Klasy

Wykład 4

Klasy

- Definiujemy własne typy danych (klasy)
  - Dodajemy składowe takie jak `int`, `float` lub inne klasy
  - Opisujemy zachowanie tworząc tzw. metody, czyli funkcje składowe
- Definicja klasy
  - ```
    class nazwa {
    ... //ciało klasy
    }; //ważny jest średnik na końcu
    ```
- Tworzenie obiektów zdefiniowanych przez klasę odbywa się w znany już sposób
  - ```
    nowy_typ a;
    ```
- Klasa, a obiekt
  - Klasa to typ obiektu, a nie sam obiekt

Składniki klasy

- Składniki definiujemy w ciele klasy
  - ```
    class CPunkt {
    public:
            float x;
            float y;
    };
    ```
- Dostęp do składników
  - `obiekt.składnik`
  - `wskaźnik_do_obiektu->składnik`
    - `(*wskaźnik_do_obiektu).składnik`
  - `referencja.składnik`
- Przykład cpp_4.1
    ```

Funkcje składowe

- Składnikami klas mogą być także funkcje – nazywamy je metodami
  - Metody zwykle pracują na składowych klasy
  - Deklaruje się je w znany już sposób
    - `void licz();`
    - `float max(float, float);`
  - Nazwy deklarowane w klasie mają zakres ważności równy obszarowi całej klasy
    - Klasa nie ma początku ani końca

Enkapsulacja

- Wszystkie elementy klasy zostają zamknięte w jednym kontenerze
- Jest ona bardzo ważną cechą języka C++, ponieważ odzwierciedla sposób naszego myślenia o obiektach
  - Opisujemy składowe i metody do których możemy się odwoływać na rzecz konkretnych obiektów
  - Możemy używać metod o takich samych nazwach w stosunku do obiektów różnych typów. Nie jest to przeładowywanie nazw, ponieważ mamy różne zakresy nazw

Hermetyzacja informacji

- Składniki i metody zamknięte w klasie mogą być dostępne na zewnątrz, ale NIE MUSZĄ
- Do określania poziomu dostępu służą etykiety
  - **`private`** – oznacza, że składniki i metody są dostępne tylko z wnętrza klasy
  - **`protected`** – związane z dziedziczeniem (na razie traktowane jak **`private`**)
  - **`public`** – oznacza, że składniki i metody są dostępne zarówno z wnętrza klasy jak i spoza klasy
- Przykład cpp_4.2

Funkcje składowe – definicja

- Gdzie można zdefiniować funkcje składowe
  - Wewnątrz samej klasy, wtedy automatycznie stają się funkcjami **`inline`**
    - Definiuje się je wtedy tak jak zwykłe funkcje
  - Na zewnątrz klasy. W samej klasie funkcje zostają tylko zadeklarowane
    - **`typ_zwracany nazwa_klasy::funkcja(...) {...}`**
    - Nie są wtedy **`inline`**, ale można je takimi uczynić
- Przykład cpp_4.3

Poprawna deklaracja klasy

- Deklarację klasy umieszcza się w osobnym pliku ze względu na
  - Przejrzystość
  - Możliwość używania obiektów danej klasy w różnych modułach programu
    - Aby stworzyć obiekt danego typu musimy znać jego definicję
  - Wykorzystanie klas stworzonych przez innych programistów i umieszczonych w modułach bibliotecznych
  - Definicje prostych klas można w całości umieścić w pliku nagłówkowym
    - Nie dotyczy zajęć laboratoryjnych
- Przykład cpp_4.4

Zalecana notacja

- Nazwy składników **`private`** i **`protected`** mają przedrostek „m_" lub „_" pisane małymi literami
  - **`int _x;`**
- Funkcje składowe **`private`** i **`protected`** mają przedrostek „mf_" lub lub „_" ewentualnie bez przedrostka pisane małymi literami
  - **`void _licz(int m_x);`**
- Nazwy klas zaczynają się wielką literą, jeśli nazwa jest wieloczłonowa każdy człon zaczyna się od wielkiej litery
  - **`FileReader, TcpIpSender`**
- Można także używać tzw. notacji węgierskiej zapisu zmiennych
  - Polega na dołączeniu odpowiedniego przyrostka literowego, (oznaczającego jej typ) do nazwy zmiennej np. s – oznacza zmienną zawierającą łańcuch znaków, b – zmienną typu **`bool`**, i – **`int`**, itd..

Zalecana notacja...

- Nazwy plików z kodem źródłowym klas powinny być takie jak sama nazwa klasy
  - `Punkt.cpp`
  - `Punkt.h`
- Deklaracje przyjaźni umieszczamy na samej górze
- Potem umieszczamy instrukcja `typedef`
- Metody publiczne umieszczamy zaraz po ewentualnych deklaracjach przyjaźni i „typedef-ach"
- Składowe mają być niepubliczne i umieszczane na dole klasy
- W ciele klasy występują tylko deklaracje metod

Wskaźnik `this`

- Istnienie wskaźnika **`this`** ma związek z wywołaniem funkcji składowych klasy na rzecz określonego obiektu tej klasy
  - Funkcja składowa w pamięci umieszczana jest tylko raz
  - Musi wiedzieć na rzecz, którego obiektu danej klasy ma pracować
- W rzeczywistości do wnętrza funkcji przekazywany jest wskaźnik do konkretnego obiektu, który pokazuje funkcji na którym egzemplarzu obiektu tej klasy ma pracować
  - ```
    void punkt::Set(float x, float y)
    {
            this->_x = x; this->_y = y;
    }
    ```
  - Normalnie kompilator sam za nas wstawia wskaźnik **`this`**
  - Przykład kiedy sami musimy wstawić wskaźnik **`this`**
  - ```
    void punkt::Set(float _x, float _y)
    {
            this->_x = _x; this->_y = _y;
    }
    ```
- Wskaźnik **`this`** jest typu **`const`** i ma typ obiektu klasy, na który wskazuje

Konstruktor

- Konstruktor jest specjalną funkcja składową
  - Wywoływany jest automatycznie przy tworzeniu nowego obiektu
  - Służy głównie do nadawania wartości początkowych elementów składowych klasy
  - Nazwa konstruktora jest taka sama jak nazwa klasy
  - Konstruktor jest jedną z najczęściej przeładowywanych funkcji
  - Konstruktor nie jest obowiązkowy
  - Konstruktor nic nie zwraca
    - Nawet `void`!!!

Destruktor

- Destruktor jest również specjalną funkcją składową klasy
  - Jest przeciwieństwem konstruktora
  - Jest uruchamiany automatycznie tak jak konstruktor, ale przy likwidowaniu obiektu
  - Służy np. do zwalniania pamięci
  - Destruktor nazywa się tak jak nazwa klasy z tą różnicą, że poprzedzona jest znakiem '~'
  - Destruktor również nie jest obowiązkowy
  - Destruktor nic nie zwraca
    - Nawet `void`!!!
- Przykład cpp_4.5

Składnik statyczny

- Składnik statyczny jest użyteczny kiedy chcemy, aby poszczególne obiekty danej klasy posługiwały się tą samą daną
- Taka dana jest tworzona w pamięci jednokrotnie i istnieje nawet jeśli nie ma ani jednego obiektu danej klasy
- W większości przypadków eliminuję potrzebę używania zmiennych globalnych
- Składnik statyczny deklaruje się używając słowa kluczowego **`static`**
- Definicję należy umieścić tak żeby miała zakres pliku
- Można zadeklarować nawet zmienną **`static`**, która jest prywatna
  - Inicjalizujemy w normalny sposób, gdzieś w zakresie pliku, ale dostęp mam już tylko z wnętrza klasy
- Sposoby odwoływanie się do składnika statycznego
  - **`obiekt.składnik`**
  - **`wskaźnik->składnik`**
  - **`klasa::składnik //najczęstszy`**

Statyczna funkcja składowa

- Statyczną funkcję składową wykorzystuje się do operowanie na składowych statycznych
- Funkcję statyczna można wywołać nie tylko na rzecz określonego obiektu klasy, ale także na rzecz samej klasy
- Nie jest możliwe odwołanie się do nie-statycznego składnika klasy
- Do statycznej funkcji składowej nie jest wysyłany wskaźnik `this`
- Praktycznie we wszystkich potrzebnych przypadkach funkcje statyczne zastępują funkcje globalne znane z C
- Przykład cpp_4.6

Funkcje składowe typu `const`

- Funkcja która jest **`const`** zapewnia, że jeśli wywoła się ją na rzecz jakiegoś obiektu to nie będzie modyfikować jego danych składowych
  - Wyjątek?
- Jeśli obiekt danej klasy jest **`const`** to na jego rzecz możemy wywołać tylko funkcje składowe, które zagwarantują że nie wykonają modyfikacji tego obiektu
- Składową funkcję typu **`const`** deklaruje się następująco
  - **`typ nazwa_fn(typ) const;`**
- Konstruktory i destruktory nie mogą mieć przydomka **`const`**
- Przykład cpp_4.7

Funkcje zaprzyjaźnione

- Funkcja zaprzyjaźniona z klasą to tak funkcja, która ma dostęp do wszystkich składowych klasy mimo iż nie jest składnikiem tej klasy
- Funkcja może być przyjacielem więcej niż jednej klasy
- Dzięki funkcjom zaprzyjaźnionym możemy dawać dostęp do zmiennych prywatnych funkcjom, które nie mogłyby być funkcjami składowymi
- Do funkcji zaprzyjaźnionej nie jest przesyłany wskaźnik **`this`**
- Deklaracja funkcji zaprzyjaźnionej
  - **`friend typ nazwa (...);`**
  - Miejsce deklaracji nie ma znaczenia (**`private`**, **`proteced`** lub **`public`**)
- Przykład cpp_4.8

Funkcje zaprzyjaźnione ...

- Funkcja zaprzyjaźniona może zostać zdefiniowana wewnątrz klasy
  - Jest wtedy **`inline`**
  - Ale dalej jest zwykłą funkcją, a nie funkcja składową klasy
- W wypadku przeładowywania nazw funkcja zaprzyjaźniona to ta, której parametry dokładnie odpowiadają parametrom z deklaracji przyjaźni
- Funkcja zaprzyjaźniona może być także funkcją składową innej klasy

Klasy zaprzyjaźnione

- Wszystkie funkcje składowe jednej klasy mogą być zaprzyjaźnione z drugą klasą
  - Można po kolei deklarować przyjaźń dla każdej funkcji z osobna
  - Można też napisać deklarację przyjaźni z klasą
    - `friend (class) nazwa_klasy`
- Przyjaźń klas jest jednostronna
  - Wszystkie funkcje przyjaciela mają dostęp do zmiennych prywatnych danej klasy, ale nie na odwrót

Klasy zaprzyjaźnione...

- Możliwa jest przyjaźń dwustronna
  - Problem, przy deklaracji z funkcją składową innej klasy kompilator musi znać wcześniej deklarację tej klasy
  - Rozwiązaniem jest deklaracja zapowiadająca całej klasy

```
class Pierwsza;
class Druga {
      friend class Pierwsza; ...};

class Pierwsza {
      friend class Druga; ...};
```

- Przyjaźń nie jest przechodnia
  - Jeżeli klasa A deklaruje przyjaźń z klasą B, a klasa B z klasą C to nie oznacza, że klasa A uznaję klasę C za przyjaciela
- Przyjaźń nie jest dziedziczona
- Przykład cpp_4.9

Klasy i tablice, referencje oraz wskaźniki

- Obiekty danej klasy mogą być przechowywane w tablicach tak jak obiekty typów wbudowanych
- Odwołujemy się do nich w znany już sposób
  - ```
    CPoint a[10];
    a[0].skladnik = 1;
    ```
- Przy inicjalizacji za pomocą listy należy pamiętać, że przecinek oddziela inicjalizacje poszczególnych obiektów, a nie składników tworzonych obiektów
  - Z wyjątkiem agregatów
- Można także używać referencji oraz wskaźników do poszczególnych obiektów danej klasy
- Jeżeli tworzymy tablicę obiektów reprezentowaną przez wskaźnik przy pomocy operatora **new** to dana klasa musi zawierać domyślny (bezargumentowy) konstruktor
  - Nie można używać inicjalizacji tak jak w przypadku zwykłych tablic
- Przykład cpp_4.10 i cpp_4.11
    ```

Struktury

- Struktura to inaczej klasa, w której przez domniemanie wszystkie składniki i metody są publiczne
  - Dotyczy języka C++
- W języku C struktura nie zawiera metod
- Definicja
  - ```
    struct nazwa {
            int i; int j;
    };
    ```
  - ```
    class nazwa {
    public:
            int i; int j;
    };
    ```
- Przykład cpp_4.12

Unia

- Unia jest pewnego rodzaju pudełkiem, w którym można trzymać obiekty różnych typów
  - Ale w danym momencie tylko jedną zmienną
- Unia zajmuje w pamięci tyle miejsca ile zajmuje największy obiekt, który można w niej trzymać
- Definicja
  - ```
    union nazwa {
              int i; long l; char c;
    }; // rozmiar long
    ```
- Unia anonimowa
  - Nie posiada nazwy
  - Do składników odwołujemy się bez operatora '.'
- Należy pamiętać, że po zapisaniu do unii obiektu jakiegoś typu należy odczytywać ten sam typ
- Przykład cpp_4.13
    ```

Pola bitowe

- Pola bitowe to specjalny typ składnika klasy polegający na tym, iż dane przechowywane są na określonej liczbie bitów
- ```
  class CPort{
      unsigned int read : 1;
      unsigned int status : 2;
  };
  ```
- Raczej nie stosuje się, chyba że w specjalnych sytuacjach, gdzie konieczne jest duże upakowanie danych lub obsługa specjalistycznej aparatury
- Dużo zależy od implementacji
  - Np. przełamywanie bajtów, kolejność umieszczania, interpretacja znaku
- Przykład cpp_4.14
  ```