03. Wprowadzenie do klas w języku C++

1 Stos jako struktura

Stos (ang. *stack*) jest jedną z podstawowych struktur danych, przechowującą i udostępniającą elementy według strategii LIFO (ang. *Last In, First Out*) — ostatni na wejściu, pierwszy na wyjściu. To znaczy, że nowe elementy są dokładane na wierzch stosu, a zdejmowany jest zawsze element umieszczony na stosie jako ostatni.

Przyjęło się, że stos powinien implementować następujące funkcje:

- `push(element)` - dodanie nowego elementy na wierzch stosu,
- `pop()` - ściągnięcie ostatniego elementu,
- `empty()` - sprawdzenie czy stos jest pusty.

Zadanie 01 Zapoznaj się z kodem w pliku źródłowym *stack.c*. Skompiluj go używając polecenia `gcc` i uruchom.

```
$ gcc stack.c -o structstack.out
$ ./structstack.out
```

Zadanie 02 Sprawdź co się stanie, jeżeli odkomentujesz linię 33 i uruchomisz program. Czy użytkownik powinien mieć możliwość wykonania takiej czynności? Co się stanie po odkomentowaniu linii 34?

Odpowiedz na pytanie jakie niedogodności, ograniczenia oraz potencjalne niebezpieczeństwa wiążą się ze strukturami jako typem danych?

2 Definicja klasy

Klasy (ang. *class*) łączą w sobie opis struktury danych z opisem operacji na niej wykonywanych. Dane składowe są nazywane *atrybutami* (ang. *attributes*), natomiast funkcje zaimplementowane wewnątrz klasy *metodami* (ang. *methods*).

```
class Foo {           // nazwa klasy
  public:
    int attr;         // atrybut
    void method() {   // metoda
      std::cout << "attr=" << attr << std::endl;
    }
};
```

W ciałach metod wewnątrz definicji klasy dozwolony jest dostęp do wszystkich składowych klasy (zarówno atrybutów, jak i innych metod).

Egzemplarze danej klasy to *obiekty* (ang. *objects*) lub *instancje klasy* (ang. *class instances*). Każdy obiekt ma swój własny stan, czyli własne wartości atrybutów. Dostęp do metod i atrybutów klasy z poziomu utworzonego obiektu uzyskuje się poprzez operator kropki `.` lub strzałki `->` w przypadku wskaźników (analogicznie jak w przypadku struktur).

```
Foo foo;            // utworzenie obiektu klasy Foo

foo.attr = 15;      // dostep do atrybutu
foo.method();       // wywolanie metody
```

Obiekty mogą „komunikować się" między sobą jedynie przy pomocy zdefiniowanych wcześniej metod. Wysłanie komunikatu do obiektu, aby wykonał on pewną operację, np. zmienił swój stan (tj. zmiana wartości atrybutu), odpowiada wywołaniu odpowiedniej metody.

Zadanie 03 Zapoznaj się z kodem źródłowym w pliku `foo_simple.cpp` zawierającym przykład definicji klasy oraz jej użycia. Skompiluj program przy użyciu kompilatora `g++`. Czy jest możliwe skompilowanie go przy pomocy kompilatora `gcc`?

Zadanie 04 Przerób implementację stosu z pliku *stack.c* na wersję obiektową. Możesz wykorzystać szablon z pliku *stack.cpp*.

3 Konstruktor i destruktor

Konstruktor (ang. *constructor*) to specjalna metoda wywoływana automatycznie podczas tworzenia obiektu, która w szczególności umożliwia nadanie początkowych wartości składowym klasy. Nazwa konstruktora jest taka sama jak nazwa klasy.

```
class Foo {
  public:
    Foo() {         // konstruktor bezargumentowy
      attr = 0;     // nadanie poczatkowej wartosci atrybutowi
    }
  // ...
```

Destruktor (ang. *destructor*) jest specjalną metodą wywoływaną przed usunięciem obiektu z pamięci. Nazwa destruktora jest taka sama jak nazwa klasy dodatkowo poprzedzona znakiem tyldy.

```
class Foo {
  public:
    ~Foo() {
      std::cout << "destroying object" << std::endl;
    }
  // ...
```

Zadanie 05 Do klasy `Stack` dodaj bezargumentowy konstruktor w miejsce metody `init()`, który zainicjalizuje atrybut `n` wartością 0 oraz wypisze komunikat *„tworzę nowy stos"*.

Zadanie 06* Do klasy `Stack` dodaj dwuargumentowy konstruktor przyjmujący tablicę liczb całkowitych, które powinny zostać dodane do stosu podczas jego tworzenia w takiej kolejności, w jakiej występują w tablicy. Drugim argumentem jest rozmiar tablicy. Dodanie konstruktora powinno umożliwić następujące tworzenie obiektu:

```
int tab[3] = {5, 10, 15};

Stack stack(tab, 3);
stack.pop()            // zwraca 15
```

Zadanie 07 Do klasy `Stack` dodaj destruktor, który wypisze komunikat *„niszczę stos z X elementami”*, gdzie X to liczba aktualnie przechowywanych elementów w stosie.

4 Hermetyzacja

Hermetyzacja lub *enkapsulacja* (ang. *encapsulation*) realizuje założenie programowania obiektowego polegające na publicznym udostępnianiu jedynie tych elementów programu, które są niezbędne do prawidłowego działania jego funkcjonalności. W przypadku klas jest to najczęściej ukrywanie pewnych atrybutów i metod tak, aby były one dostępne tylko dla metod wewnętrznych. Innymi słowy, programista powinien móc zrobić z obiektem klasy tylko niezbędne minimum wymagane do jej właściwego użycia i to wyłącznie za pomocą udostępnionych mu metod, aby zminimalizować ryzyko wystąpienia błędu.

W przeciwieństwie do struktur, wszystkie składowe klasy domyślnie są prywatne.

Zadanie 08 Odpowiedz na pytanie jakie korzyści niesie ze sobą hermetyzacja?

Które składowe w klasie `Stack` nie powinny być udostępniane publicznie? Podziel elementy klasy na sekcję publiczną i prywatną przy pomocy słów kluczowych `public` i `private`.

Zadanie 09 Do klasy `Stack` dodaj metodę `int size()`, która będzie zwracać liczbę elementów na stosie, oraz metodę `int top()`, która będzie zwracać ostatni element stosu bez usuwania go.

Zadanie 10 Zaprojektuj klasy (określając ich atrybuty, przykładowe metody, widoczność składowych, przydatne konstruktory, klasy wewnętrzne itd.) mające reprezentować następujące byty:

- punkt w przestrzeni trójwymiarowej,
- samochód,
- wycieczkę w systemie rezerwacji podróży,
- równoległobok w układzie współrzędnych,
- czas i datę.

Zadanie domowe 11 (1 pkt) Zmień wewnętrzną implementację klasy `Stack` na wersję listową. Stos powinien posiadać identyczne funkcjonalności jak ten prezentowany na ćwiczeniach realizowane przez te same metody publiczne *pop, push, empty, size, top* (innymi słowy nie powinno być konieczności zmiany kodu w funkcji `main` programu).

Różnicą będzie wykorzystanie w wewnętrznej implementacji własnej klasy `ListElem` (element listy jednokierunkowej), dzięki czemu zniknie ograniczenie na liczbę elementów w stosie.

Pamiętaj o zwalnianiu pamięci w destruktorze.