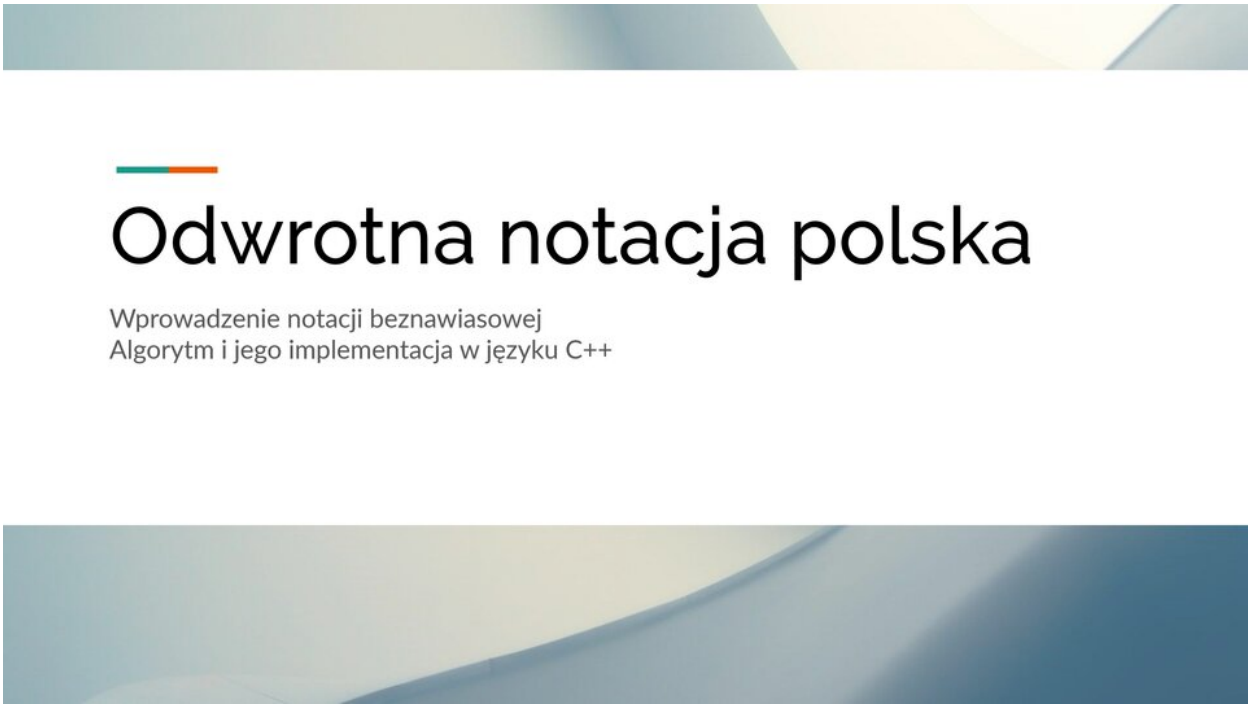
- wyrażenie arytmetyczne zamienione na ONP

```cpp
#include <iostream>

int main () {

using namespace std;

    // Miejsce na wpisanie kodu
}
```

```
1
```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/R1QhNKhLzgwVG

Film nawiązujący do odwrotnej notacji polskiej.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.36 KB w języku polskim

Polecenie 3

Na podstawie filmu podsumuj w notatce najważniejsze informacje dotyczące konwersji wyrażeń arytmetycznych zapisanych za pomocą notacji infiksowej do postaci postfiksowej.

Polecenie 4

Uruchom symulację i przekształć wyrażenie arytmetyczne zapisane w notacji **infiksowej** na **postfiksową**. Informacje teoretyczne dotyczące tego zagadnienia znajdziesz w e-materiale Odwrotna notacja polska.

Uwaga, w symulacji nie stosujemy pełnego nawiasowania.

Symulacja 1

W poniższej symulacji interaktywnej twoim zadaniem jest samodzielne przekształcenie wyrażenia arytmetycznego zapisanego w notacji **infiksowej** na notację **postfiksową.**

Czasem by przejść dalej, konieczne będzie dokonanie wyboru z podanych opcji. W przypadku slajdów wyjaśniających zawsze możesz przejść do kolejnego slajdu (opcja `Przejdź do kolejnego slajdu`) lub wrócić do poprzedniego slajdu (opcja `Wróć do poprzedniego slajdu`), natomiast w slajdach, gdzie konieczny jest wybór kolejnego kroku rozwiązania problemu, powrót do wcześniejszego slajdu jest możliwy, natomiast przejście do kolejnego już nie.

Od trzeciego slajdu możesz również zacząć od nowa i wrócić do pierwszego slajdu (opcja `Wróć do pierwszego slajdu i zacznij od nowa`).

> **Ważne!**
>
> W wyrażeniu zastosowaliśmy operator ^. Wykorzystujemy go jako operator potęgowania.

> **Przykład 1**
>
> **W jaki sposób czytać informacje w symulacji?**
>
> 1. Wyrażenie i pobierany element: `27 * 5 + 23 - (26 / (24 ^ 4))`
> 2. Stos: +
> 3. Przekształcone wyrażenie: `27 5 * 23`
> 4. Co należy zrobić z analizowanym elementem ?
> 5. Umieść element na stosie.
>   Stos: + -
>   Wyjście: 27 5 * 23
>
>   Dodaj element do wyniku.
>   Stos: +
>   Wyjście: 27 5 * 23 -
> 6. Wróć do poprzedniego slajdu
>
>   Wróć do pierwszego slajdu i zacznij od nowa

1

Zwróć uwagę na to, że element 23 jest wyróżniony. To on jest analizowany.

2

Ten element mówi, co w danym momencie znajduje się na stosie.

3

Tak wygląda dotychczas przekształcone wyrażenie.

4

Polecenia, które należy wykonać.

5

Pierwsze wiersze obu opcji wskazują kroki, które mają zostać wykonane, natomiast kolejne pokazują, jak powinny wyglądać stos oraz aktualny wynik po wykonaniu danego kroku.

6

Kolejne opcje pozwalają na nawigację po symulacji.

Symulacja

Polecenie 5

Zapisz przykład wyrażenia arytmetycznego w notacji infiksowej. Następnie przekaż go koleżance lub koledze, a od innej osoby weź zadanie dla siebie. Przeprowadź analizę otrzymanego wyrażenia, przekształcając je do postaci postfiksowej – w razie potrzeby skorzystaj z symulacji.

Słownik

pełne nawiasowanie

szczególne ustawienie nawiasów w wyrażeniu arytmetycznym w notacji infiksowej; dzięki niemu każde działanie ma jednoznacznie określone dwa operandy za pomocą ujęcia w nawiasy, a także całe działanie jest objęte nawiasem, np. `(a+(b·c))`

Przeczytaj

Polecenie 1

Przeanalizuj prezentację, która rozszerza program omówiony w filmie, umożliwiając użytkownikowi wprowadzanie operandów składających się z wielu cyfr.

Specyfikacja problemu:

Dane:

- `wyrazenie` – łańcuch znaków, pobrane od użytkownika wyrażenie zapisane w klasycznej notacji, które ma zostać przekształcone na swój odpowiednik w ONP

Wynik:

- wyrażenie arytmetyczne zamienione na ONP

1

W tej prezentacji spróbujemy ulepszyć algorytm zaprezentowany w poprzedniej sekcji. Sprawimy, że program będzie obsługiwał wyrażenia, których argumenty będą składać się z więcej niż jednego znaku. Na potrzeby rozwiązania przyjmijmy, że wyrażenie pobierzemy od użytkownika, a każdy z argumentów, operatorów, nawiasów będzie oddzielony przy pomocy spacji.

Przykładowe poprawne wyrażenia:

```
( a + b * c ) / d =
23 - 12 * ( 12 / 3 ) =
f * 12x / 2 =
```

2

Implementację rozwiązania rozpoczniemy od importu dwóch bibliotek: `iostream` oraz `list`. Zadaniem pierwszej jest obsługa strumieni wejścia/wyjścia. Druga pozwala na użycie struktury danych zwanej listą. Lista umożliwia symulację działania stosu, a także łatwiejsze przechowywanie elementów tworzących wyrażenie wprowadzone przez użytkownika.

Pamiętamy o ustawieniu przestrzeni nazw (`std`) oraz o przygotowaniu pustej funkcji `main`.

```
#include <iostream>
#include <list>

using namespace std;

int main(){

    return 1;
}
```

3

Przygotujmy funkcję `pobierzPriorytet`. Jej zadaniem będzie zwrócenie wartości liczbowej, która odpowiada priorytetowi operatora przekazanego jako argument. Zwróćmy uwagę na typ przekazanego argumentu: jest to napis. Tworzymy instrukcję warunkową, której zadaniem jest sprawdzenie, który z operatorów jest rozpatrywany. Wewnątrz każdej z podinstrukcji zwracamy wartość 1 (dla operatorów + i -), 2 (dla * i /) bądź 3 (dla operatora potęgowania ^).

```
#include <iostream>
#include <list>

using namespace std;
```

```
5
6  int pobierzPriorytet(string op){
7      if(op == "+" || op == "-"){
8          return 1;
9      }
10     else if(op == "*" || op == "/"){
11         return 2;
12     }
13     else if(op == "^"){
14         return 3;
15     }
16     else{
17         return 0;
18     }
19 }
20
21 int main(){
22
23     return 1;
24 }
```

4

Wewnątrz funkcji `main` tworzymy nową listę przechowującą napisy o nazwie `stos`. Deklarujemy napis `wyrazenie`, do którego za pomocą funkcji `getline` zapisujemy pobraną ze standardowego wejścia (`cin`) linijkę tekstu. Tworzymy drugą listę napisów: `elementyWyrazenia`. Będziemy w niej przechowywać poszczególne części przekazanego przez użytkownika wyrażenia, rozdzielone za pomocą znaku spacji.

```
1 #include <iostream>
2 #include <list>
3
4 using namespace std;
```

```
5
6  int
   pobierzPriorytet(string
   op){
7      if(op == "+" || op ==
   "-"){
8          return 1;
9      }
10     else if(op == "*" ||
   op == "/"){
11         return 2;
12     }
13     else if(op == "^"){
14         return 3;
15     }
16     else{
17         return 0;
18     }
19 }
20
21 int main(){
22     list<string> stos;
23
24     string wyrazenie;
25     getline(cin,
   wyrazenie);
26
27     list<string>
   elementyWyrazenia;
28
29     return 1;
30 }
```

5

Tworzymy zmienną pomocniczą `pozycjaSpacji`. Jej zadaniem będzie wskazanie indeksu, pod którym został umieszczony pierwszy znaleziony znak spacji. Umożliwi nam to podzielenie wyrażenia na części, a same operandy będą mogły składać się z więcej niż jednego znaku.

Dodajemy pętlę `while`. W jej nagłówku zapiszemy – do wcześniej stworzonej zmiennej `pozycjaSpacji` – indeks pierwszego wystąpienia spacji. Ciało pętli wykona się dopiero wtedy, gdy wartość zapisana w zmiennej `pozycjaSpacji` jest różna od stałej `string::npos`. Stała ta jest najczęściej wykorzystywana w celu pokazania, że wartość nie istnieje. Innymi słowy: pętla będzie wykonywać się, dopóki w badanym napisie istnieją spacje.

```
#include <iostream>
#include <list>

using namespace std;

int pobierzPriorytet(string op){
    if(op == "+" || op == "-"){
        return 1;
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;
```

```
28
29      int pozycjaSpacji = 0;
30
31      while((pozycjaSpacji =
    wyrazenie.find(" ")) !=
    string::npos){
32
33      }
34
35      return 1;
36  }
```

6

Wewnątrz pętli wykonujemy dwie operacje. Dodajemy do listy `elementyWyrazenia` fragment napisu, zaczynając od pierwszego znaku, aż do spacji (jej nie dodajemy). W drugim kroku, ten sam fragment, wraz ze znakiem spacji, usuwamy z napisu `wyrazenie`.

Poza pętlą, na koniec listy `elementyWyrazenia`, dołączamy pozostały fragment napisu `wyrazenie`.

```
 1  #include <iostream>
 2  #include <list>
 3
 4  using namespace std;
 5
 6  int
    pobierzPriorytet(string
    op){
 7      if(op == "+" || op ==
    "-"){
 8          return 1;
 9      }
10      else if(op == "*" ||
    op == "/"){
11          return 2;
12      }
13      else if(op == "^"){
14          return 3;
```

```
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin,
wyrazenie);

    list<string>
elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji =
wyrazenie.find(" ")) !=
string::npos){

 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
pozycjaSpacji));
        wyrazenie.erase(0,
pozycjaSpacji + 1);
    }

 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
string::npos));

    return 1;
}
```

Zaczynamy przetwarzać uzyskane we wcześniejszym kroku fragmenty wyrażenia. Piszemy pętlę `while`. Będzie się wykonywać,

dopóki lista `elementyWyrazenia` będzie posiadała jakiekolwiek elementy.

Do zmiennej `elementWyrazenia` zapisujemy pierwszy element listy. Uzyskujemy go, wywołując na liście metodę `front()`. Następnie ten sam element usuwamy z listy, wykorzystując metodę `pop_front()`.

```
#include <iostream>
#include <list>

using namespace std;

int pobierzPriorytet(string op){
    if(op == "+" || op == "-"){
        return 1;
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;

    int pozycjaSpacji = 0;

```

```
31      while((pozycjaSpacji =
wyrazenie.find(" ")) !=
string::npos){
32
 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
pozycjaSpacji));
33          wyrazenie.erase(0,
pozycjaSpacji + 1);
34      }
35
 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
string::npos));
36
37
 while(!elementyWyrazenia.
empty()){
38          string
elementWyrazenia =
elementyWyrazenia.front();
39
 elementyWyrazenia.pop_fro
nt();
40
41      }
42
43      return 1;
44  }
```

8

Badamy zawartość napisu `elementWyrazenia`. Jeżeli składa się on z pojedynczego symbolu "=", oznacza to, że przetworzyliśmy całe wyrażenie. Dopóki lista `stos` nie jest pusta, wypisujemy na standardowe wyjście operatory, zaczynając od tego znajdującego się najwyżej (znajduje się na końcu listy). W tym celu wykorzystujemy metodę `back`. Aby usunąć ostatni element z listy, wywołujemy metodę `pop_back()`.

```cpp
#include <iostream>
#include <list>

using namespace std;

int pobierzPriorytet(string op){
    if(op == "+" || op == "-"){
        return 1;
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji = wyrazenie.find(" ")) != string::npos){

     elementyWyrazenia.push_back(wyrazenie.substr(0, pozycjaSpacji));
        wyrazenie.erase(0, pozycjaSpacji + 1);
```

```
34      }
35
        elementyWyrazenia.push_back(wyrazenie.substr(0, string::npos));
36
37
        while(!elementyWyrazenia.empty()){
38              string elementWyrazenia = elementyWyrazenia.front();
39
        elementyWyrazenia.pop_front();
40
41
        if(elementWyrazenia == "="){
42
43
        while(!stos.empty()){
44                      string op = stos.back();
45                      cout << op << " ";
46
        stos.pop_back();
47                  }
48              }
49      }
50
51      return 1;
52  }
```

Rozpatrujemy kolejny przypadek – zmienna `elementWyrazenia` przechowuje operator. Piszemy pętlę `while`, która będzie wykonywać się tak długo, aż `stos` nie będzie pusty. Pętla może zostać przerwana wcześniej, gdy priorytet

operatora zapisanego w `elementWyrazenia` jest wyższy od tego, umieszczonego na szczycie stosu. Poza instrukcją warunkową, wewnątrz pętli, wypisujemy i zdejmujemy operator znajdujący się na szczycie stosu.

```
#include <iostream>
#include <list>

using namespace std;

int pobierzPriorytet(string op){
    if(op == "+" || op == "-"){
        return 1;
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji = wyrazenie.find(" ")) != string::npos){
```

```
32
 elementyWyrazenia.push_back(wyrazenie.substr(0, pozycjaSpacji));
33          wyrazenie.erase(0, pozycjaSpacji + 1);
34      }
35
 elementyWyrazenia.push_back(wyrazenie.substr(0, string::npos));
36
37
 while(!elementyWyrazenia.empty()){
38          string elementWyrazenia = elementyWyrazenia.front();
39
 elementyWyrazenia.pop_front();
40
41
 if(elementWyrazenia == "="){
42
43
 while(!stos.empty()){
44                  string op = stos.back();
45                  cout << op << " ";
46
 stos.pop_back();
47              }
48          }
49          else if(elementWyrazenia == "+" || elementWyrazenia == "-"
50                  || elementWyrazenia == "*" || elementWyrazenia == "/"
51                  || elementWyrazenia == "^"){
52
```

```
53
 while(!stos.empty()){
54
 if(pobierzPriorytet(eleme
ntWyrazenia) >
pobierzPriorytet(stos.back
())){
55                     break;
56                 }
57                 cout <<
stos.back() << " ";
58
 stos.pop_back();
59             }
60
 stos.push_back(elementWyr
azenia);
61         }
62     }
63
64     return 1;
65 }
```

10

Rozpatrzmy przypadek, gdy w zmiennej `elementWyrazenia` został zapisany znak nawiasu otwierającego "(". Jedyną operacją, jaką wykonamy, będzie umieszczenie go na szczycie stosu.

```
1 #include <iostream>
2 #include <list>
3
4 using namespace std;
5
6 int
pobierzPriorytet(string
op){
7     if(op == "+" || op ==
"-"){
8         return 1;
```

```
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji = wyrazenie.find(" ")) != string::npos){

 elementyWyrazenia.push_back(wyrazenie.substr(0, pozycjaSpacji));
        wyrazenie.erase(0, pozycjaSpacji + 1);
    }

 elementyWyrazenia.push_back(wyrazenie.substr(0, string::npos));


 while(!elementyWyrazenia.empty()){
        string elementWyrazenia =
```

```
elementyWyrazenia.front();
39
 elementyWyrazenia.pop_fro
nt();
40
41
 if(elementWyrazenia ==
"="){
42
43
 while(!stos.empty()){
44                 string op
= stos.back();
45                 cout << op
<< " ";
46
 stos.pop_back();
47             }
48         }
49         else
if(elementWyrazenia == "+"
|| elementWyrazenia == "-"
50                 ||
elementWyrazenia == "*" ||
elementWyrazenia == "/"
51                 ||
elementWyrazenia == "^"){
52
53
 while(!stos.empty()){
54
 if(pobierzPriorytet(eleme
ntWyrazenia) >
pobierzPriorytet(stos.back
())){
55                     break;
56                 }
57                 cout <<
stos.back() << " ";
58
 stos.pop_back();
59             }
60
 stos.push_back(elementWyr
azenia);
```

```
        }
        else
if(elementWyrazenia == "
("){
 stos.push_back("(");
        }
    }

    return 1;
}
```

11

Jeżeli natrafimy na znak nawiasu zamykającego ")", wypisujemy i zdejmujemy wszystkie operatory ze stosu, do czasu odnalezienia znaku nawiasu otwierającego. Nawias również zdejmujemy ze stosu.

```
#include <iostream>
#include <list>

using namespace std;

int
pobierzPriorytet(string
op){
    if(op == "+" || op ==
"-"){
        return 1;
    }
    else if(op == "*" ||
op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}
```

```
20
21 int main(){
22     list<string> stos;
23
24     string wyrazenie;
25     getline(cin,
   wyrazenie);
26
27     list<string>
   elementyWyrazenia;
28
29     int pozycjaSpacji = 0;
30
31     while((pozycjaSpacji =
   wyrazenie.find(" ")) !=
   string::npos){
32
    elementyWyrazenia.push_ba
   ck(wyrazenie.substr(0,
   pozycjaSpacji));
33         wyrazenie.erase(0,
   pozycjaSpacji + 1);
34     }
35
    elementyWyrazenia.push_ba
   ck(wyrazenie.substr(0,
   string::npos));
36
37
    while(!elementyWyrazenia.
   empty()){
38         string
   elementWyrazenia =
   elementyWyrazenia.front();
39
    elementyWyrazenia.pop_fro
   nt();
40
41
    if(elementWyrazenia ==
   "="){
42
43
    while(!stos.empty()){
```

```
44                  string op
= stos.back();
45                  cout << op
<< " ";
46
 stos.pop_back();
47              }
48          }
49          else
if(elementWyrazenia == "+"
|| elementWyrazenia == "-"
50                  ||
elementWyrazenia == "*" ||
elementWyrazenia == "/"
51                  ||
elementWyrazenia == "^"){
52
53
 while(!stos.empty()){
54
 if(pobierzPriorytet(eleme
ntWyrazenia) >
pobierzPriorytet(stos.back
())){
55                      break;
56                  }
57                  cout <<
stos.back() << " ";
58
 stos.pop_back();
59              }
60
 stos.push_back(elementWyr
azenia);
61          }
62          else
if(elementWyrazenia == "
("){
63
 stos.push_back("(");
64          }
65          else
if(elementWyrazenia ==
")"){
```

```
66
 while(stos.back() != "(")
{
67                  cout <<
stos.back() << " ";
68
 stos.pop_back();
69              }
70
 stos.pop_back();
71          }
72      }
73
74      return 1;
75  }
```

12

Czas na rozpatrzenie ostatniego przypadku: w zmiennej `elementWyrazenia` został zapisany jeden z operandów działania. Wypisujemy wartość zapisaną w `elementWyrazenia`. Oto cały program:

```
1  #include <iostream>
2  #include <list>
3
4  using namespace std;
5
6  int
   pobierzPriorytet(string
   op){
7      if(op == "+" || op ==
   "-"){
8          return 1;
9      }
10     else if(op == "*" ||
   op == "/"){
11         return 2;
12     }
13     else if(op == "^"){
14         return 3;
```

```
    }
    else{
        return 0;
    }
}

int main(){
    list<string> stos;

    string wyrazenie;
    getline(cin,
wyrazenie);

    list<string>
elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji =
wyrazenie.find(" ")) !=
string::npos){

 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
pozycjaSpacji));
        wyrazenie.erase(0,
pozycjaSpacji + 1);
    }

 elementyWyrazenia.push_ba
ck(wyrazenie.substr(0,
string::npos));


 while(!elementyWyrazenia.
empty()){
        string
elementWyrazenia =
elementyWyrazenia.front();

 elementyWyrazenia.pop_fro
nt();


 if(elementWyrazenia ==
```

```
"="){
42
43
 while(!stos.empty()){
44                 string op
= stos.back();
45                 cout << op
<< " ";
46
 stos.pop_back();
47             }
48         }
49         else
if(elementWyrazenia == "+"
|| elementWyrazenia == "-"
50                 ||
elementWyrazenia == "*" ||
elementWyrazenia == "/"
51                 ||
elementWyrazenia == "^"){
52
53
 while(!stos.empty()){
54
 if(pobierzPriorytet(eleme
ntWyrazenia) >
pobierzPriorytet(stos.back
())){
55                     break;
56                 }
57                 cout <<
stos.back() << " ";
58
 stos.pop_back();
59             }
60
 stos.push_back(elementWyr
azenia);
61         }
62         else
if(elementWyrazenia == "
("){
63
 stos.push_back("(");
64         }
```

```
65          else
if(elementWyrazenia ==
")"){
66
 while(stos.back() != "(")
{
67                  cout <<
stos.back() << " ";
68
 stos.pop_back();
69              }
70
 stos.pop_back();
71          }
72          else{
73              cout <<
elementWyrazenia << " ";
74          }
75      }
76
77      return 1;
78  }
```

Polecenie 2

Zmodyfikuj program przedstawiony w prezentacji tak, by przekształcał wyrażenie arytmetyczne zapisane w notacji postfiksowej na postać infiksową.

Uruchom swój program w środowisku lokalnym i przetestuj dla następującego wyrażenia arytmetycznego zapisanego w postaci postfiksowej:

```
8 9 * 3 6 + *
```

Zastosuj pełne nawiasowanie.

Poprawny wynik dla powyższych danych:

```
((8*9)*(3+6))
```

```
// Poniżej znajdziesz program z sekcji Przeczytaj, który należy

#include <iostream>
#include <list>

using namespace std;

int pobierzPriorytet(string op){
    if(op == "+" || op == "-"){
        return 1;
    }
    else if(op == "*" || op == "/"){
        return 2;
    }
    else if(op == "^"){
        return 3;
    }
    else{
        return 0;
    }
}

int main(){
```

```
    list<string> stos;

    string wyrazenie;
    getline(cin, wyrazenie);

    list<string> elementyWyrazenia;

    int pozycjaSpacji = 0;

    while((pozycjaSpacji = wyrazenie.find(" ")) != string::npos
        elementyWyrazenia.push_back(wyrazenie.substr(0, pozycja
        wyrazenie.erase(0, pozycjaSpacji + 1);
    }
    elementyWyrazenia.push_back(wyrazenie.substr(0, string::npo

    while(!elementyWyrazenia.empty()){
        string elementWyrazenia = elementyWyrazenia.front();
        elementyWyrazenia.pop_front();

        if(elementWyrazenia == "="){

            while(!stos.empty()){
                string op = stos.back();
                cout << op << " ";
                stos.pop_back();
            }
        }
        else if(elementWyrazenia == "+" || elementWyrazenia ==
                || elementWyrazenia == "*" || elementWyrazenia
                || elementWyrazenia == "^"){

            while(!stos.empty()){
                if(pobierzPriorytet(elementWyrazenia) > pobierz
                    break;
                }
                cout << stos.back() << " ";
                stos.pop_back();
            }
            stos.push_back(elementWyrazenia);
        }
        else if(elementWyrazenia == "("){
            stos.push_back("(");
```

```
        }
        else if(elementWyrazenia == ")"){
            while(stos.back() != "("){
                cout << stos.back() << " ";
                stos.pop_back();
            }
            stos.pop_back();
        }
        else{
            cout << elementWyrazenia << " ";
        }
    }

    return 1;
}
```

Twoje zadania

1. Program przekształca wyrażenie arytmetyczne zapisane w notacji postfiksowej na postać infiksową.

```
// Miejsce na wpisanie kodu
```

```
```

Słownik

notacja klasyczna (notacja infiksowa)

sposób zapisu dwuargumentowego wyrażenia arytmetycznego, w którym operator działania zostaje zapisany między operandami, np. `2 + 2`

notacja polska (notacja prefiksowa)

sposób zapisu wyrażenia arytmetycznego, w którym operator umieszczony jest przed operandami, np. `+ 2 2`

odwrotna notacja polska (ONP, notacja postfiksowa)

sposób zapisu wyrażenia arytmetycznego, w którym operator umieszczony jest za operandami, np. `2 2 +`

operand

liczba, na której wykonywane jest działanie

> **Przykład 1**
>
> Dla wyrażenia arytmetycznego:
>
> ```
> 27 * 5
> ```
>
> liczby 27 oraz 5 to **operandy**, a znak mnożenia jest **operatorem**

pełne nawiasowanie

szczególne ustawienie nawiasów w wyrażeniu arytmetycznym w notacji infiksowej; dzięki niemu każde działanie ma jednoznacznie określone dwa operandy za pomocą ujęcia w nawiasy, a także całe działanie jest objęte nawiasem, np. `(a+(b·c))`

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który będzie wypisywać znaki z łańcuchu znaków `ciag` tak długo, aż napotka znak, który jest cyfrą.

Przetestuj swój program dla następujących danych:

```
ciag = "asfkbahwdkcvkjxheewid1fdsf"
```

Specyfikacja problemu:

Dane:

- `ciag` – łańcuch znaków, zawierający co najmniej jedną cyfrę

Wynik:

Program na standardowym wyjściu wypisuje znaki z łańcucha znaków `ciag` do czasu napotkania na cyfrę.

Twoje zadania

1. Program wypisuje znaki z `ciag` dopóki nie napotka znaku, który jest cyfrą.

Ćwiczenie 2

Napisz program, który doda znaki operatorów arytmetycznych (dodawania `+`, odejmowania `-`, mnożenia `*` oraz dzielenia `/`) kolejno występujących w łańcuchu znaków `wyrazenie` do łańcucha znaków `napis`, a następnie wypisze `napis`.

Przetestuj swój program dla następujących danych:

- `napis = ""`
- `wyrazenie = "(3/4*9)-4+8/9"`

Specyfikacja problemu:

Dane:

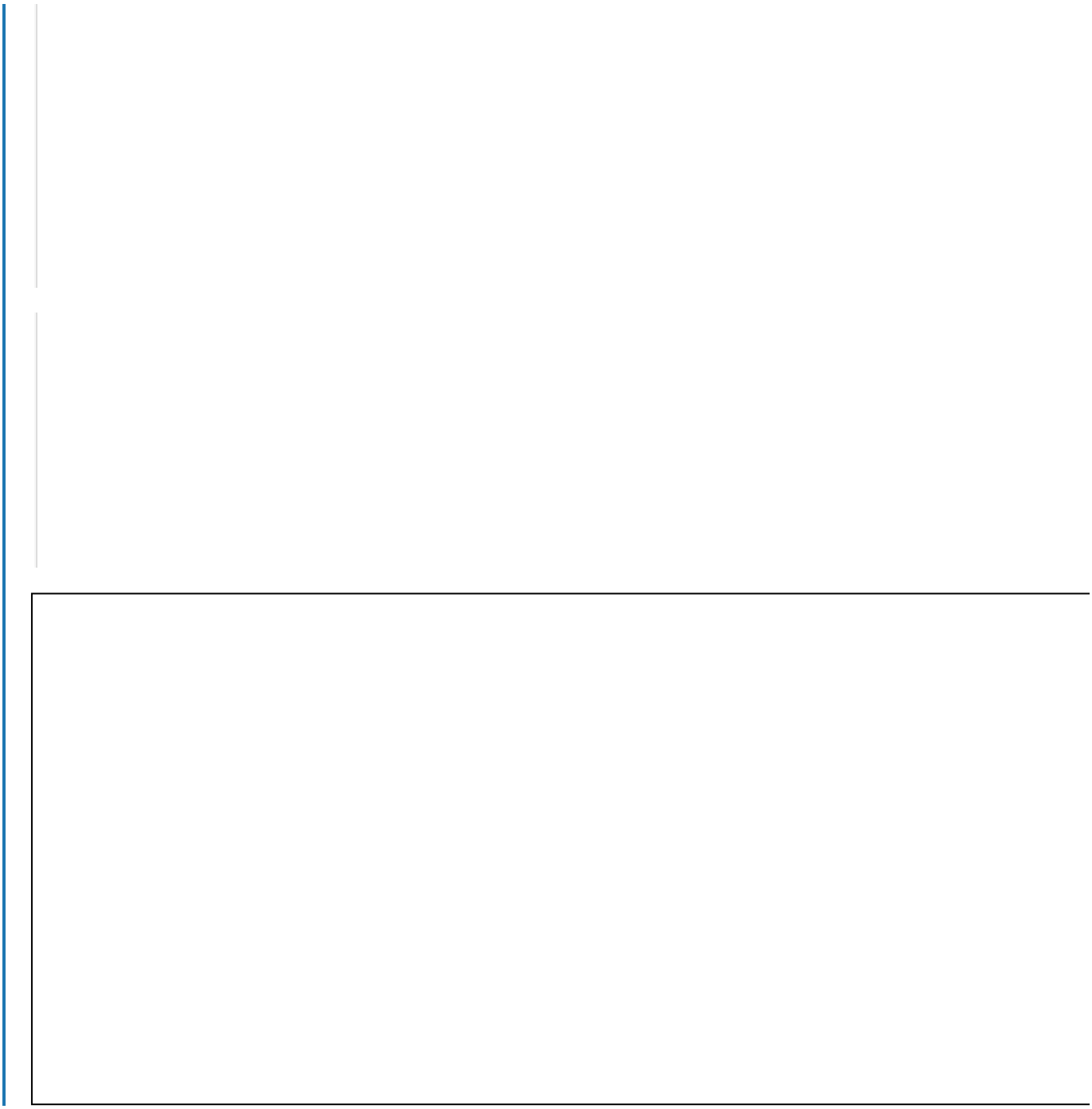
- `napis` – łańcuch znaków
- `wyrazenie` – łańcuch znaków

Wynik:

Program wypisuje łańcuch znaków powstały poprzez sklejenie łańcucha znaków `napis` i kolejnych operatorów arytmetycznych, występujących w łańcuchu znaków `wyrazenie`.

Twoje zadania

1. Program dodaje operatory arytmetyczne (dodawania `+`, odejmowania `-`, mnożenia `*` oraz dzielenia `/`) z łańcucha znaków `wyrażenie` do łańcucha znaków `napis`. Program wypisuje łańcuch znaków `napis`.



Ćwiczenie 3

Napisz program, który w sposób rekurencyjny zamienia zapis wyrażenia arytmetycznego w łańcuchu znaków `klasyczna` z notacji klasycznej na ONP i wypisuje je w jednym wierszu. Każde z wyrażeń arytmetycznych zapisanych w łańcuchu `klasyczna` powinno być otoczone parą nawiasów okrągłych, a między operandami i operatorami nie powinien występować znak spacji.

Przetestuj swój program dla następujących danych:

```
klasyczna = "((8*9)*(3+6))"
```

Specyfikacja problemu:

Dane:

- `klasyczna` – ciąg znaków, wyrażenie arytmetyczne zapisane w notacji infiksowej, gdzie dozwolonymi znakami są jednocyfrowe liczby, operatory: dodawania `+`, odejmowania `-`, mnożenia `*`, dzielenia `/` oraz nawiasy okrągłe: otwierające `(` oraz zamykające `)`.

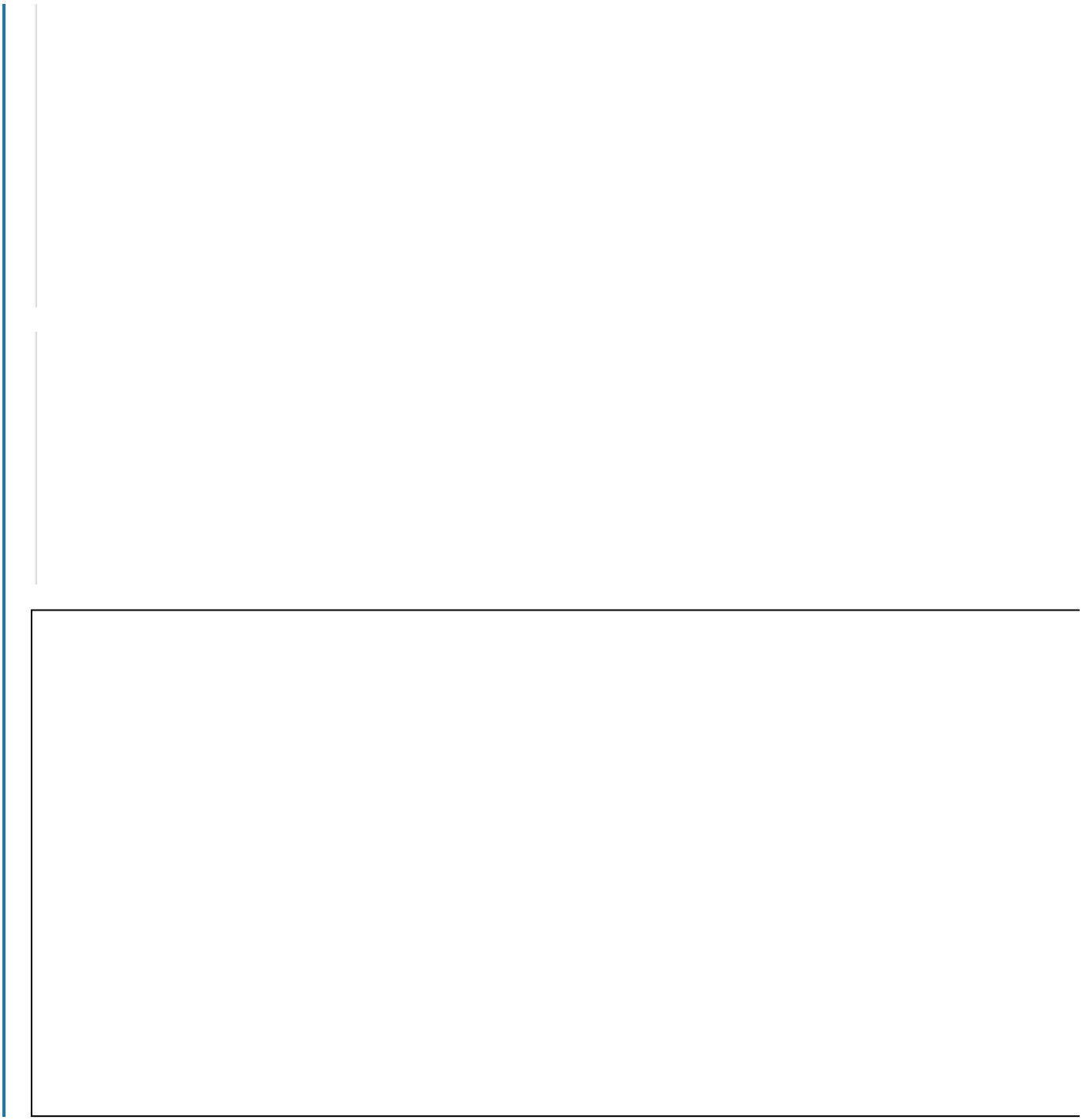
Wynik:

Na standardowym wyjściu program wypisuje wyrażenie arytmetyczne `klasyczna` zamienione na odwrotną notację polską.

Napisz program, który zamieni notację klasyczną na odwrotną notację polską.

Twoje zadania

1. Program zamienia zapis wyrażenia w łańcuchu znaków `klasyczna` z notacji klasycznej na odwrotną notację polską.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Odwrotna notacja polska w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) wykorzystuje znane sobie algorytmy przy rozwiązywaniu i programowaniu rozwiązań następujących problemów:

d) zamiany wyrażenia na postać w odwrotnej notacji polskiej i obliczanie jego wartości na podstawie tej postaci,

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

i) struktury dynamiczne: stos, kolejka, lista (do realizacji algorytmu: ONP, symulacji problemu Flawiusza, sortowania leksykograficznego),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;

- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Wykorzystasz w praktyce sposób zapisywania wyrażeń arytmetycznych z zastosowaniem odwrotnej notacji polskiej.
- Zaimplementujesz w języku C++ algorytm przekształcania wyrażenia arytmetycznego zapisanego z wykorzystaniem notacji infiksowej do postaci w ONP.
- Rozwiążesz przykładowe zadania wymagające wykazania się znajomością ONP.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Odwrotna notacja polska w języku C++”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Film samouczek”.

Faza wstępna:

1. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – co to jest odwrotna notacja polska?
   – czym się różni od notacji polskiej?
   – w jaki sposób zmieniamy notację klasyczną na ONP ?
   – jak nazywamy strukturę danych, w której element danych dodany jako ostatni jest zwracany jako pierwszy?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z multimedium.** Uczniowie zapoznają się z poleceniem 1 z sekcji „Film samouczek" i je wykonują. W parach wypracowują rozwiązanie, następnie wykonują polecenie 2.
2. Uczniowie w parach rozwiązują polecenie 3 z sekcji „Film samouczek". Następnie na forum klasy omawiają swoje odpowiedzi.
3. Uczniowie wykonują indywidualnie polecenia 4 oraz 5.
4. **Praca z tekstem.** Nauczyciel czyta polecenie 1 z sekcji „Przeczytaj". Uczniowie analizują prezentację multimedialną.
5. **Ćwiczenie umiejętności.** Nauczyciel przechodzi do sekcji „Sprawdź się". Uczniowie indywidualnie rozwiązują ćwiczenia nr 1 i 2. Osoba, która poprawnie rozwiąże zadanie jako pierwsza, wygrywa, a nauczyciel może nagrodzić ją oceną za aktywność.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 2 z sekcji „Przeczytaj".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Nauczyciel może wykorzystać multimedium w sekcji „Przeczytaj” do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Odwrotna notacja polska w języku C++”.