Konstruktory i destruktory w języku C++

Konstruktory i destruktory w języku C++

Źródło: Matthew Hamilton, domena publiczna.

Podczas pracy z aplikacjami zorientowanymi obiektowo powinniśmy zdawać sobie sprawę z roli specyficznych funkcji: konstruktora i destruktora obiektów.

Więcej o programowaniu obiektowym znajdziesz w e-materiale Programowanie obiektowe – projekt, etap I.

Ogólne informacje na temat konstruktorów i destruktorów zostały omówione w e-materiale Konstruktory i destruktory. Teraz zajmiemy się implementacją tych funkcji 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:

- Konstruktory i destruktory w języku Python,
- Konstruktory i destruktory w języku Java.

> **Twoje cele**
>
> - Przeanalizujesz informacje o programowaniu obiektowym.
> - Zaimplementujesz konstruktory i destruktory w języku C++.
> - Napiszesz własne programy obiektowe w języku C++.

Przeczytaj

Programowanie obiektowe – powtórzenie wiadomości

W paradygmacie programowania obiektowego wyróżniamy dwa ważne pojęcia: klasa i obiekt. Co one oznaczają?

Najłatwiej wytłumaczyć to na przykładzie. Załóżmy, że piszemy grę komputerową o walczących ze sobą zakonach rycerskich. Już na początku pojawia się zasadnicze pytanie: jak przedstawić postać rycerza w programie?

Nietrudno sobie wyobrazić, jak rycerz powinien wyglądać na ekranie. Znacznie trudniej odzwierciedlić to w kodzie programu. W programowaniu wykorzystujemy zmienne, tablice, funkcje, instrukcje warunkowe i wiele innych elementów. W jaki sposób z ich pomocą można stworzyć postać rycerza?

Niezbędne są **klasy**. Z technicznego punktu widzenia klasa to zbiór zmiennych oraz funkcji opisujących postać, jaką chcemy przenieść ze świata realnego do wirtualnego. Zmienne klasy nazywa się **polami klasy**, a funkcje – **metodami klasy**. Pola klasy służą do opisania cech danej postaci, zaś metody do opisania czynności, jakie może ona wykonać. Wróćmy zatem do omawianego przykładu.

Podstawowymi cechami rycerza w grze, opisywanymi przez pola klasy lub wartości atrybutów klasy, mogą być:

- imię,
- płeć,
- siła,
- szybkość,
- wiek,
- zdrowie.

Przykładowe czynności rycerza opisywane przez metody klasy to:

- atak,
- chodzenie,
- bieg,
- skok,
- zażycie eliksiru.

Stwórzmy przykładową klasę rycerza w języku C++:

```cpp
#include <iostream>

using namespace std;

class Rycerz /* nazwy klas zaczynamy wielką literą */ {
    public: // znaczenia tej linii nie musisz jeszcze znać, zosta

    /* Pola klasy */

    string imie; // imie ustawiamy jako ciąg znaków
    string plec;
    int wiek; // wiek ustawiamy jako liczbę całkowitą
    int sila;
    int zdrowie;

    /* Metody */

    void atak() {
        cout << "Rycerz wykonał atak" << endl;
    }

    void skok() {
        cout << "Rycerz wykonał skok" << endl;
    }

    void zazycieEliksiru() {
        cout << "Rycerz zażył eliksir" << endl;
    }
};
```

> **Ważne!**
>
> W języku C++ definicję klasy kończymy znakiem `;`.

Klasa to tylko schemat opisujący, z czego ma składać się dana postać. Żadna z wymienionych cech nie ma przypisanej wartości, ponieważ żaden rycerz nie został jeszcze utworzony. Przypomnijmy, czym są obiekty.

Obiekty to konkretne reprezentacje danej klasy. Używając jednego schematu rycerza (klasy), możemy stworzyć dowolną liczbę rycerzy (obiektów).

Stwórzmy więc kilka obiektów klasy `Rycerz`:

```cpp
#include <iostream>

using namespace std;

class Rycerz /* nazwy klas zaczynamy wielką literą */ {
    public: // znaczenia tej linii nie musismy jeszcze znać, zost

    /* Pola klasy */

    string imie; // imie ustawiamy jako napis
    string plec;
    int wiek; // wiek ustawiamy jako liczbę całkowitą
    int sila;
    int zdrowie;

    /* Metody */

    void atak() {
        cout << "Rycerz wykonał atak" << endl;
    }

    void skok() {
        cout << "Rycerz wykonał skok" << endl;
    }

    void zazycieEliksiru() {
        cout << "Rycerz zażył eliksir" << endl;
    }
};

int main() {
    Rycerz rycerz1; // pierwszy obiekt klasy 'Rycerz' o nazwie 'r
    Rycerz rycerz2; // drugi obiekt klasy 'Rycerz' o nazwie 'ryce
    Rycerz rycerz3; // trzeci obiekt klasy 'Rycerz' o nazwie 'ryc

    return 0;
}
```

Kilka obiektów już istnieje, ale żaden nie ma ustawionych wartości swoich cech. Każdemu polu publicznemu danego obiektu przypisujemy wartość z użyciem konstrukcji `nazwaObiektu.nazwaPola`. Przypiszmy tym obiektom poszczególne cechy.

```cpp
#include <iostream>

using namespace std;

class Rycerz /* nazwy klas zaczynamy wielką literą */ {
    public: // znaczenia tej linii nie musimy jeszcze znać, zosta

    /* Pola klasy */

    string imie; // imie ustawiamy jako napis
    string plec;
    int wiek; // wiek ustawiamy jako liczbę całkowitą
    int sila;
    int zdrowie;

    /* Metody */

    void atak() {
        cout << "Rycerz wykonał atak" << endl;
    }

    void skok() {
        cout << "Rycerz wykonał skok" << endl;
    }

    void zazycieEliksiru() {
        cout << "Rycerz zażył eliksir" << endl;
    }
};

int main() {
    Rycerz rycerz1; // pierwszy obiekt klasy 'Rycerz' o nazwie 'r
    Rycerz rycerz2; // drugi obiekt klasy 'Rycerz' o nazwie 'ryce
    Rycerz rycerz3; // trzeci obiekt klasy 'Rycerz' o nazwie 'ryc

    rycerz1.imie = "Artur"; // przypisujemy wartość do pola 'imie
    rycerz1.plec = "mężczyzna"; // przypisujemy wartość do pola '
    rycerz1.wiek = 35; // przypisujemy wartość do pola 'wiek' obi
    rycerz1.sila = 150; // przypisujemy wartość do pola 'sila' ob
    rycerz1.zdrowie = 657; // przypisujemy wartość do pola 'zdrow

```

```
42      return 0;
43  }
```

Proces przypisywania wartości polom klasy można uprościć, korzystając z **konstruktorów**.

Rola konstruktora i destruktora

Konstruktor

Konstruktor to specjalna metoda pozwalająca na stworzenie obiektu danej klasy oraz ewentualne przypisanie do tego obiektu wartości pól danej klasy. Każda utworzona klasa posiada swój tak zwany konstruktor domyślny, który nie zawsze jest widoczny w definicji klasy. Nie przypisuje on żadnych wartości i służy wyłącznie do tworzenia obiektów danej klasy. W przedstawionym wcześniej kodzie, w linijkach 36, 37 oraz 38, został wykorzystany konstruktor domyślny.

Zasady budowy konstruktora:

- konstruktor nie może mieć określonego typu zwracanego,
- nazwa konstruktora jest taka sama jak nazwa klasy,
- w nawiasach umieszczamy argumenty, których wartości zostaną przypisane do pól klasy,
- klasa może zawierać kilka konstruktorów (muszą jednak różnić się liczbą argumentów lub typem argumentów),
- utworzenie własnego konstruktora wymusza na nas jego stosowanie (konstruktor domyślny zostaje „zastąpiony i niedostępny").

Stwórzmy konstruktor dla klasy `Rycerz` i użyjmy go do nadania wartości pól:

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Rycerz /* nazwy klas zaczynamy wielką literą */ {
6      public: // znaczenia tej linii nie musimy jeszcze znać, zosta
7
8      /* Pola klasy */
9
10     string imie; // imie ustawiamy jako napis
11     string plec;
12     int wiek; // wiek ustawiamy jako liczbę całkowitą
```

```
13      int sila;
14      int zdrowie;
15
16      /* Konstruktor */
17
18      Rycerz (string imie, string plec, int wiek, int sila, int zdr
19          this.imie = imie;
20          this.plec = plec;
21          this.wiek = wiek;
22          this.sila = sila;
23          this.zdrowie = zdrowie;
24      }
25
26      /* Metody */
27
28      void atak() {
29          cout << "Rycerz wykonał atak" << endl;
30      }
31
32      void skok() {
33          cout << "Rycerz wykonał skok" << endl;
34      }
35
36      void zazycieEliksiru() {
37  cout << "Rycerz zażył eliksir" << endl;
38      }
39  };
40
41  int main() {
42      Rycerz rycerz1 ("Artur", "mężczyzna", 35, 150, 657); // pierw
43      Rycerz rycerz2 ("Geralt", "mężczyzna", 45, 450, 906); // drug
44      Rycerz rycerz3; // utworzenie obiektu w ten sposób (z życiem
45
46      return 0;
47  }
```

Jak widać, jest to sposób o wiele prostszy niż standardowe przypisywanie wartości.
W konstruktorze dodatkowo użyliśmy operatora `this`, który pozwala odwoływać się bezpośrednio do pól danej klasy. Dzięki temu możemy wygodnie użyć takich samych nazw argumentów konstruktora.

> **Ważne!**

> W klasie może istnieć więcej niż jeden konstruktor (czyli możemy je przeciążać). Musi jednak spełniać przynajmniej jedno z dwóch wymagań: mieć inną liczbę argumentów albo inne typy argumentów od pozostałych konstruktorów. Dzięki temu program w sposób jednoznaczny wie, którego konstruktora należy użyć.

Destruktor

Destruktor to specjalna metoda wywoływana tuż przed usunięciem obiektu. Służy do wykonania wszystkich czynności, które składają się na jego „zniszczenie", tak aby nie nastąpiły później w programie żadne nieprzewidziane błędy. Istnieje destruktor domyślny.

Zasady budowy destruktora:

- destruktor nie może mieć określonego typu zwracanego,
- nazwa destruktora składa się z nazwy klasy poprzedzonej znakiem ~,
- destruktor nie może posiadać argumentów (w związku z czym klasa może mieć tylko jeden destruktor).

Stwórzmy przykładowy destruktor dla klasy `Rycerz`:

```
#include <iostream>

using namespace std;

class Rycerz /* nazwy klas zaczynamy wielką literą */ {
    public: // znaczenia tej linii nie musisz jeszcze znać, zosta

    /* Pola klasy */

    string imie; // imie ustawiamy jako napis
    string plec;
    int wiek; // wiek ustawiamy jako liczbę całkowitą
    int sila;
    int zdrowie;

    /* Konstruktor */

    Rycerz (string konImie, string konPlec, int konWiek, int konS
        imie = konImie;
        plec = konPlec;
        wiek = konWiek;
        sila = konSila;
```

```
23          zdrowie = konZdrowie;
24      }
25
26      /* Metody */
27
28      void atak() {
29          cout << "Rycerz wykonał atak" << endl;
30      }
31
32      void skok() {
33          cout << "Rycerz wykonał skok" << endl;
34      }
35
36      void zazycieEliksiru() {
37          cout << "Rycerz zażył eliksir" << endl;
38      }
39
40      /* Destruktor */
41
42      ~Rycerz() {
43          std:cout << "Nastąpiło zniszczenie obiektu" << endl;
44      }
45  };
46
47  int main() {
48      Rycerz rycerz1 ("Artur", "mężczyzna", 35, 150, 657); // pierw
49      Rycerz rycerz2 ("Geralt", "mężczyzna", 45, 450, 906); // drug
50      Rycerz rycerz3; // utworzenie obiektu w ten sposób (z użyciem
51
52      return 0;
53  }
```

W tym przypadku destruktor wykona się (nastąpi zniszczenie obiektu) automatycznie w momencie zakończenia pracy programu i to właśnie wtedy zostanie wypisany komunikat: „Nastąpiło zniszczenie obiektu”. Zastanów się, w jakich innych przypadkach mógłby wykonać się dekonstruktor.

Słownik

> **destruktor**

przeciwieństwo konstruktora; specjalna metoda, wywoływana przez program przed usunięciem obiektu
klasa

typ danych; szablon dla stworzenia obiektu
konstruktor

specjalna funkcja (metoda) danej klasy, wywoływana, gdy tworzona jest instancja danej klasy
paradygmat programowania

wzorzec pisania oprogramowania, który definiuje, w jaki sposób programista postrzega sterowanie i wykonywanie programu komputerowego; przykłady paradygmatów to: programowanie obiektowe, programowanie funkcyjne, programowanie strukturalne oraz wiele innych

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z filmem. Wynotuj najważniejsze informacje.

Film dostępny pod adresem /preview/resource/RlvHt5z4IYdXC

Film przedstawia elementy programowania obiektowego w języku C++.

Plik o rozmiarze 767.00 B w języku polskim

Ważne!

Występujące w programie słowo *public* odnosi się do dostępności poszczególnych składowych klasy.

Polecenie 2

Zapoznaj się z prezentacją, w której pokazano poszczególne etapy tworzenia przykładowej klasy oraz jej obiektów. Powtórz zaprezentowane kroki i stwórz inną przykładową klasę.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Na początek stwórzmy ciało klasy o nazwie `Samochod`.

```
#include <iostream>

using namespace std;

class Samochod {

};
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Zdefiniujmy pola dla tej klasy: marka i model.

```
#include <iostream>

using namespace std;

class Samochod {
    public:

    /* Pola klasy */

    string marka;
    string model;
};
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Stwórzmy teraz konstruktor, dzięki któremu będziemy mogli wygodnie tworzyć nowe obiekty tej klasy.

```
#include <iostream>

using namespace std;

class Samochod {
    public:

    /* Pola klasy*/

    string marka;
    string model;

    /* Konstruktor */

    Samochod(string samochodMarka, string samochodModel) {
        marka = samochodMarka;
        model = samochodModel;
    }
};
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Następnie dodajmy funkcję główną `main()`, w której stworzymy nowy obiekt klasy `Samochod` o nazwie `tesla`.

```
#include <iostream>

using namespace std;

class Samochod {
```

```
    public:

    /* Pola klasy */

    string marka;
    string model;

    /* Konstruktor */

    Samochod(string samochodMarka, string samochodModel) {
        marka = samochodMarka;
        model = samochodModel;
    }
};

int main() {
    Samochod tesla("Tesla", "Model X");

    return 0;
}
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

5

Na koniec wypiszmy na ekranie wartości pól tej klasy.

```
#include <iostream>

using namespace std;

class Samochod {
    public:

    /* Pola klasy*/
```

```
 9
10      string marka;
11      string model;
12
13      /* Konstruktor */
14
15      Samochod(string
    samochodMarka, string
    samochodModel) {
16              marka =
    samochodMarka;
17              model =
    samochodModel;
18      }
19  };
20
21  int main() {
22      Samochod
    tesla("Tesla", "Model X");
23
24      cout << tesla.marka +
    " " + tesla.model;
25
26      return 0;
27  }
```

Na ekranie powinien pojawić się napis: Tesla Model X.

Polecenie 3

Przeanalizuj prezentację przedstawiającą krok po kroku tworzenie klasy pojedynczego elementu na liście, będącego częścią aplikacji typu TO-DO, czyli listy rzeczy do zrobienia. Zastanów się, o jakie inne funkcjonalności bazujące na klasach można ją wzbogacić.

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Na początek stwórzmy ciało klasy o nazwie `Element`.

```
#include <iostream>

using namespace std;

class Element {

};
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Zdefiniujmy teraz pola typu `string` dla tej klasy: `nazwa`, `kolor` oraz typu `int` – pole `liczbaDni`, które będzie przechowywać liczbę dni pozostałych do terminu wykonania.

```
#include <iostream>

using namespace std;

class Element {
    public:

    /* Pola klasy */

    string nazwa;
    string kolor;
    int liczbaDni;
};
```

Materiał audio dostępny pod adresem:

3

https://zpe.gov.pl/b/PHF1hggdK

Następnie stwórzmy konstruktor, dzięki któremu będziemy mogli wygodnie tworzyć nowe obiekty tej klasy, oraz destruktor, który przy usuwaniu elementu wyświetli komunikat: Zrobione.

```
#include <iostream>

using namespace std;

class Element {
    public:

    /* Pola klasy */

    string nazwa;
    string kolor;
    int liczbaDni;

    /* Konstruktor */

    Element(string nazwaElementu, string kolorElementu, int liczbaDniNaElement) {
        nazwa = nazwaElementu;
        kolor = kolorElementu;
        liczbaDni = liczbaDniNaElement;
    }

    /* Destruktor */

    ~Element() {
        cout << "Zrobione";
    }
};
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Dodajmy teraz funkcję główną `main()`, w której stworzymy nowy obiekt klasy `Element` o nazwie `element`.

```
#include <iostream>

using namespace std;

class Element {
    public:

    /* Pola klasy*/

    string nazwa;
    string kolor;
    int liczbaDni;

    /* Konstruktor */

    Element(string nazwaElementu, string kolorElementu, int liczbaDniNaElement) {
        nazwa = nazwaElementu;
        kolor = kolorElementu;
        liczbaDni = liczbaDniNaElement;
    }

    /* Destruktor */

    ~Element() {
        cout << "Zrobione";
    }
};

```

```
29 int main() {
30     Element
   element("Projekt",
   "czerwony", 13);
31
32     return 0;
33 }
```

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

5

Wypiszmy na ekranie nazwę tego elementu.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Element {
6      public:
7
8      /* Pola klasy*/
9
10     string nazwa;
11     string kolor;
12     int liczbaDni;
13
14     /* Konstruktor */
15
16     Element(string
   nazwaElementu, string
   kolorElementu, int
   liczbaDniNaElement) {
17         nazwa =
   nazwaElementu;
18         kolor =
   kolorElementu;
19         liczbaDni =
   liczbaDniNaElement;
20     }
21
```

```
    /* Destruktor */

    ~Element() {
        cout <<
"Zrobione";
    }
};

int main() {
    Element
element("Projekt",
"czerwony", 13);

    cout << element.nazwa
+ " ";

    return 0;
}
```

Po wykonaniu programu na ekranie powinien pojawić się rezultat: Projekt Zrobione.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PHF1hggdK

Na koniec stworzymy jeszcze drugi konstruktor, który umożliwi np. stworzenie obiektu klasy `Element`, ale bez dołączonej liczby dni na wykonanie.

```
#include <iostream>

using namespace std;

class Element {
    public:

    /* Pola klasy*/

    string nazwa;
    string kolor;
```

```
12      int liczbaDni;
13
14      /* Konstruktor */
15
16      Element(string
    nazwaElementu, string
    kolorElementu, int
    liczbaDniNaElement) {
17          nazwa =
    nazwaElementu;
18          kolor =
    kolorElementu;
19          liczbaDni =
    liczbaDniNaElement;
20      }
21
22      Element(string
    nazwaElementu, string
    kolorElementu) /* nowy
    konstruktor */ {
23          nazwa =
    nazwaElementu;
24          kolor =
    kolorElementu;
25      }
26
27      /* Destruktor */
28
29      ~Element() {
30          cout <<
    "Zrobione";
31      }
32  };
33
34  int main() {
35      Element
    element("Projekt",
    "czerwony", 13);
36
37      cout << element.nazwa
    + " ";
38
39      return 0;
40  }
```

Po wykonaniu programu na ekranie powinien pojawić się rezultat: Projekt Zrobione.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Uzupełnij klasę `Pies`, dodając do niej pole `imie` oraz konstruktor. Następnie stwórz obiekt tej klasy o nazwie `pies` i wyświetl na ekranie zawartość jego pola `imie`.

Swój program przetestuj dla pola `imie` o wartości `Snoopy`.

Twoje zadania

1. Napisz program wypisujący zawartość pola `imię` obiektu klasy `Pies` o nazwie `pies`.

Ćwiczenie 2

Stwórz klasę `Telefon` i dodaj do niej dwa pola: `marka` oraz `model`. Oprócz tego stwórz konstruktor inicjujący wartości pól dla tej klasy oraz dopisz i użyj metody `wypiszInfo()`, która wyświetli zawartość pól tej klasy.

Swój program przetestuj dla marki `Fonotel` i modelu `Dzwonix`.

Twoje zadania

1. Program wypisuje markę oraz model telefonu oddzielone myślnikiem.

Ćwiczenie 3

Zmodyfikuj klasę `Telefon`, dodając do niej nowe pole `czyUzywany` i kolejny konstruktor, który będzie inicjalizować trzy pola. Stwórz nowy obiekt tej klasy z użyciem nowego konstruktora i wypisz jego dane za pomocą metody `wypiszInfo ()`. Nie zapomnij o zmodyfikowaniu tej metody.

Swój program przetestuj dla marki: `Dzwonotel`; modelu: `Brzęczak`; wartości pola `czyUzywany`: `tak`.

Twoje zadania

1. Program tworzy obiekt telefon i wyświetla jego dane w następującej formie: `marka – model – czyUzywany`.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Konstruktory i destruktory w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

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 rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:
> >
> > > 1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;
> > >
> > > 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):

- Przeanalizujesz informacje o programowaniu obiektowym.
- Zaimplementujesz konstruktory i destruktory w języku C++.
- Napiszesz własne programy obiektowe w języku C++.

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: „Konstruktory i destruktory w języku C++”. Uczniowie zapoznają się z multimedium w sekcji „Prezentacja multimedialna”.

Faza wstępna:

1. Nauczyciel wprowadza uczniów szczegółowo w temat lekcji i jej cele. Może posłużyć się wyświetloną na tablicy zawartością sekcji „Wprowadzenie”.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Prezentacja multimedialna".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie zapoznają się z poleceniem 2 i analizują prezentację, w której pokazany jest krok po kroku proces utworzenia przykładowej klasy samochodu i kilku jej obiektów. Równolegle tworzą swój własny kod w C++. Następnie uczniowie wykonują polecenie nr 3. Zapoznają się z prezentacją, w której pokazany jest proces tworzenia klasy pojedynczego elementu w aplikacji typu TO-DO, czyli w liście rzeczy do zrobienia.
3. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1-3 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.

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).
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie rozwiązują problem 1 z sekcji „Prezentacja multimedialna" i porównują swoje rozwiązania z przedstawionym w filmie.

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:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj", „Prezentacja multimedialna", „Sprawdź się" jako materiał do lekcji powtórkowej.