Laboratorium Informatyki II
Ćwiczenie 4

1 Dynamiczna alokacja zmiennych

W programach pisanych w języku C++ dynamiczną alokację zmiennych w pamięci należy realizować w oparciu o operatory `new` i `delete`. Pierwsza przypisuje do wskaźnika adres pierwszej komórki przydzielanego bloku pamięci. Druga z nich zwalnia przydzielony obszar. Operatory są więc odpowiednikami stosowanych w języku C instrukcji `malloc()` i `free()`.

Za pomocą operatora `new` można alokować dynamicznie zmienne i tablice. Procedura przebiega podobnie jak w języku C. W pierwszym kroku deklarowany jest wskaźnika na wybrany typ, w kolejnym przydzielany jest obszar pamięci. Należy także pamiętać o zwolnieniu pamięci gdy zmienna nie jest już potrzebna.

Utworzony instrukcją `int *wsk;` wskazuje losową komórkę pamięci, która może być używana przez dowolny program lub system operacyjny. Operator `new` rezerwuje i przypisuje do wskaźnika wolny obszar pamięci.

```
int *wsk1 = new int[5];
int *wsk2 = new int;
int *wsk3;
wsk3 = new int[5];

delete[] wsk1;
delete wsk2;
delete[] wsk3;
```

Do wskaźnika `wsk1` przypisano obszar odpowiadający pięciu komórkom pamięci typu `int`, może więc być traktowany jak tablica. Natomiast wskaźnik `wsk2` do którego przypisano pojedynczą komórkę typu `int` używać należy jak zwykły wskaźnik. Zwalniając pamięć tablicy dynamicznej należy po operatorze `delete` umieścić pusty nawias kwadratowy.

Poniższy przykład pokazuje zastosowanie dynamicznej alokacji pamięci do utworzenia tablicy liczb całkowitych i zapełnienia jej danymi pobranymi od użytkownika.

```
#include <iostream>
int dane(int);
int main()
{
    int *tab,n;
    std::cout << "Podaj rozmiar tablicy: ";
    std::cin >> n;
    tab = new int[n];
    for(int i=0;i<n;i++)
```

```
        tab[i] = dane(i);
    std::cout << "tab={";
    for(int i=0;i<n;i++){
        std::cout << tab[i];
        if(i<n-1) std::cout << ",";
        else std::cout << "}" << std::endl;
    }
    delete[] tab;
}
int dane (int i)
{
    std::cout << "Podaj [" << i+1 << "] = ";
    int x;
    std::cin >> x;
    return x;
}
```

W programie wykorzystano funkcje do pobierania danych z klawiatury. W pierwszym kroku pobierana jest informacja o rozmiarze tablicy, następnie za pomocą operatora `new` tworzona jest tablica o żądanym rozmiarze. Za pomocą pętli for i funkcji `dane()` tablica zapełniana jest danymi. W kolejnej pętli for dane są wyświetlane na ekranie. Program kończy operator `delete` zwalniający pamięć zajętą przez tablicę.

2 Dynamiczna alokacja tablicy wielowymiarowej

Podobnie jak w języku C i za pomocą funkcji `calloc()`, `malloc()` można z wykorzystanie operatora `new` deklarować tablice wielowymiarowe. W zależności od stopnia tablicy konieczne jest wykonanie właściwej liczby kroków. W przypadku tablicy 2D w pierwszym kroku tworzona jest tablica wskaźników do przechowania poszczególnych wierszy tablicy. W drugim dla każdego wiersza tworzona jest osobna tablica. Zwalnianie pamięci przebiega w kolejności przeciwnej, w pierwszej kolejności należy zwolnić pamięć zajmowaną przez poszczególne wiersze, a dopiero na koniec tablicę wskaźników wierszy, jak pokazano na poniższym przykładzie. Analogicznie jak w języku C można utworzyć tablicę w której każdy wiersz zawiera inną liczbę komórek.

```
#include <iostream>
int main()
{
    int **tab,w,k;
    std::cout << "Podaj liczbę wierszy w=";
    std::cin >> w;
    tab = new int*[w];
    std::cout << "Podaj liczbę kolumn k=";
    std::cin >> k;
    for(int i=0;i<w;i++)
        tab[i] = new int[k];
    std::cout << "tab[][]=" << std::endl;
    for(int i=0;i<w;i++){
        std::cout << "|";
        for(int j=0;j<k;j++){
            tab[i][j] = i*j+1;
```

```
            std::cout.width(3);
            std::cout << tab[i][j];
            if(j<k-1) std::cout << ",";
            else std::cout << "|" << std::endl;
        }
    }
    for(int i=0;i<w;i++)
        delete[] tab[i];
    delete[] *tab;
}
```

W analogiczny sposób można zdefiniować tablicę o większej liczbie wymiarów.

3 Zmiana rozmiaru tablicy

W języku C++ nie ma odpowiednika funkcji `realloc()`. Konieczne staje się opracowanie własnego rozwiązania. Projektując algorytm należy uwzględnić możliwość powiększania lub pomniejszania rozmiaru tablicy przy przenoszeniu danych z tablicy źródłowej.

```
#include <iostream>
void drukuj(int*,int);
int* zmiana(int*,int*);
int main()
{
    int n = 10;
    int *tab = new int[n];
    drukuj(tab,n);
    tab = zmiana(tab,&n);
    drukuj(tab,n);
    tab = zmiana(tab,&n);
    drukuj(tab,n);
    delete[] tab;
}
void drukuj(int *tab,int n)
{
    std::cout << "tab[" << tab << "]: ";
    for(int i=0;i<n;i++)
        std::cout << tab[i] << " ";
    std::cout << std::endl;
}
int* zmiana(int *tab,int *stary)
{
    int nowy;
    std::cout << "Podaj nowy rozmiar tablicy n=";
    std::cin >> nowy;
    int *newTab = new int[nowy];
    int n;
    nowy>*stary ? n=*stary:n=nowy;
    for(int i=0;i<n;i++)
        newTab[i] = tab[i];
```

```
    delete[] tab;
    *stary=nowy;
    return newTab;
}
```

W analogiczny sposób można zaprogramować zmianę rozmiaru tablicy wielowymiarowej.

Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać programy. Zastosować dynamiczną alokację pamięci za pośrednictwem operatorów `new` i `delete`.

a) Napisać program gromadzący dane kontaktowe (imię, nazwisko, tel., email, itp). Program wyświetla zgromadzone dane w postaci zapętlonej przeglądarki wizytówek.

b) Napisać program generujący tablicę 2D o zadanej przez użytkownika liczbie wierszy. Każdy wiersz zawiera liczbę komórek generowaną losowo w momencie tworzenia tablicy. Tablica zapełniana jest losowymi wartościami rzeczywistymi (typu `float` lub `double`) wygenerowanymi w zakresie <A,B> podanym przez użytkownika.

c) Rozbudować program z zadania b) o możliwość zmiany rozmiaru tablicy

- liczby komórek w poszczególnych wierszach
- liczby wierszy

Program przy zmianie rozmiaru zachowuje dane z tablicy bazowej i uzupełnia nowymi danymi przy powiększaniu.