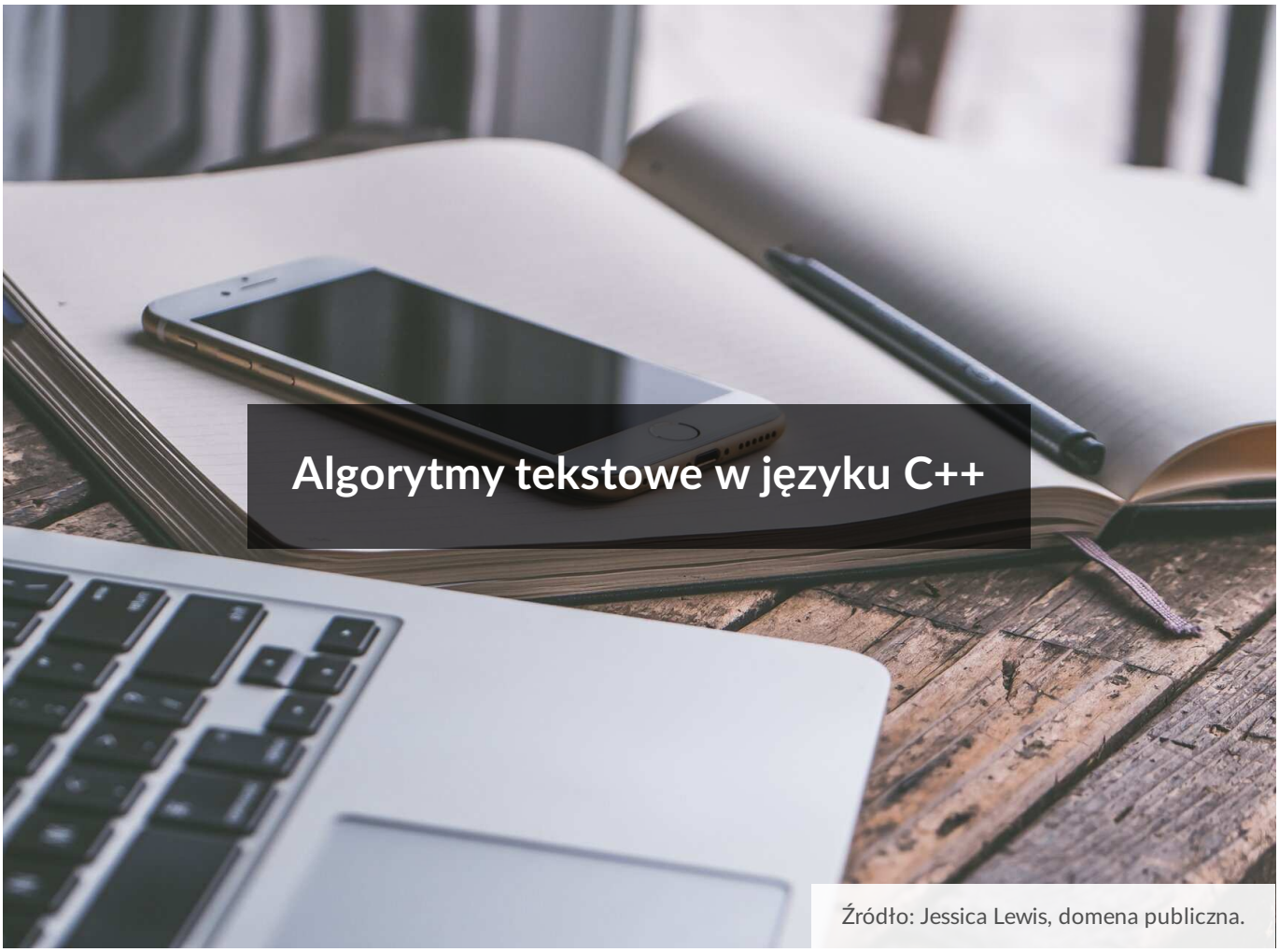
Algorytmy tekstowe w języku C++



Źródło: Jessica Lewis, domena publiczna.

Algorytmy tekstowe są przydatne do przetwarzania oraz przeszukiwania danych tekstowych. Co ciekawe, możemy je wykorzystać w popularnej grze w wisielca.
W e-materiale Algorytmy tekstowe omówiliśmy jej zasady oraz opracowaliśmy odpowiedni pseudokod.

W tym e-materiale zajmiemy się implementacją tej gry w języku C++.

Implementację algorytmów tekstowych w pozostałych językach programowania znajdziesz w e-materiałach:

- Algorytmy tekstowe w języku Java,
- Algorytmy tekstowe w języku Python.

Twoje cele

- Prześledzisz i utrwalisz zasady gry w wisielca.
- Przeanalizujesz, w jaki sposób dokonać implementacji gry w wisielca w języku C++.
- Zaimplementujesz algorytmy tekstowe, rozwiązując postawione problemy.

Film samouczek

Problem 1

Napisz w języku C++ program symulujący grę w wisielca.

Specyfikacja:

Założenia:

- nie rysuj szubienicy – liczbę szans przechowuj w zmiennej,
- komputer losuje słowo z wcześniej przygotowanej tablicy,
- gracz zgaduje słowo poprzez wprowadzanie kolejnych liter,
- gra kończy się, gdy gracz zgadnie słowo w całości lub gdy zabraknie mu szans.

Dane:

- `slowa` – słowa, z których komputer ma losować; tablica łańcuchów znaków

Wynik:

W kolejnych iteracjach pętli gry program – w przypadku podania przez użytkownika litery występującej w słowie – wyświetla niepełny napis uzupełniony o podaną literę lub odejmuje pozostałe szanse w przypadku litery niewystępującej w danym słowie.

```cpp
#include <iostream>

using namespace std;

int main () {

    // Tutaj dodaj kod. Żeby coś wypisać, użyj polecenia: cout
}
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z filmem.

Film dostępny pod adresem /preview/resource/RYnSw9yvCQi5I

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Film nawiązujący do treści materiału: Algorytmy tekstowe - gra w wisielca.

Kod programu zaprezentowanego w filmie:

Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.
Plik o rozmiarze 1.98 KB w języku polskim

Polecenie 2

Opracuj notatkę podsumowującą najważniejsze informacje przedstawione w filmie.

Przeczytaj

Implementacja w języku C++ – podsumowanie

Wiemy, jak przebiega partia gry w wisielca, spróbujmy więc napisać program, który ją odzwierciedli. Przedstawimy alternatywę dla realizacji kodu gry, zaprezentowanej w filmie samouczku.

Pierwszym krokiem jest zadeklarowanie dwóch zmiennych:

- `int zycia = 10;` - będzie przechowywać pozostałą liczbę prób (pozostałe życia), służącą do odgadnięcia litery lub całego słowa,
- `string nadpisane = "";` - będzie przechowywać słowo z aktualnie odgadniętymi literami.

> **Ważne!**
>
> Ponieważ zmienna `nadpisane` jest łańcuchem znaków `string`, należy pamiętać o dodaniu odpowiedniej biblioteki:

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

using namespace std;

int main(){
}
```

Kolejnym krokiem jest pobranie od pierwszego gracza słowa, które gracz drugi będzie próbował odgadnąć (w kodzie zaprezentowanym w filmie słowo było losowane z predefiniowanej tablicy haseł). Ciąg znaków pobrany z klawiatury zapisujemy w zmiennej `slowo`.

```
string slowo;
cin >> slowo;
```

Następnie wypełniamy zmienną `nadpisane` znakami. Liczba znaków podkreślenia odpowiada długości słowa wymyślonego przez gracza pierwszego. Zmienną wypisujemy na ekran, aby zobaczył ją gracz drugi i wiedział jaka jest długość słowa. Zadeklarowana zostaje także zmienna logiczna `czyOdgadniete`. Od jej wartości zależeć będzie, czy gracz drugi odgadł już słowo czy nie.

```
for (int i = 0; i < slowo.length(); i++) {
    nadpisane += "_";
}

cout << nadpisane << endl;

bool czyOdgadniete = false;
string odgadywanie;
```

Teraz należy zadeklarować pętlę, w której będzie się odbywała najważniejsza część algorytmu – pobieranie od gracza drugiego kolejnych liter (lub całego słowa), które następnie zostaną sprawdzone, czy są częścią wymyślonego słowa.

```
while (zycia > 0) {
    cout << "Podaj literę lub całe słowo do sprawdzenia" << endl;
    cin >> odgadywanie;
```

Ponieważ gracz może zgadywać pojedynczą literę lub całe słowo (wersja gry opisywana w filmie zakładała odgadywanie jedynie pojedynczych liter), to należy obsłużyć oba przypadki za pomocą instrukcji warunkowej `if ... else`.

```
if (odgadywanie.length() == 1) {

} else if(odgadywanie.length() > 1) {

}
```

Zajmijmy się teraz pierwszym przypadkiem, tzn. użytkownik odgaduje pojedynczą literę. Wewnątrz instrukcji warunkowej sprawdzane jest, czy podana przez gracza litera występuje w wymyślonym słowie. Jeżeli występuje, zmienna `czyZawiera` przyjmuje wartość `true` i następuje natychmiastowe wyjście z pętli, natomiast w przeciwnym przypadku przyjmuje wartość `false`.

```
bool czyZawiera = false;

if (odgadywanie.length() == 1) {
    for (int i = 0; i < slowo.length(); i++) {
        if (slowo[i] == odgadywanie[0]) {
```

```
            czyZawiera = true;
            break;
        }
    }

} else if(odgadywanie.length() > 1) {

}
```

Kolejnym krokiem jest sprawdzenie wartości zmiennej `czyZawiera` i zdefiniowanie odpowiednich operacji:

- Jeżeli zapisana w zmiennej wartość logiczna to `true`:

- nadpisywana jest zmienna `nadpisane` – znaki podkreślenia w odpowiednich miejscach zamieniane są na rzeczywiste odgadnięte litery.

- Jeżeli zapisana w zmiennej wartość logiczna to `false`:

- graczowi odjęte zostaje jedno życie i wyświetla się komunikat informujący o tym, że litera którą podał, nie występuje w słowie wymyślonym przez gracza pierwszego.

```
bool czyZawiera = false;

if (odgadywanie.length() == 1) {
    for (int i = 0; i < slowo.length(); i++) {
        if (slowo[i] == odgadywanie[0]) {
            czyZawiera = true;
            break;
        }
    }

    if (czyZawiera == true) {
        for (int i = 0; i < slowo.length(); i++) {
            if (slowo[i] == odgadywanie[0]) {
                nadpisane[i] = odgadywanie[0];
            }
        }
    } else {
        zycia--;
        cout << "Słowo nie zawiera litery " + odgadywanie << endl
    }
```

```
21
22 } else if(odgadywanie.length() > 1) {
23
24 }
```

Następnie sprawdzane jest, czy słowo zostało w pełni odgadnięte, czy też zawiera jeszcze znaki podkreślenia. Od zmiennej `czyOdgadniete` zależy to, czy wyjdziemy z pętli i zakończymy algorytm.

```
1  bool czyZawiera = false;
2
3  if (odgadywanie.length() == 1) {
4      for (int i = 0; i < slowo.length(); i++) {
5          if (slowo[i] == odgadywanie[0]) {
6              czyZawiera = true;
7              break;
8          }
9      }
10
11     if (czyZawiera == true) {
12         for (int i = 0; i < slowo.length(); i++) {
13             if (slowo[i] == odgadywanie[0]) {
14                 nadpisane[i] = odgadywanie[0];
15             }
16         }
17     } else {
18         zycia--;
19         cout << "Słowo nie zawiera litery " + odgadywanie << endl
20     }
21
22     czyOdgadniete = true;
23
24     for (int i = 0; i < nadpisane.length(); i++) {
25         if (nadpisane[i] == '_') {
26             czyOdgadniete = false;
27             break;
28         }
29     }
30
31     if (czyOdgadniete == true) {
32         break;
```

```
    }

} else if(odgadywanie.length() > 1) {

}
```

Teraz zdefiniujmy operacje, których wykonywanie nastąpi, gdy użytkownik nie wprowadzi pojedynczej litery tylko ciąg znaków.

Sprawdzane jest, czy ciąg znaków podany przez gracza jest taki sam jak ciąg znaków wymyślony przez gracza pierwszego, tj. czy użytkownik poprawnie odgadł całe słowo. W przypadku, gdy wyrazy te są identyczne, oznacza to, że gracz drugi poprawnie odgadł słowo i następuje wyjście z pętli. W przeciwnym wypadku dekrementujemy zmienną `zycia`, która określa liczbę pozostałych prób.

```
bool czyZawiera = false;

if (odgadywanie.length() == 1) {
    for (int i = 0; i < slowo.length(); i++) {
        if (slowo[i] == odgadywanie[0]) {
            czyZawiera = true;
            break;
        }
    }

    if (czyZawiera == true) {
        for (int i = 0; i < slowo.length(); i++) {
            if (slowo[i] == odgadywanie[0]) {
                nadpisane[i] = odgadywanie[0];
            }
        }
    } else {
        zycia--;
        cout << "Słowo nie zawiera litery " + odgadywanie << endl
    }

    czyOdgadniete = true;

    for (int i = 0; i < nadpisane.length(); i++) {
        if (nadpisane[i] == '_') {
            czyOdgadniete = false;
```

```
            break;
        }
    }

    if (czyOdgadniete == true) {
        break;
    }

} else if(odgadywanie.length() > 1) {

    if (odgadywanie == slowo) {
        czyOdgadniete = true;
        break;
    } else {
        zycia--;
        cout << "Podane słowo nie jest odgadywanym hasłem" << end
    }
}
```

Po każdej iteracji pętli wyświetlane są dwie informacje:

- zmienna `nadpisane`, czyli słowo wypełnione o odgadnięte litery,
- informacja o pozostałych życiach (próbach).

```
cout << nadpisane << endl;
cout << "Pozostałe życia: " + to_string(zycia) << endl;
```

Ostatnie, co należy zrobić (już poza główną pętlą), to wyświetlić komunikat o przebiegu gry:

```
if (czyOdgadniete == false) {
    cout << "Niestety nie udało ci się odgadnąć słowa" << endl;
} else {
    cout << "Udało ci się odgadnąć słowo!" << endl;
}
```

Pełny kod programu prezentuje się następująco:

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

```cpp

using namespace std;

int main(){

    int zycia = 10;
    string nadpisane = "";
    string slowo;
    cin >> slowo;

    for (int i = 0; i < slowo.length(); i++) {
        nadpisane += "_";
    }

    cout << nadpisane << endl;

    bool czyOdgadniete = false;
    string odgadywanie;

    while (zycia > 0) {
        cout << "Podaj literę lub całe słowo do sprawdzenia" << e
        cin >> odgadywanie;
        bool czyZawiera = false;

        if (odgadywanie.length() == 1) {
            for (int i = 0; i < slowo.length(); i++) {
                if (slowo[i] == odgadywanie[0]) {
                    czyZawiera = true;
                    break;
                }
            }

            if (czyZawiera == true) {
                for (int i = 0; i < slowo.length(); i++) {
                    if (slowo[i] == odgadywanie[0]) {
                        nadpisane[i] = odgadywanie[0];
                    }
                }
            } else {
                zycia--;
                cout << "Słowo nie zawiera litery " + odgadywanie
            }
```

```
45
46              czyOdgadniete = true;
47
48              for (int i = 0; i < nadpisane.length(); i++) {
49                  if (nadpisane[i] == '_') {
50                      czyOdgadniete = false;
51                      break;
52                  }
53              }
54
55              if (czyOdgadniete == true) {
56                  break;
57              }
58
59          } else if(odgadywanie.length() > 1) {
60
61              if (odgadywanie == slowo) {
62                  czyOdgadniete = true;
63                  break;
64              } else {
65                  zycia--;
66                  cout << "Podane słowo nie jest odgadywanym hasłem
67              }
68          }
69          cout << nadpisane << endl;
70          cout << "Pozostałe życia: " + to_string(zycia) << endl;
71      }
72      if (czyOdgadniete == false) {
73          cout << "Niestety nie udało ci się odgadnąć słowa" << end
74      } else {
75          cout << "Udało ci się odgadnąć słowo!" << endl;
76      }
77      return 0;
78  }
79
```

Słownik

> **dekrementacja**
>
> operacja zmniejszenia wartości zmiennej o jeden; szeroko wykorzystywana w programowaniu np. jako zmiana wartości iteratora w pętli

inkrementacja

operacja zwiększenia wartości zmiennej o jeden (przeciwieństwo dekrementacji); charakteryzuje się podobnym zastosowaniem jak dekrementacja

iteracja

technika programowania, opierająca się na powtarzaniu tej samej operacji określoną liczbę razy lub do momentu, w którym zadany warunek zostanie spełniony

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Rozpatrzmy grę w łańcuch słów: każdy z graczy na przemian mówi jedno słowo. Musi ono rozpoczynać się literą, którą kończyło się poprzednie słowo: jeśli pierwszy gracz powie „lokomotywa”, drugi może powiedzieć np. „arbuz”. Napisz program, który zasymuluje działanie drugiego gracza – dla podanego słowa znajdzie (w tablicy znanych słów) słowo zaczynające się jego ostatnią literą. Działanie programu przetestuj dla słów „wiadro” i „rebus”. Przyjmij jako znaną listę słów:

```
string znaneSlowa[] = {"arbuz", "lokomotywa", "wiatrak", "krowa", "mleko", "ananas", "osada", "krab"
```

Specyfikacja:

Dane:

- `znaneSlowa` – słowa, w których program będzie mógł szukać odpowiedzi; tablica łańcuchów znaków
- wprowadzone przez pierwszego gracza słowa, dla których program będzie mógł szukać odpowiedzi

Wynik:

Program na standardowym wyjściu wypisuje słowo zaczynające się ostatnią literą wprowadzonego słowa, jeśli znajdzie takie w tablicy `znaneSlowa`, lub „PRZEGRALEM”, jeśli nie znajdzie odpowiedniego słowa.

Twoje zadania

1. Program drukuje na standardowe wyjście wynik działania funkcji `lancuchSlow` dla podanych słów.

Ćwiczenie 2

Napisz program sprawdzający dla gry, w której uczestnicy układają słowa z liter tworzących inne słowo. Funkcja sprawdzająca ma przyjmować dwa słowa. Jeśli pierwsze z nich składa się tylko i wyłącznie z liter tworzących drugie słowo, funkcja powinna zwracać `true`, zaś w przeciwnym wypadku `false`. Działanie programu przetestuj dla par słów: „motyka” i „lokomotywa”; „loki” i „lokomotywa”; „wakat” i „lokomotywa”.

Specyfikacja:

Dane:

- `slowo` – słowo, którego litery będą sprawdzane; łańcuch znaków
- `slowoBazowe` – słowo, z którego liter powinno składać się `slowo`; łańcuch znaków

Wynik:

Program na standardowym wyjściu wyświetli 1, jeśli pierwsze słowo będzie składało się tylko z liter drugiego, lub w przeciwnym wypadku wyświetli 0.

Twoje zadania

1. Program drukuje na standardowe wyjście wynik funkcji `sprawdz` (`1` lub `0`) dla przykładowych słów.

Ćwiczenie 3

Przeprowadzana jest gra w „wisielca” – gracz próbuje odgadnąć słowo, którego litery są niewidoczne. Aby tego dokonać, może wybrać literę, która zostanie odsłonięta. Jeśli litera nie występuje w danym słowie, gracz traci jeden punkt życia. Dany jest algorytm, który gra w „wisielca” według następujących zasad:

1. Pierwsza sprawdzana litera to 'A' (pozycja w alfabecie 0).
2. Jeżeli sprawdzana litera, której pozycja w alfabecie to n, występuje w słowie k razy, następną sprawdzaną literą będzie litera na pozycji $n * 2 + k + 1$.
3. Jeśli dana litera nie występuje w słowie, sprawdzana jest litera na pozycji $n + 1$.
4. Jeżeli pozycja kolejnej litery przekracza zakres alfabetu ('Z' występuje na pozycji 25), wykonujemy dzielenie $modulo 26$ aktualnej pozycji.
5. W przypadku natrafienia na już sprawdzoną literę, nie sprawdzamy jej ponownie, a następną sprawdzaną literą będzie litera na pozycji $n + 1$.

Działanie algorytmu przetestuj dla podanej tablicy:

```
string slowa[5] = {
        "MAMA",
        "MATURA",
        "SZKOLA",
        "ALGORYTM",
        "INFORMATYKA"
    };
```

Załóż, że do dyspozycji jest 16 punktów życia.

Specyfikacja:

Dane:

- `slowa` – słowa do odgadnięcia przez program; tablica łańcuchów znaków

Wynik:

Program wypisuje na standardowym wyjściu liczbę odgadniętych słów.

Twoje zadania

1. Program przeprowadza symulację gry w „wisielca” oraz liczy, ile razy udało się wygrać określonym algorytmem.

Dla nauczyciela

Autor: zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Algorytmy tekstowe w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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. Rozumienie, analizowanie i rozwiązywanie problemów.

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

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

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

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

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):

- Prześledzisz i utrwalisz zasady gry w wisielca.
- Przeanalizujesz, w jaki sposób dokonać implementacji gry w wisielca w języku C++.
- Zaimplementujesz algorytmy tekstowe, rozwiązując postawione problemy.

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: „Algorytmy tekstowe w języku C++”. Uczniowie mają zapoznać się z treściami w sekcji

„Przeczytaj”.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie”. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. Nauczyciel prosi uczniów o przykłady wykorzystania algorytmów tekstowych.

Faza realizacyjna:

1. **Praca z tekstem.** Metodą burzy mózgów uczniowie referują najważniejsze informacje, jakie znaleźli w sekcji „Przeczytaj”, przygotowując się do lekcji. W razie potrzeby nauczyciel dopowiada i wyjaśnia niezrozumiałe kwestie.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek”. Uczniowie podemują próbę samodzielnego napisania programu do gry w wisielca. Hasłem do odgadnięcia niech będzie słowo „tajemnica”. Następnie porównują swoje rozwiązanie z zaprezentowanym w filmie.
3. **Ćwiczenie umiejętności.** Poszukiwanie najefektywniejszego rozwiązania problemu. Uczniowie wykonują w parach ćwiczenie nr 1 z sekcji „Sprawdź się”, a następnie porównują swój kod omawiając go wspólnie na forum. Nauczyciel ocenia efektywność zastosowanego rozwiązania.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się”, a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie”. W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. 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 ćwiczenie 3 z sekcji „Sprawdź się”.

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:

- Treści w sekcji „Film samouczek” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.