Wrocław 2017.09.28

Autor: Michał Przewoźniczek
Konsultacja: Dariusz Konieczny, Marcin Komarnicki

Zaawansowane Metody Programowania Obiektowego – zadanie 1
Alokacja i dealokacja

UWAGI:

1. **Pisząc własny program można użyć innego nazewnictwa niż to przedstawione w treści zadania. Należy jednak użyć jakiejś spójnej konwencji kodowania, zgodnie z wymaganiami kursu.**
2. **Program NALEŻY NAPISAĆ STRUKTURALNIE, bez użycia mechanizmów programowania obiektowego.**
3. **Kod implementujący interfejs programu NIE MUSI być autorskim dziełem studenta. Może być efektem pracy zespołowej, kodem wzorowanym lub ściągniętym z określonego miejsca w sieci. Musi jednak spełniać wymagania funkcjonalne, oraz wymagania dotyczące jakości w ramach kursu (np. użycie wszelkich instrukcji łamiących przepływ sterowania [rzucanie wyjątków, użycie continue i break wewnątrz pętli, goto, itd.] będzie karane).**

C++ pozwala na użycie mechanizmów programowania obiektowego, ale pozwala również na programowanie strukturalne. Jedną z zalet C++ jest możliwość łączenia podejścia strukturalnego i obiektowego. Program w ramach zadania nr 1 należy napisać całkowicie strukturalnie, bez użycia klas i struktur. Używane mają być wyłącznie zmienne i odpowiednie funkcje.

Należy oprogramować zestaw funkcji, pozwalający na obsługę wektorów rzadkich. Wektor rzadki to taki wektor, który w większości zawiera tą samą wartość (np. zero). W związku z tym żeby przechowywać wartości wektora rzadkiego wystarczy przechować tylko te pozycje, które mają wartość inną niż domyślna.

Wymagania funkcjonalne:

Zestaw funkcji ma pozwalać na zrealizowanie następujących operacji:

- Zmianę długości wektora rzadkiego
- Określenie domyślnej wartości dla wektora rzadkiego (tylko jeśli znajdują się w nim wyłącznie wartości domyślne, w przeciwnym wypadku operacja ma zwrócić informację o niepowodzeniu operacji)
- Przypisanie określonej komórce wektora wartości typu int
- Odczyt wartości określonego elementu wektora rzadkiego
- Zwrócenie informacji o obiekcie do zmiennej typu string w formacie: (len: <liczba pozycji> values: <wszystkie wartości z tablicy oddzielone przecinkami>)

  Na przykład: „len: 10 values: 0,0,0,0,1,0,2,0,3,0”

- Wartości elementów, które są różne od domyślnych, mają być przechowywane w dwóch dynamicznie alokowanych tablicach. Pierwsza ma przechowywać wartości, a druga offsety na przykład:

Domyślna wartość to zero. Długość wektora rzadkiego to 100. Tablica wartości może przechowywać: 2,3,4,5, a tablica offsetów: 10, 20, 30, 40, 50. Wtedy wartości to same zera z wyjątkiem pozycji 11, 21, 31, 41 i 51 (offsety numeruje się od zera), które przyjmują wartości, odpowiednio: 2,3,4,5.

- Przechowywane mają być wyłącznie te wartości, które są różne od zadanej wartości domyślnej. Na przykład, jeżeli wartością domyślną wektora rzadkiego jest zero, a elementu o offsecie 5 ma wartość 4, to informacja o tym jest przechowywana w tablicy wartości i offsetów. Jeżeli wartość elementu o offsecie 5 zostanie zmieniona na 0, to ta informacja nie ma być przechowywana w tablicach.

- Program musi pozwalać na obsługę dowolnie długich wektorów rzadkich, o dowolnej liczbie pozycji o wartościach różnych od domyślnej. Oznacza to, że tablice wartości i offsetów muszą być realokowane przy dodaniu nowej pozycji o wartości różnej od domyślnej (nie musi się to odbywać za każdym razem, ale wtedy kiedy pojemność tablicy wartości i offsetów zostanie przekroczona). Na przykład:

Załóżmy, że tablica wartości nazywa się pi_values, a tablica offsetów pi_offsets. Załóżmy że zaalokowano je dla 4 elementów, ale obecnie są tam przechowywane tylko 3, np.: pi_table przechowuje 4,5,6, a offsety tych wartości to 10, 20, 30. Jeżeli chcemy dołożyć kolejną wartość, to nie jest potrzebna realokacja tablicy. Jednak dołożenie 5-tej wartości będzie wymagać zaalokowania większych tablic pi_values, pi_offsets. Zauważ, że w takiej sytuacji, zamiast alokować tablice 5-elementowe, można od razu zaalokować tablice 8-elementowe (na przykład). Dzięki temu realokacje będą wykonywane rzadziej.

- Zestaw funkcji do obsługi wektora rzadkiego musi zawierać funkcję, która zwalnia pamięć wszystkich dynamicznie zaalokowanych zmiennych (w tym tablicy wartości i offsetów).

Program musi spełniać wszystkie powyższe wymogi. Niespełnienie dowolnego z nich oznacza, że za zadanie nie zostaną przyznane żadne punkty.

Uwagi:

1. Proszę pamiętać, że wartości domyślne (nie chodzi o wartość domyślną wektora rzadkiego, ale np. o domyślną długość wektora rzadkiego na starcie programu) powinny być przechowywane w stałych. W przeciwnym wypadku będzie to traktowane jak błąd (otrzymacie Państwo mniej punktów).
2. Funkcje do obsługi wektora rzadkiego nie mogą korzystać ze zmiennych globalnych. Wszystkie parametry muszą zostać przekazane do funkcji poprzez jej interfejs. Korzystanie ze zmiennych globalnych oznacza, że program zostanie oceniony na niższą ocenę.

Wymagania dotyczące interfejsu

Uwaga: ta część programu nie musi być wykonana samodzielnie, musi jednak spełniać wszystkie wymogi, które są stawiane kodowi źródłowemu, oraz musi spełniać wszystkie wymagania funkcjonalne. Kod sterujący interfejsem może indywidualnym dziełem studenta, grupy studentów, może być to kod ściągnięty z Internetu (pod warunkiem, że nie narusza to obowiązującego prawa).

Interfejs ma pozwolić na sterowanie programem za pomocą konsoli. Obsługiwane mają być następujące polecenia.

mvec <len> <def> - wykonanie polecenia usuwa istniejący wektor rzadki (jeśli jakiś istnieje) i tworzy nowy wektor, o długości <len> i wartości domyślnej wektora rzadkiego <def>
len <len> - zmiana długości wektora rzadkiego. Jeśli zostaje zmniejszona to, to te pozycje, które inne niż domyślna, a nie mieszczą się w nowym zakresie powinny zostać usunięte z tablicy wartości i tablicy offsetów.
def <off> <val> - ustalenie wartości <*val*> dla offsetu <*off*> wektora rzadkiego
print – wykonanie polecenia wypisuje na ekran aktualny stan wektora rzadkiego
del – usuwa wszystkie dynamicznie alokowane zmienne dla wektor rzadkiego, jeśli wektor obecnie istnieje.

W przypadku wpisania błędnego polecenia, polecenia, które jest niewykonalne, lub posiada błędne parametry program ma poinformować o tym użytkownika.

Przykład:

Wykonanie komend:

mvec 50 1	(utworzenie wektora rzadkiego o długości 50 i domyślnej wartości 1)
def 30 0	(nadanie wartości 0 dla 30 offsetu wektora rzadkiego)
def 40 8	
def 50 8	(offset o wartości 50 nie istnieje – informacja o błędzie)
len 35	(zmniejszenie długości wektora, niektóre pozycje wypadają z tablicy wartości i tablicy offsetów)
def 20 1	(nic nie robi, bo 1 to wartość domyślna)
def 30 1	(1 to wartość domyślna, więc informacja o wartości offsetu 30 jest usuwana z tablicy wartości i tablicy offsetów)
print	
del	(skasowanie wszystkich dynamicznie zaalokowanych zmiennych)