

## **Zaawansowane Metody Programowania Obiektowego – zadanie 2** **Alokacja, konstruktory, destruktory, zaprzyjaźnianie**

### **UWAGA:**

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 OBIEKTOWO.**
3. **Podobnie jak w zadaniu nr 1, 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).**

Oprogramować klasę „CSparseMatrix”, do obsługi wielowymiarowych maczy rzadkich. Maczy rzadka to taka maczy, która w większości zawiera tą samą wartość (np. zero). W związku z tym żeby przechowywać wartości maczy rzadkiej wystarczy przechować te pozycje, które mają wartość inną niż domyślna.

### **Wymagania funkcjonalne dla klasy**

Klasa CSparseMatrix ma pozwalać na zrealizowanie następujących operacji:

- Określenie liczby wymiarów maczy rzadkiej i określenie zakresów dla poszczególnych wymiarów przed wypełnieniem maczy wartościami
- **Liczba wymiarów, podobnie jak zakres mogą być dowolnie duże**
- Określenie domyślnej wartości dla maczy rzadkiej
- Przypisanie określonej komórce maczy wartości typu int
- Odczyt wartości określonej komórki
- Utworzenie klonu obiektu (innego obiektu posiadającego te same wartości w tablicy)
- Przypisanie obiektowi A, wartości i stanu tabeli w obiekcie B (po wykonaniu takiej operacji w obiekcie A tabela ma posiadać tą samą długość i te same wartości, co tablica w obiekcie B)
- Klasa musi pozwalać na nadanie dowolnemu obiektowi nazwy (wartości typu string)
- Klasa musi pozwalać na odczyt nazwy obiektu
- Klasa musi posiadać konstruktory:
  - Domyślny (nadający obiektowi domyślną nazwę)
  - Parametrowy (pobierający nazwę obiektu z parametru)
  - Kopiujący (nazwa obiektu jest kopiowana i jest do niej doklejany dodatkowy tekst „\_copy”)

Na przykład: konstruktor domyślny nadaje nazwę „def\_name”, a konstruktor kopiujący, który jako parametr otrzyma obiekt o nazwie „test” utworzy obiekt o nazwie „test\_copy”

- Każdy konstruktor ma wyprowadzać na ekran napis: „create: <nazwa obiektu>”
- Destruktor ma wyprowadzać na ekran napis: „destroy: <nazwa obiektu>”
- Zwrócenie informacji o obiekcie do zmiennej typu string w formacie: („size: [<długości poszczególnych wymiarów>] values: [<współrzędne>]:<wartość> ”oddzielone

średnikami). Na przykład: „size: [2,3,1] values: [0,0,0]:0; [0,1,0]:0; [0,2,0]:2; [1,0,0]:0; [1,1,0]:3; [1,2,0]:0;”

### Wymagania dotyczące wykonania klasy CSparseMatrix

- Klasa ma przechowywać wyłącznie te wartości, które są różne od zadanej wartości standardowej
- Wartości, komórek, które są różne od standardowych mają być przechowywane w jednowymiarowej tablicy obiektów klasy CSparseCell
- Należy rozważyć, czy tablica obiektów klasy CSparseCell powinna przechowywać te obiekty bezpośrednio (np. jako CSparseCell \*pc\_defined\_cells), czy też jako wskaźniki do dynamicznie alokowanych obiektów (np. jako CSparseCell \*\*pc\_defined\_cells). **Należy wiedzieć jaka jest różnica pomiędzy tymi rozwiązaniami, jakie są ich zalety i wady.**
- Wymagania do klasy CSparseCell:
  - Przechowuje koordynaty i wartość komórki w macierzy rzadkiej, która ma inną wartość niż domyślna
  - **Klasa CSparseCell ma być zaprzyjaźniona z klasą CSparseMatrix, klasa CSparseMatrix ma wykorzystywać ten fakt**
- Klasa musi pozwalać na obsługę macierzy rzadkich o dowolnej liczbie wymiarów i dowolnych zakresach dla tych wymiarów. Oznacza to, że klasa CSparseMatrix musi realokować tablicę obiektów CSparseCell przy dodaniu nowej pozycji o wartości różnej od domyślnej (podobnie jak w zadaniu numer 1 nie musi się to odbywać za każdym razem, ale wtedy kiedy pojemność tablicy wartości i offsetów zostanie przekroczona).
- Klasa musi posiadać destruktor, który usuwa tablicę obiektów CSparseCell

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

**Uwaga:** proszę pamiętać, że wartości domyślne powinny być przechowywane w stałych. W przeciwnym wypadku będzie to traktowane jak błąd (otrzymacie Państwo mniej punktów).

**Program musi posiadać tekstowy interfejs użytkownika, podobny jak w zadaniu nr 1, który będzie pozwalał na:**

- Dynamiczne utworzenie dowolnej liczby obiektów typu CSparseMatrix
- Określenie/modyfikację liczby wymiarów i ich zakresu tablicy dla dowolnego z utworzonych dynamicznie obiektów CSparseMatrix
- Skasowanie dowolnego dynamicznie utworzonego obiektu typu CSparseMatrix
- Skasowanie wszystkich dynamicznie utworzonych obiektów typu CSparseMatrix
- Nadanie nowej nazwy dowolnemu z dynamicznie utworzonych obiektów CSparseMatrix.
- Sklonowanie dowolnego dynamicznie utworzonego obiektu CSparseMatrix i dodanie klona do listy/puli dynamicznie utworzonych obiektów klasy CSparseMatrix
- Wypisanie na ekran dowolnego dynamicznie utworzonego obiektu CSparseMatrix (należy użyć metody zwracającej stan obiektu CSparseMatrix w zmiennej typu string)
- Program ma być odporny na błędy użytkownika (np. w przypadku, gdy wskaże on obiekt CSparseMatrix spoza dostępnego zakresu)
- Dynamicznie tworzone obiekty CSparseMatrix można przechowywać w dowolny sposób (np. w tablicy, lub w wektorze). Musi on jednak zapewniać możliwość zdefiniowania dowolnej liczby obiektów.
- W przypadku wpisania błędnego polecenia, polecenia, które jest niewykonalne, lub posiada błędne parametry program ma poinformować o tym użytkownika.

**Zastanów się, jaka część programu powinna być odpowiedzialna za tworzenie, przechowywanie i zarządzanie obiektami matryc rzadkich. Czy powinna zajmować się tym warstwa interfejsu, czy też może potrzebny jest do tego oddzielny obiekt?**

**Należy zwrócić uwagę na to, że wszystkie dynamicznie utworzone obiekty powinny być skasowane, po zakończeniu działania programu. Pominięcie skasowania obiektów będzie traktowane jak błąd funkcjonalności i spowoduje brak punktów za zadanie.**

**Polecenia, które można wprowadzić do konsoli.**

*addmat* <dimNum> <dim0size> <dim1size> ... <dimNum-1size> <def> <!name!> - wykonanie polecenia dynamicznie tworzy nową matrycę rzadką i dodaje ją do puli (zgodnie z wytycznymi utworzone matryce można przechowywać w wektorze, liście lub tablicy). <dimNum> oznacza liczbę wymiarów nowej matrycy, <dimXsize> oznacza zakres dla danego wymiaru, <def> to wartość, <!name!> to nazwa matrycy rzadkiej, która **może, ale nie musi** zostać podana. W przypadku braku zdefiniowanej nazwy, obiekt matrycy rzadkiej powinien zostać stworzony z nazwą domyślną (np. „SparseMatrix”).

*list* – wypisanie na ekran liczby matryc rzadkich przechowywanych obecnie przez program, wraz z ich nazwami w formacie: „<MatNum> matrices:\n[<off>]: <name> size: [<długości poszczególnych wymiarów>]\n”, gdzie *MatNum* to liczba matryc, *off*, to offset danej matrycy, *długości poszczególnych wymiarów* są tak zdefiniowane jak przy wypisywaniu pojedynczej matrycy na ekran, a /n oznacza znak końca linii. Na przykład:

2 matrices:

[0] – „wektor” size: [58]

[1] – „testowa” size: [2, 67, 33]

*del* <matoff> - skasowanie matrycy przechowywanej na offsecie <matoff>

*delall* – skasowanie wszystkich obecnie przechowywanych matryc  
*def* <matoff> <dim0> <dim1>... <dimNum-1> <val> - ustalenie wartości <val> dla matrycy rzadkiej o offsecie <matoff> i koordynatów <dim0> <dim1>... <dimNum-1>  
*print* <matoff> – wykonanie polecenia wypisuje na ekran aktualny stan matrycy rzadkiej o offsecie <matoff>  
*clone* <matoff> - sklonowanie matrycy o offsecie <matoff> i dodanie jej do puli matryc na ostatniej pozycji  
*rename* <matoff> <newName> - zmiany nazwy matrycy rzadkiej o offsecie <matoff>

### Przykład:

*Addmat 5 20 30 40* (próba dodania nowej matrycy – nieudana za mało danych dla 5 wymiarów)  
*Addmat 3 20 30 40 1 pierwsza* (próba dodania nowej matrycy – udana)  
*Addmat 1 100 0 wektor*  
*Addmat 4 100 100 100 100 4* (próba dodania nowej matrycy – udana, nazwa jest domyślna)  
*list* (wypisze na ekran jak poniżej)

3 matrices:

[0] – pierwsza size: [20, 30, 40]  
[1] – wektor size: [100]  
[2] – Sparse Mat size: [100 100 100 100]

*rename 4 test* (próba zmiany nazwy matrycy – nieudana)  
*rename 2 test* (próba zmiany nazwy matrycy – udana)  
*Addmat 1 5 0 wektor2*  
*del 1* (usunięcie matrycy o nazwie wektor; uwaga offseety matryc powyżej 1 ulegają zmianie)  
*def 2 2 5* (przypisanie 3 elementowi wektora2 wartości 5)  
*def 0 5 30 20 67* (próba przypisanie przypisania wartości 67 w matrycy pierwsza – nieudana, bo wartość offsetu drugiej współrzędnej za duża)  
*def 0 5 29 20 67* (próba przypisanie przypisania wartości 67 – udana)  
*clone 2* (sklonowanie matrycy wektor2 i dodanie jej na koniec puli matryc)  
*print 3* (wypisze na ekran jak poniżej)

size: [5] values: [0]:0; [1]:0; [2]:5; [3]:0; [4]:0;

*delall*