Opracowanie: dr inż. Łukasz Maliński i mgr inż. Błażej Nycz, Politechnika Śląska, 2021

LABORATORIUM 1.3
DYNAMICZNA ALOKACJA PAMIĘCI

Poruszane zagadnienia z dziedziny programowania:

- Dynamiczna alokacja tablicy 1D - *„Opus Magnum C++11”*, **wyk. 3**
- Dynamiczna alokacja tablicy 2D - *„Opus Magnum C++11”*, **wyk. 3**
- Alokacja ciągła i fragmentaryczna - **wyk. 3**
- Wskaźniki wyższego stopnia - *„Opus Magnum C++11”*, **wyk. 3**
- Przekazywanie tablic dynamicznych do funkcji - **wyk. 3**
- Zwracanie adresu przez funkcję - **wyk. 3**

Umiejętności do opanowania:

- alokacja i zwalnianie pamięci na zmienną, tablicę 1D, lub instancję złożonego typu danych,
- weryfikacja poprawności alokacji pamięci,
- przekazywanie położenia pamięci alokowanej dynamicznie między różnymi funkcjami w programie, tak aby nie dopuścić przy tym do wycieku pamięci,
- proceduralne tworzenie złożonych wielowymiarowe tablic dynamicznych, (alokacja fragmentaryczna i ciągła),
- proceduralne rozpoznawanie typu/rozmiaru alokacji.

Oznaczenia odnośnie samodzielności pracy:

■ – **Ten problem koniecznie rozwiąż w pełni samodzielnie**. Możesz posługiwać się literaturą, wykładami i dokumentacją w celu sprawdzenia niuansów składniowych, ale nie szukaj gotowych rozwiązań i algorytmów. **Jest to bardzo ważne z punktu widzenia nauki i oszukiwanie przyniesie tylko poważne braki w późniejszych etapach nauki!**

▲ – Rozwiązując ten problem możesz posiłkować się rozwiązaniami zapożyczonymi od innych programistów, **ale koniecznie udokumentuj w kodzie źródło**. Nie kopiuj kodu bezmyślnie, tylko dostosuj go do kontekstu zadania. **Koniecznie musisz dokładnie zrozumieć rozwiązanie, które adoptujesz, gdyż inaczej wiele się nie nauczysz!**

● – Rozwianie tego problemu właśnie polega na znalezieniu i użyciu gotowego rozwiązania, **ale koniecznie podaj źródło**. Nie będzie wielkiej tragedii jeśli wykażesz się tylko ograniczonym zrozumieniem rozwiązania, gdyż nie musi być ono trywialne.

Przykładowe zadania do rozwiązania w ramach samodzielnej nauki (nie oceniane):

Zadanie A (z rozwiązaniem):

■ Korzystając z dynamicznej alokacji pamięci z C++ (new/delete), utwórz:

- zmienną typu double i przypisz jej dowolną wartość przez dereferencję,
- instancję struktury dowolnego typu (o dwóch polach) i ręcznie wypełnij ją treścią (nadaj wartości polom) już po utworzeniu instancji
- tablicę typu bool (n-elementów, podane przez użytkownika) i zainicjuj (przy tworzeniu) wszystkie jej elementy na 0.

■ Na koniec programu koniecznie zwolnij wszystkie alokacje pamięci.

Wskazówki: Pamiętaj, że inaczej zwalnia się pamięć dla pojedynczej zmiennej, oraz instancji, a inaczej dla tablicy. Zwróć uwagę, jak odwołujemy się do pól instancji struktury wskazywanej wskaźnikiem (jakim operatorem) . Struktura nie musi być wyszukana dwa pola, nawet tego samego typu (np.: int), w pełni wystarczą.

Przykładowe rozwiązanie:

```
struct MojTyp
{
  int poleA;
  int poleB;
};

void main(void)
{
  int n;
  cout << "Podaj rozmiar tablicy: "; cin >> n;

  //Dynamiczne alokacje zmiennych/tablic:
  double* ilePrzezyl = new double;
  MojTyp* pakietLiczb = new MojTyp;
  bool* tabelkaLogiczna = new bool[n] {};

/* Dość ważne jest, aby rozmiar tablicy był wyznaczany jakoś w trakcie działania programu.
Może być pobierany od użytkownika, lub wyliczany z jakiś innych danych. W zasadzie, jeśli
rozmiar da się jednoznacznie ustalić już na etapie kompilacji, to tworzenie tablicy dyna-
micznej jest często niepotrzebną komplikacją programu. */

  //Praca z danymi alokowanymi dynamicznie:
  *ilePrzezyl = 56.5;
  pakietLiczb->poleA = 2;
  pakietLiczb->poleB = 3;

/* Interakcja ze zmienną wymaga dereferencji wskaźnika (*). Interakcję z polem instancji
najwygodniej jest wykonać operatorem ->.*/

  //Zwolnienie pamięci :
  delete ilePrzezyl;
  ilePrzezyl = nullptr;

  delete pakietLiczb;
  pakietLiczb = nullptr;

  delete[] tabelkaLogiczna; // delete[] dla tablicy!
  tabelkaLogiczna = nullptr;

}
```

Zadanie B (z rozwiązaniem):

- Napisz program obliczający średnią arytmetyczną z ***n*** wczytanych liczb całkowitych, zgromadzonych w tablicy. Liczba ***n*** powinna być nie mniejsza niż 10 (proszę to wymusić) i podana na początku działania programu. Tablica przechowująca liczby powinna mieć dokładnie ***n*** elementów (być utworzona dynamicznie po zadaniu jej rozmiaru przez użytkownika programu).
- Przed liczeniem średniej, liczby w tablicy wygeneruj używając generatora liczb losowych, ale przed wpisywaniem wartości upewnij się, że pamięć została pomyślnie zaalokowana.
- Przed zakończeniem programu obowiązkowo zwolnij pamięć!

Wskazówki: Pamiętaj, aby nie przekroczyć rozmiaru tablicy przy wypełnianiu, gdyż uszkodzone zostaną znaczniki opakowujące tablicę w pamięci.

Przykładowe rozwiązanie:

```
void main(void)
{
  int n = 0;
  srand(time(0));

  //Pobranie liczby elemnetów z wymuszeniem:
  cout << "Podaj ilosc elementow: " << endl;
  while (n < 10)
    cin >> n;

  //Alokacja tablicy dynamicznej o n elementach:
  int* tab = new(std::nothrow) int[n];

  if (tab != nullptr) //Sprawdzenie poprawności alokacji
  {
    //Pobranie wartości poszczególnych liczb z sumowaniem:
    long suma = 0;
    for (size_t i = 0; i < n; i++)
    {
      tab[i] = 1 + rand() % 20;
      suma += tab[i];
    }

    //Wyliczenie średniej i wypisanie wyniku:
    cout << "Srednia: " << (double)suma / n << endl;

    //Zwolnienie pamięci po tablicy dynamicznej:
    delete[] tab;
    tab = nullptr;
  }
}
```

Zadanie C (z rozwiązaniem):

■ Napisz program wypełniający tablicę 2D tabliczką mnożenia. Zakres tabliczki ***n*** należy podać na początku programu i powinien być on liczbą nie mniejszą niż 5 (jeśli użytkownik poda mniejszą liczbę to przyjmij ***n*** = 5). Liczby zapisz w tablicy dynamicznej (alokacja fragmentaryczna) o wymiarach ***n*** × ***n***. Nie zapomnij o zwolnieniu pamięci na koniec.

Wskazówki: Pamiętaj, że zwalnianie pamięci alokowanej fragmentarycznie wymaga ostrożności w kolejności operacji. Najpierw zwolnij poszczególne wiersze tablicy, a potem dopiero tablicę wskaźników (kolejność odwrotna jak przy tworzeniu).

Przykładowe rozwiązanie:

```
void main(void)
{
  //Pobranie zakresu tabliczki mnożenia:
  cout << "Podaj zakres tabliczki mnozenia" << endl;
  int zakres;
  cin >> zakres;

  //Zastąpienie wartosci gdy podana jest mniejsza od 5 (wyrażenie warunkowe):
  zakres = (zakres < 5) ? 5 : zakres;

  //Alokacja fragmentaryczna dwuwymiarowej tablicy dynamicznej:
  unsigned long** tabMnozenia = new unsigned long* [zakres];
  for (size_t i = 0; i < zakres; i++)
    tabMnozenia[i] = new unsigned long[zakres] {};

  //Wypełnianie tablicy liczbami:
  for (size_t i = 0; i < zakres; i++)
  {
    for (size_t j = 0; j < zakres; j++)
```

```
    {
      tabMnozenia[i][j] = i * j;
      cout << tabMnozenia[i][j] << "\t";
    }
    cout << endl;
  }

  //Zwolnienie pamięci:
  for (size_t i = 0; i < zakres; i++)
    delete[] tabMnozenia[i];
  delete[] tabMnozenia;
  tabMnozenia = nullptr;
}
```

Zadania domowe (oceniane)

UWAGA: Można oddać tylko jedno zadanie domowe. Ocena za zadanie domowe jest uwzględniana tylko jeśli uzyska się przynajmniej 1,0 pkt za rozwiązanie zadań podanych na zajęciach!

Zadanie 1 [1,0 pkt]:

▲ Napisz funkcję do zmiany rozmiarów tablicy 2D alokowanej w sposób ciągły.

Wejście: położenie aktualnej tablicy, bieżące wymiary, nowe wymiary,

Wyjście: adres nowej alokacji.

Wzoruj się na logice funkcji opisanej na wykładzie do zmiany rozmiaru tablicy 1D (ogólny schemat postępowania jest bardzo podobny). Funkcja powinna przenieść tyle danych do nowej alokacji ile się da, przy czym powinno zostać zachowane przypisanie danych do wierszy. Przenosząc dane kieruj się następującymi wytycznymi:

1) Dane z jednego wiersza nie powinny być nigdy przeniesione do innego wiersza. Na przykład, redukcja kolumn - zmiana z 3x5 do 3x3 –wyglądać powinna tak:

```
 1  2  3  4  5      1  2  3
 6  7  8  9 10  => 6  7  8
11 12 13 14 15     11 12 13
```

2) Jeśli następuje redukcja liczby wierszy, to całe wiersze powinny być tracone.

3) Jeśli wymiary są rozszerzane, to nowe elementy powinny być ustawione na 0.

4) jeśli którykolwiek z nowych rozmiarów to 0, to tylko zwolnij pamięć.

Wskazówki: Zauważ, że tablica ta ma dwa wymiary, więc logikę postępowania przy zmianach rozmiarów trzeba do tego dostosować. Może warto samo kopiowanie danych wydzielić do osobnej funkcji. Jeśli tak zrobisz, to zastanów się gdzie lepiej jest określać zakresy kopiowania w funkcji kopiującej, czy zmieniającej rozmiar? Wykonywana tu jest alokacja ciągłą, więc całą tablicę trzeba realokować, nie da się tego zrobić wiersz po wierszu.

Ocenianie:

Poprawna logika funkcji:	0,6 pkt.
Poprawne przenoszenie danych:	0,2 pkt.
Ogólna jakość kodu:	0,2 pkt.

Zadanie 2 [1,0 pkt]:

▲ Napisz funkcję do zmiany rozmiarów tablicy 2D alokowanej fragmentarycznie.

Wejście: położenie aktualnej tablicy, bieżące wymiary, nowe wymiary,

Wyjście: adres nowej alokacji.

Reguły przenoszenia danych są identyczne jak w zadaniu 4. Cała różnica tylko tkwi w sposobie alokacji. Postaraj się w pełni wykorzystać swoją wiedzę o alokacji fragmentarycznej do redukcji operacji. Zauważ że:

1) Jeśli nie zmienia się liczba wierszy, to nie trzeba realokować tablicy wskaźników, wystarczy realokować każdy wiersz z osobna (użyj funkcji z wykładu).
2) Jeśli nie zmienia się liczba kolumn, to wierszy nie trzeba w ogóle realokować tylko poprzepinać je odpowiednio do nowej tablicy wskaźników.

Wskazówki: Cała trudność tego zadania polega na przemyśleniu jak ograniczyć liczbę realokacji. Ponieważ każdy wiersz jest osobną alokacją, to zmiana jego rozmiaru sprowadza się do zmiany rozmiaru tablicy 1D (to już jest napisane w przykładzie z wykładu). Realokacja tablicy wskaźników, to też w zasadzie realokacja tablicy 1D.,Trzeba jednak pamiętać, że nie jest to tablica wartości, tylko wskaźników. Kopiowane są adresy nie liczby.

Ocenianie:

Poprawna logika funkcji: 0,6 pkt.
Poprawne przenoszenie danych: 0,2 pkt.
Ogólna jakość kodu: 0,2 pkt.

Zadanie 3 [2,0 pkt]:

Uwaga: to zadanie będzie dalej rozwijane w ramach zadania ambitnego!

▲ Napisz funkcję zwalniającą alokację tablicy 2D, która może być zalokowana w sposób ciągły, lub fragmentaryczny.

Wejście: adres alokacji, wymiary tablicy (wariant łatwiejszy),

Wyjście: brak, ale funkcja powinna ustawić przesłany jej wskaźnik (adres alokacji) na nullptr, aby zaznaczyć, ze pamięć nie jest już przydzielona.

■ Funkcja powinna samodzielnie zidentyfikować z jakiego typu alokacją ma do czynienia i podjąć właściwą procedurę zwalania pamięci:

Wariant łatwiejszy (zalecany na początek):

Wymiary tablicy należy do funkcji przekazać. Nie wolno jednak w żaden sposób przekazywać typu alokacji (funkcja musi samodzielnie zidentyfikować typ).

Wskazówki: Aby rozpoznać typ alokacji musimy rozumieć jak jest ona zbudowana w pamięci. Pamiętaj, żeby nie wnioskować na bazie wartości w tablicy tylko na bazie adresów elementów, gdyż wartości mogą się zgodzić przypadkiem. Trzeba też wziąć pod uwagę, że tablica może mieć tylko jeden wiersz, wtedy oba typy zwalania są właściwe.

Wariant trudniejszy (zrealizuj dopiero gdy zadziała już łatwiejszy):

Funkcja powinna samodzielnie rozpoznać wymiary tablicy. Przyjmujemy jednak zawsze, że liczba elementów w każdym wierszu tablicy jest taka sama.

Wskazówki: Rozpoznawanie wymiarów, to znacznie trudniejsze zadanie i wymaga reinterpretacji danych zapisanych w znacznikach alokacji pamięci (kilka bajtów przed alokacją). Na wykładzie jest slajd poświęcony, temu gdzie szukać rozmiaru alokacji w pamięci poprzedzającej alokację. Zauważ ponadto, że informacja ta jest zapisywana w pojedynczych bajtach. Zadaj sobie też pytanie, czy zapisana jest liczba elementów, czy liczba bajtów?

Najpierw należy rozpoznać rozmiar alokacji tablicy wskaźników, potem można rozpoznać dopiero rozmiar alokacji tablicy/tablic danych. Znając rozmiar tablicy wskaźników, można odczytywać rozmiary tablic wierszy. Pamiętaj że alokacja ciągła ma zawsze jeden wiersz, nawet jeśli tablica wskaźników ma więcej elementów.

Bardzo ważne jest tu zauważenie, że sprawdzenie rozmiaru zerowego wiersza jeszcze nie wystarczy do wykrycia liczby kolumn. Dla alokacji ciągłej rozmiar alokacji, to przecież: liczba wierszy × liczba kolumn.

Opracowanie: dr inż. Łukasz Maliński i mgr inż. Błażej Nycz, Politechnika Śląska, 2021

Ocenianie:

```
Wariant łatwiejszy:
      Rozpoznanie typu alokacji:            0,8 pkt.
      Implementacja zwalniania:             0,2 pkt.
      Modyfikacja wskaźnika:                0,2 pkt.
Wariant trudniejszy:
      Automatyczne rozpoznanie wymiarów:    0,6 pkt.
Ogólna jakość kodu:                         0,2 pkt.
```

Uwaga: liczba punktów za wariant łatwiejszy jest 2 razy większa niż za wariant trudniejszy, gdyż jego realizacja jest bardziej dopasowana do oczekiwań stawianych studentowi 2 semestru. Punkty za wariant trudniejszy i tak dodają się do tych za wariant łatwiejszy. Wariant trudniejszy jest skierowany tylko do prawdziwych pasjonatów programowania, dla których punkty satysfakcja z pokonania problemu jest więcej warta niż punkty.