Zastosowanie funkcji w prostych programach w języku C++

Zastosowanie funkcji w prostych programach w języku C++

Źródło: Pixabay, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

W e-materiale Zastosowanie funkcji w prostych programach przedstawiliśmy algorytm rozwiązywania układu dwóch równań, z dwoma niewiadomymi, za pomocą metody wyznaczników. W tym e-materiale zaprezentujemy jego implementację w języku C++.

Implementację w pozostałych językach programowania znajdziesz w e-materiałach:

- Zastosowanie funkcji w prostych programach w języku Java,
- Zastosowanie funkcji w prostych programach w języku Python.

Więcej zadań? Przejdź do e-materiału Zastosowanie funkcji w prostych programach – zadania maturalne.

> **Twoje cele**
>
> - Zastosujesz zasady programowania strukturalnego w rozwiązywaniu problemów.
> - Zaimplementujesz funkcje w języku C++.
> - Napiszesz program, którego zadaniem będzie rozwiązanie układu równań z dwiema niewiadomymi, przy pomocy metody wyznaczników.

Prezentacja multimedialna

Problem 1

Napisz program, który rozwiąże układ równań z dwiema niewiadomymi przy pomocy metody wyznaczników.

Specyfikacja:

Dane:

- `a, b, c, d, e, f` – liczby całkowite stanowiące współczynniki układu równań; ich wartości wprowadzane są przez użytkownika z poziomu klawiatury

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

Wynik:

Program wyświetla parę liczb rzeczywistych `(x,y)` jako rozwiązanie układu równań lub komunikaty: `Układ sprzeczny` albo `Nieskończenie wiele rozwiązań`.

Przykład działania programu:

```
Podaj a:
7
Podaj b:
2
Podaj c:
1
Podaj d:
3
Podaj e:
4
Podaj f:
2
x: 0
y: 0.5
```

Wynikiem działania programu jest więc para liczb rzeczywistych: rozwiązanie układu równań dla współczynników wprowadzonych przez użytkownika.

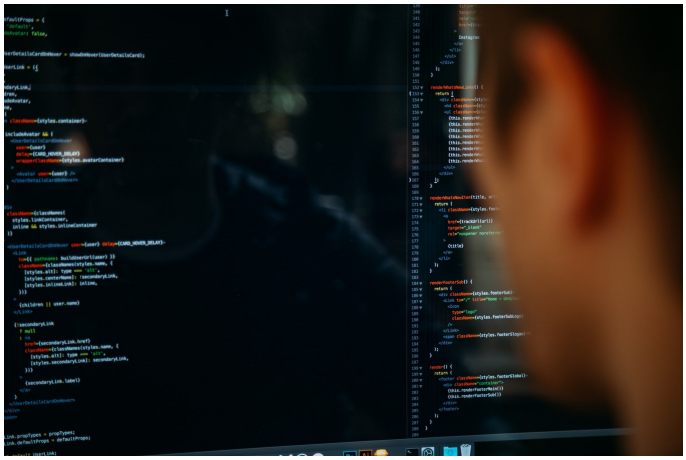
Polecenie 1

Porównaj swoje rozwiązanie z filmem.

Polecenie 2

Przeanalizuj prezentację dotyczącą przykładów zastosowania funkcji. Następnie zastanów się w jakich sytuacjach funkcje przynoszą nam korzyści, a kiedy ich wykorzystanie jest zbędne i nieefektywne.

1



Źródło: charlesdeluvio, unsplash, CC0.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Każda funkcja w języku C++ ma trzy własności:

- posiada typ zwracanej wartości (w przypadku, gdy funkcja nie zwraca żadnej wartości, w miejscu typ wartości wpisujemy `void`),
- posiada swoją nazwę,
- może posiadać dowolną liczbę parametrów.

W języku C++ żadna zmienna nie może zostać użyta przed zadeklarowaniem jej. To samo dotyczy funkcji. W przypadku funkcji mamy dwie możliwości jej deklaracji.

1) Zapisanie całej funkcji (nagłówek oraz ciało funkcji) nad funkcją `main`.

2) Zapisanie nagłówka funkcji (w którym zawrzemy typ oraz nazwę naszej funkcji, a następnie typy parametrów, dopisując nazwy (lub nie dopisując nazw) poszczególnych parametrów) nad funkcją `main()`, a całość zakończymy średnikiem. Następnie pod ciałem funkcji `main()` zapiszemy nagłówek funkcji włącznie z nazwami przyjmowanych parametrów, a potem pod nagłówkiem zapiszemy ciało funkcji. Po zapisaniu nagłówka funkcji pod funkcją `main()` nie zapisujemy średnika!

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Przykładowy program, w którym zostały zadeklarowane trzy funkcje, w każdy możliwy sposób.

Funkcja `funkcja` została zadeklarowana poprzez podanie typu i nazwy funkcji oraz typów parametrów nad funkcją `main`, a ciało funkcji – włącznie z nazwami parametrów – zostało zapisane pod funkcją `main`.

Funkcja `funkcja2` została zadeklarowana poprzez podanie typu i nazwy parametrów oraz samej funkcji nad funkcją `main`, a ciało funkcji – włącznie z nazwami parametrów – zostało zapisane pod funkcją `main`.

Funkcja `funkcja3` została zadeklarowana poprzez podanie typu i nazwy parametrów (ta funkcja nie przyjmuje żadnych parametrów, dlatego nawias po nazwie funkcji jest pusty) oraz ciała funkcji nad funkcją `main`.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Parametry każdej funkcji mogą mieć wartości domyślne. Jeśli funkcja zostanie wywołana bez przekazania do niej argumentów, zostaną użyte wartości domyślne.

WAŻNE!

Jeśli w deklaracji jednej funkcji część parametrów ma wartości domyślne, a część nie, to parametry z wartościami domyślnymi powinny znaleźć się na końcu deklaracji funkcji.

Przykład:

```
int FunkcjaTestowa(int a, int b,
int c = 4, int d = 3)
```

Przykład błędnej deklaracji:

```
int FunkcjaTestowa(int a, int b =
3, int c, int d = 12)
```

Przy wywołaniu funkcji i podaniu jej dwóch argumentów w pierwszym przypadku argumenty zostaną podstawione pod zmienne a oraz b, a zmiennym c oraz d zostaną przypisane wartości domyślne.

W przypadku błędnego zapisu zmienna c zostanie bez argumentu.

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Przykład programu z dwiema funkcjami. Jedną, która ma odpowiednio ustawione parametry

z wartościami domyślnymi oraz drugą, która ma je w błędnej kolejności.

5

Materiał audio dostępny pod adresem:

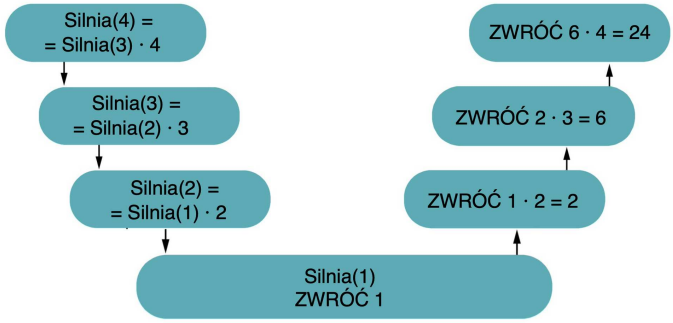
https://zpe.gov.pl/b/PgzqabXth

Funkcje mogą być wywoływane rekurencyjnie. Jeśli funkcja jest wywołana rekurencyjnie, to oznacza, że została wywołana przez samą siebie. W takim przypadku tworzy się kolejna instancja funkcji. Przykład funkcji zapisanej rekurencyjnie to:

Funkcja ta będzie wywoływała samą siebie, przekazując dwa argumenty. Pierwszy argument: `liczba` jest to liczba, której potęgę chcemy policzyć. Drugi argument - `potega` wskazuje na wartość wykładnika potęgi. Zauważmy, że gdy wykładnik będzie równy 0, przerywamy wywołanie rekurencyjne, zwracając wartość 1 (każda liczba podniesiona do potęgi 0 wynosi 1). Przykładowo dla `liczba` = 5 oraz `potega` = 3 funkcja ta będzie początkowo wywołana z argumentami 5 oraz 3, następnie po sprawdzeniu warunku zostanie ponownie wywołana z argumentami 5 oraz 2, ponieważ w momencie wywołania tej instancji funkcji parametr `liczba` przyjmuje wartość argumentu 5, a parametr `potega` wartość 3 - 1 = 2. Potem postępujemy analogicznie, do momentu, gdy wartość argumentu `potega` będzie równa zero. W tym wypadku funkcja zwraca wartość 1 oraz zostaje zakończona dana instancja funkcji `DoPotegi`. Zwrócona wartość wraca do instancji wywołującej, tam zostaje przemnożona przez 5, ponieważ to jest liczba, której potęgę chcemy obliczyć. Cały proces się powtarza do momentu, aż pierwsza

instancja funkcji `DoPotegi` nie zwróci wartości.

6



Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Przykładowy program obliczający silnię rekurencyjnie:

Kod działa następująco: po podaniu argumentu, dla którego chcemy obliczyć wartość silni, zostaje wywołana funkcja `Silnia`, w której sprawdzany jest warunek, czy podany argument jest równy 1 bądź 0 – w takim przypadku wartość silni wynosi jeden. W innym przypadku zwracana jest wartość funkcji `Silnia` z argumentem o jeden mniejszym, a następnie jest ona przemnażana przez wartość argumentu a w obecnej instancji. Proces obliczania wartości silni, gdy parametr a przyjmuje wartość argumentu 4, został przedstawiony na rysunku.

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Funkcja `main()` jest typu `int`, może więc zwracać wartości.

1) W przypadku, gdy zostanie zwrócona wartość `EXIT_SUCCESS` lub `0`, to środowisko, w którym funkcja została uruchomiona, przyjmie, iż program został zakończony bez błędów.

2) Gdy zwrócone zostanie `EXIT_FAILURE`, to środowisko przyjmie, iż program został zakończony porażką/błędem.

Wartości `EXIT_SUCCESS` oraz `EXIT_FAILURE` zostały zdefiniowane w bibliotece `cstdlib`.

Jeśli funkcja `main` zostanie zakończona normalnie – bez błędów – kompilator uzna, iż została zwrócona wartość zero, nawet jeśli programista w kodzie źródłowym nie zawarł instrukcji `return 0;`. Zwróć uwagę, że program główny też jest funkcją. Zwraca wartość typu `int`.

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Przykład wykorzystania `EXIT_FAILUREE`:

Przykład wykorzystania `EXIT_SUCCESS`:

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Wywoływanie funkcji zawsze wiąże się z dodatkową liczbą operacji. Czasami lepiej po prostu wstawić fragment kodu, który wykonuje

funkcja w miejsce jej wywołania. W tym przypadku wykorzystujemy sformułowanie `inline`. Zapisujemy je przed podaniem typu funkcji. Przy wykorzystaniu tego sformułowania kompilator wstawia kod funkcji w miejsce jej wywołania, dzięki czemu nie są wykonywane dodatkowe operacje lub przeskoki między funkcją `main` a funkcją wywoływaną.

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PgzqabXth

Przykładowy program wykorzystujący instrukcję `inline`:

Przeczytaj

Przekazywanie argumentów przez referencję

W języku C++ funkcje domyślnie kopiują pojedyncze zmienne na czas wywołania funkcji. W momencie zakończenia funkcji, kopie te są usuwane z pamięci. Aby móc operować na oryginalnych zmiennych, np. wczytywać do nich dane, musimy zawrzeć znak & (czyt. „ampersand") przed nazwą zmiennej, w nagłówku funkcji np.
`void WczytajDane(int& a, char& b)`

Nie jest to konieczne przy operowaniu na tablicach. Ponieważ gdy przekazujemy tablicę do funkcji, jednocześnie przekazujemy adres pierwszego elementu w tej tablicy. Tablice nie są kopiowane na potrzeby funkcji, ponieważ byłoby to zbyt czasochłonne, a sam proces zajmowałby dużo pamięci.

W naszym przypadku, aby wczytane wartości były zapisane do oryginalnych zmiennych przy wywołaniu funkcji, musimy zapisać znak & przed każdą nazwą zmiennej. Nagłówek funkcji powinien być zapisany w następujący sposób:
`void WczytajDane(int& a, int& b, int& c, int& d, int& e, int& f)`

Różne funkcje o tych samych nazwach

Warto wspomnieć, że w języku C++ kilka funkcji może mieć te same nazwy. Jednak wywoływana będzie ta funkcja, która ma odpowiednie typy argumentów.

```
#include <iostream>

using namespace std;

void WczytajDane(int& a)
{
    cout << "Podaj liczbe: ";
    cin >> a;
}

void WczytajDane(char& a)
{
    cout << "Podaj znak: ";
    cin >> a;
}
```

```
16
17 int main()
18 {
19     int a;
20     char b;
21
22     WczytajDane(a);
23     WczytajDane(b);
24
25     return 0;
26 }
```

W przedstawionym programie mamy zapisane dwie funkcje `WczytajDane()`. Jedna z nich jako parametr ma podaną referencję do zmiennej typu `int`, natomiast druga jako parametr ma podaną referencję do zmiennej typu `char`. W momencie wywołania funkcji `WczytajDane()` w naszej funkcji `main` kompilator wie, której funkcji musi użyć na podstawie podanych jej argumentów. W przypadku wywołania funkcji `WczytajDane()` dla zmiennej `a` zostanie wywołana ta funkcja, która jako parametr ma podaną referencję do zmiennej typu `int`. Natomiast w momencie wywołania funkcji `WczytajDane()` dla zmiennej `b` zostanie wywołana ta funkcja, której parametr jest typu `char`.

Występowanie co najmniej dwóch funkcji o takiej samej nazwie nazywamy przeciążaniem funkcji. W przypadku wywołania takich funkcji mogą pojawić się sytuacje niejednoznaczne lub błędy, ponieważ nie zawsze kompilator jest w stanie odpowiednio odróżnić funkcje o tej samej nazwie. W celu uniknięcia tych problemów warto rozróżniać funkcje, nadając im niejednakowe nazwy.

> **Przykład 1**
>
> Program obliczający, ile jest liczb pierwszych w przedziale $< 2; x >$

Nasz program będzie pobierał od użytkownika liczbę, a następnie sprawdzał czy kolejne liczby (począwszy od dwójki, a kończąc na liczbie x) są pierwsze. Wynikiem programu będzie informacja, ile z napotkanych liczb było pierwszymi. Zaproponowane rozwiązanie wyznaczania liczb pierwszych z zadanego zakresu jest rozwiązaniem alternatywnym do tego, które przedstawiliśmy w innym materiale dotyczącym algorytmu - Sito Eratostenesa. Nie wymaga użycia dodatkowych struktur danych – tablic, dzięki czemu zyskujemy więcej pamięci, kosztem wydajności. Po przeanalizowaniu implementacji, spróbuj samodzielnie zaimplementować algorytm Sito Eratostenesa w języku C++, z zastosowaniem funkcji.

Specyfikacja problemu:

Dane:

- `x` – koniec przedziału, w którym będą poszukiwane liczby pierwsze; liczba naturalna > 2

Wynik:

Program na standardowym wyjściu drukuje liczbę liczb pierwszych w przedziale $< 2; x >$

Krok 1.
Zapisz szkielet naszego programu.

```
#include <iostream>

using namespace std;

int main()
{

    return 0;
}
```

Krok 2.
Zapisz nagłówek funkcji wczytującej dane wraz z klamrami obejmującymi ciało funkcji.

```
#include <iostream>

using namespace std;

void WczytajDane(int& x)
{

}

int main()
{

    return 0;
}
```

> W nagłówku funkcji zapisujemy znak `&` przy każdej zmiennej. Ponieważ wczytujemy dane do tej zmiennej, potrzebujemy dostępu do oryginału. Funkcja `WczytajDane()` jest funkcją typu `void`, ponieważ pozwala wczytać dane, a nie zwraca żadnych wartości.

Krok 3. Wypełnij ciało funkcji `WczytajDane()`.

```
#include <iostream>

using namespace std;

void WczytajDane(int& x)
{
    cout << "Podaj x: ";
    cin >> x;
}

int main()
{

    return 0;
}
```

Krok 4. W funkcji `main()` zadeklaruj odpowiednią zmienną, a następnie wczytaj do niej dane za pomocą wcześniej utworzonej funkcji.

```
#include <iostream>

using namespace std;

void WczytajDane(int& x)
{
    cout << "Podaj x: ";
    cin >> x;
}

int main()
{
    int x;
    WczytajDane(x);

    return 0;
}
```

Krok 5. Zapisz funkcję sprawdzającą, czy dana liczba jest liczbą pierwszą.

```
bool czyPierwsza(int a)
{
    for(int i = 2; i <= sqrt(a); i++)
    {
        if(a % i == 0)
            return false;
    }
    return true;
}
```

Nasza funkcja jest typu `bool`, ponieważ zwraca `true`, jeśli liczba jest liczbą pierwszą oraz `false`, jeśli liczba nie jest liczbą pierwszą. Funkcja ma jeden parametr typu `int`, następnie w pętli, dla $i \in < 0, \sqrt{a} >$ sprawdzane jest, czy nasza liczba dzieli się bez reszty przez `i`. Jeśli tak – liczba nie jest pierwsza. Zatem zwracamy wartość `false`. Jeśli w trakcie działania pętli nie znajdziemy dzielnika `a` – oznacza to, że jest to liczba pierwsza. Możemy więc zwrócić `true`.

> **Ważne!**
>
> Funkcja `sqrt()` dostępna jest w bibliotece **cmath** – pamiętaj, by ją dołączyć w programie.

Krok 6. Zadeklarujmy zmienną, która będzie przechowywać liczbę wszystkich liczb pierwszych. Od razu wyzerujmy jej wartość.

```
#include <iostream>

using namespace std;

bool czyPierwsza(int a)
{
    for(int i = 2; i <= sqrt(a); i++)
    {
        if(a % i == 0)
            return false;
    }
    return true;
}

void WczytajDane(int& x)
{
    cout << "Podaj x: ";
```

```
    cin >> x;
}

int main()
{
    int x, pierwszych = 0;
    WczytajDane(x);

    return 0;
}
```

Krok 7. Zapiszmy pętlę `for`, która będzie się wykonywała od 2 do `x`, a w niej sprawdźmy – przy pomocy wcześniej zapisanej funkcji – (pamiętając o dołączeniu biblioteki `cmath`) czy liczba jest liczbą pierwszą. Jeśli tak jest, zwiększmy wartość zmiennej `pierwszych` o jeden.

```
#include <iostream>
#include <cmath>

using namespace std;

bool czyPierwsza(int a)
{
    for(int i = 2; i <= sqrt(a); i++)
    {
        if(a % i == 0)
            return false;
    }
    return true;
}

void WczytajDane(int& x)
{
    cout << "Podaj x: ";
    cin >> x;
}

int main()
{
    int x, pierwszych = 0;
    WczytajDane(x);

```

```
    for(int i = 2; i <= x; i++)
    {
        if(czyPierwsza(i))
            pierwszych++;
    }

    return 0;
}
```

Krok 8. Wypiszmy wartość zmiennej `pierwszych` wraz z odpowiednim komunikatem.

```
#include <iostream>
#include <cmath>

using namespace std;

bool czyPierwsza(int a)
{
    for(int i = 2; i <= sqrt(a); i++)
    {
        if(a % i == 0)
            return false;
    }
    return true;
}

void WczytajDane(int& x)
{
    cout << "Podaj x: ";
    cin >> x;
}

int main()
{
    int x, pierwszych = 0;

    WczytajDane(x);

    for(int i = 2; i <= x; i++)
    {
        if(czyPierwsza(i))
```

```
            pierwszych++;
    }
    cout << "Liczba liczb pierwszych w przedziale <2;"<< x <<"> :
    return 0;
}
```

Słownik

macierz

układ liczb, symboli lub wyrażeń zapisanych w postaci tablicy, w rzędach i kolumnach

metoda wyznaczników

metoda, która polega na rozwiązywaniu układu równań za pomocą wyznaczników; więcej na jej temat znajdziesz w e-materiale Metoda wyznacznikowa rozwiązywania układu równań liniowych z dwiema niewiadomymi

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który wyznaczy największy wspólny dzielnik dla n par liczb całkowitych. Przez parę liczb rozumiemy parę x_i i y_i, gdzie x_i jest i-tym elementem tablicy x, a y_i jest i-tym elementem tablicy y. Wypisz w jednej linii otrzymane wyniki dla każdej z par, oddzielając je spacją.

Działanie programu przetestuj dla następujących pięcioelementowych tablic:

```
x[5] = {75, 208, 295, 218, 33}
y[5] = {256, 270, 59, 80, 47}
```

Specyfikacja problemu:

Dane:

- n – liczba naturalna > 1
- x[n] – n-elementowa tablica liczb całkowitych
- y[n] – n-elementowa tablica liczb całkowitych

Wynik:

Wypisane w jednej linii i oddzielone spacją liczby całkowite, stanowiące największy wspólny dzielnik dla każdej z par.

Twoje zadania

1. Program oblicza, a następnie wyświetla największy wspólny dzielnik par liczb na odpowiadających sobie indeksach z tablic x i y.

```
#include <iostream>

using namespace std;

```

```
5  int main() {
6
7      int n = 5;
8      int x[n] = {75, 208, 295, 218, 33};
9      int y[n] = {256, 270, 59, 80, 47};
10
11     // W tym miejscu napisz implementację swojego rozwiązania
12 }
```

```
1
```

Ćwiczenie 2

Trójka pitagorejska to trzy liczby całkowite a, b, c, spełniające tzw. równanie Pitagorasa.

$$a^2 + b^2 = c^2$$

Napisz program, który zliczy, a następnie wyświetli liczbę trójek pitagorejskich w podanych n-elementowych tablicach. Przez trójkę rozumiemy trzy liczby a_i, b_i, c_i, gdzie a_i to i-ty element tablicy a, b_i to i-ty element tablicy b, a c_i jest i-tym elementem tablicy c.

Uwaga! Największa liczba z trójki nie zawsze jest elementem tablicy c. Trójkę zaliczamy więc do trójki pitagorejskiej, gdy jest spełnione jedno z trzech przedstawionych równań:

$$(a_i)^2 + (b_i)^2 = (c_i)^2$$

$$(a_i)^2 + (c_i)^2 = (b_i)^2$$

$$(b_i)^2 + (c_i)^2 = (a_i)^2$$

Przetestuj działanie programu dla trzech następujących dziesięcioelementowych tablic:

```
a[10] = {3, 55, 36, 30, 11, 25, 16, 7, 21, 63}
b[10] = {4, 38, 26, 29, 60, 17, 52, 24, 28, 280}
c[10] = {5, 19, 35, 53, 61, 39, 36, 25, 16, 287}
```

Specyfikacja problemu:

Dane:

- n – liczba naturalna > 1
- a [n] – n-elementowa tablica liczb całkowitych

- b[n] – n-elementowa tablica liczb całkowitych
- c[n] – n-elementowa tablica liczb całkowitych

Wynik:

Program wyświetla całkowitą liczbę trójek pitagorejskich dla podanych tablic.

Twoje zadania

1. Program sprawdza czy trójki z tablic a, b oraz c są pitagorejskie, a następnie jeśli są, zlicza ich liczbę i wyświetla na ekranie.

```
#include <iostream>

using namespace std;

int main()
{
    int n = 10;
    int a[] = {3, 55, 36, 30, 11, 25, 16, 7, 21, 63};
    int b[] = {4, 38, 26, 29, 60, 17, 52, 24, 28, 280};
    int c[] = {5, 19, 35, 53, 61, 39, 36, 25, 16, 287};
    int trojek = 0;

    for(int i = 0; i < n; i++)
    {
```

```

```

Ćwiczenie 3

Napisz program, który rozwiąże podany układ dwóch równań metodą wyznaczników. Układ podany jest w postaci tablicy łańcuchów znaków, z których każdy stanowi równanie w postaci $ax + by = c$ lub $ax - by = c$. Wypisz wyznaczoną wartość x oraz y, oddzielając je spacją. Jeśli układ nie ma rozwiązania, wypisz `Brak rozwiązania` lub `Nieskończenie wiele rozwiązań`, jeśli układ ma nieskończenie wiele rozwiązań.

Przetestuj działanie programu dla następującego układu równań:

$$\begin{cases} 2x - 3y = 20 \\ -1x + 2y = 12 \end{cases}$$

Specyfikacja problemu:

Dane:

- `rownania_str` – tablica ciągów znaków tworzących równania składające się na układ równań

Wynik:

Oddzielone spacją liczby rzeczywiste x i y, będące wynikami rozwiązania układu równań albo jeden z komunikatów: `Brak rozwiązania` lub `Nieskończenie wiele rozwiązań`.

Twoje zadania

1. Program rozwiązuje układ dwóch równań liniowych podanych w postaci dwuwymiarowej tablicy znaków.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    // równania w postaci ax+by=c lub ax-by+c
    string rownania_str[2] = {
```

```
        "2x-3y=20",
        "-1x+2y=12"
    };

    // W tym miejscu napisz implementację swojego rozwiązania
}
```

```
1
```

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Zastosowanie funkcji w prostych programach w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres podstawowy

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> 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

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.
>
> Zakres podstawowy. Uczeń:
>
> 1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

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):

- Zastosujesz zasady programowania strukturalnego w rozwiązywaniu problemów.
- Zaimplementujesz funkcje w języku C++.

- Napiszesz program, którego zadaniem będzie rozwiązanie układu równań z dwiema niewiadomymi, przy pomocy metody wyznaczników.

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: „Zastosowanie funkcji w prostych programach w języku C++". Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel inicjuje rozmowę wprowadzającą w temat lekcji. Przedstawia cele zajęć oraz kryteria sukcesu.

Faza realizacyjna:

1. Uczniowie metodą burzy mózgów przypominają sobie najważniejsze informacje dotyczące funkcji.

2. Uczniowie szukają rozwiązania dla Problemu 1 z sekcji „Prezentacja multimedialna". Porównują swój kod w parach. Chętna lub wybrana osoba przedstawia swoje rozwiązanie na forum.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1-3 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Na koniec zajęć z programowania w C++ nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak...".

Praca domowa:

1. Uczniowie wykonują polecenie 2 z sekcji „Prezentacja multimedialna".

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.
- 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:

- Uczniowie mogą zaproponować problemy matematyczne, które można rozwiązać za pomocą funkcji. W ramach pracy w grupach mogą pisać programy wykorzystujące funkcje, które rozwiązałyby problemy podane przez koleżanki i kolegów.