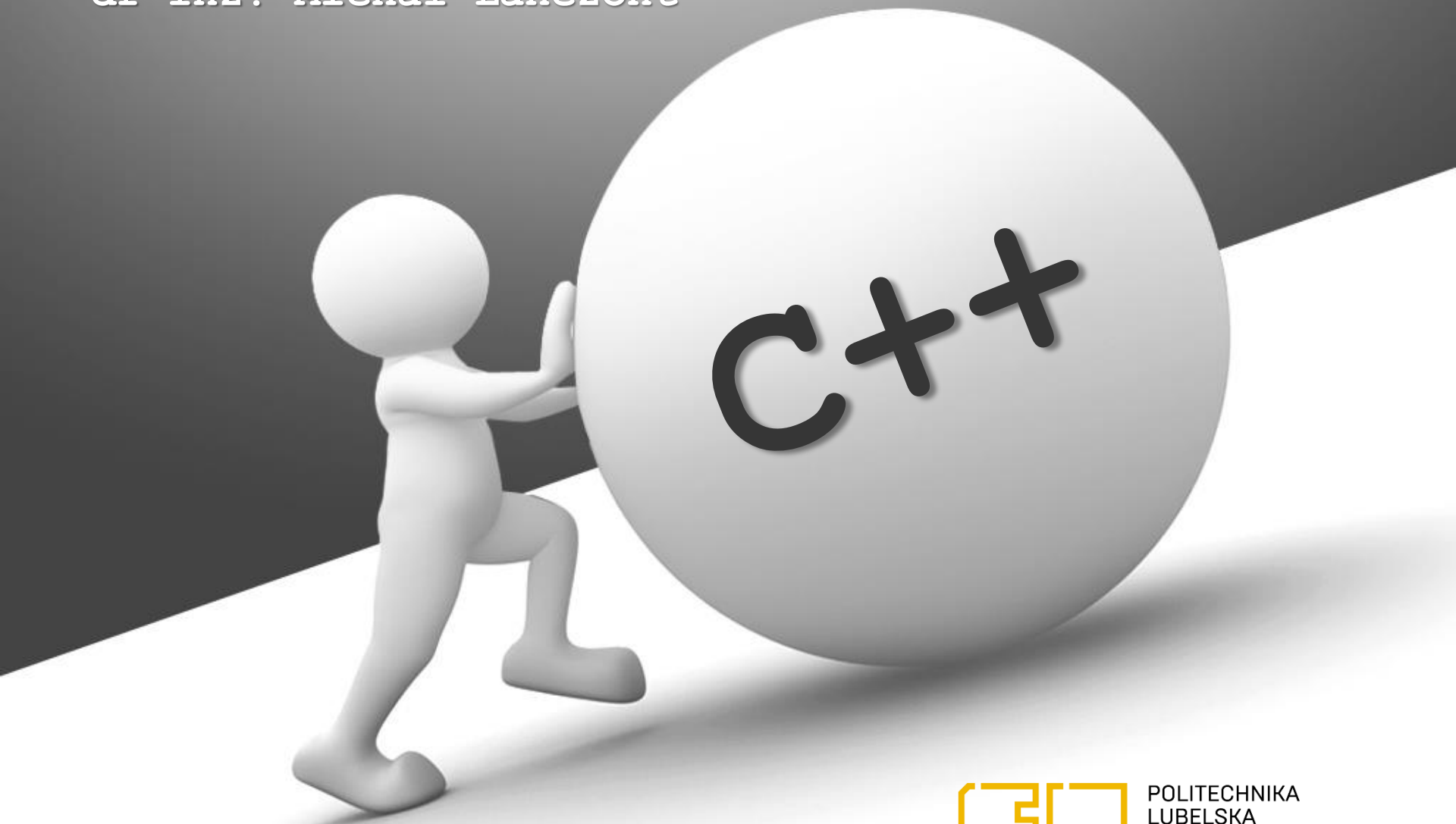
Informatyka II

dr inż. Michał Łanczont



POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Zakres wykładu

- Zapis i odczyt danych z pliku
  - Tekstowego
  - Binarnego
- Dynamiczna alokacja zmiennych
- Programowanie obiektowe
  - Hermetyzacja
  - Konstruktor
  - Destruktor
- Linked lists
- Double linked lists

Plik danych

- Standardowe wejście i wyjście – klawiatura i ekran
- Plik tekstowy i binarny – plikowe wejście i wyjście
- Funkcje obsługi portu typu file realizuje biblioteka <fstream>

Plik danych

- Proces przesyłania danych z i do pliku wymaga utworzenia obiektu klasy FILE

- Powiązanie obiektu z fizycznym plikiem na dysku
- Dla obiektu tworzy się dowiązanie symboliczne do pliku i określa tryb dostępu do niego (zapis lub odczyt)

10:38 Otwarcie dostępu do zapisu

```
ofstream plik;
plik.open(”nazwa.txt”);
```

lub

```
ofstream plik(”nazwa.txt”);
```

Procedura powoduje utworzenie pliku na dysku (jeśli nie istnieje) lub nadpisanie istniejącego. Jeżeli nie ma praw zapisu w wskazanym miejscu zgłaszany jest błąd.

```
plik.close(); // zamyka dostęp
                 do pliku
```

Otwarcie dostępu do odczytu

```
ifstream plik;
plik.open(”nazwa.txt”);
```

lub

```
ifstream plik(”nazwa.txt”);
```

Procedura otwiera dostęp do odczytu danych z pliku, o ile on istnieje. Jeżeli pliku nie ma na dysku zgłaszany jest błąd.

```
plik.close(); // zamyka dostęp do pliku
```

Przykład

- Program zapisuje dane w trybie tekstowym
- Zapis do pliku o nazwie podanej przez użytkownika
- Program zapisuje dane wprowadzone z klawiatury
- Poszczególne zadania realizowane są przez funkcje

Przykład

```
16  void zapis(string nazwa)
17  {
18      string bufor;
```

```
C:\WINDOWS\system32\cmd.exe
Podaj dane do zapis:Test procesu zapisu i odczytu danych z pliku tek
stowego. Procedura zapisu zostanie wykonana po wciśnięciu klawisza E
NTER na klawiaturze.
Zapis ukończony!
Odczyt danych z pliku:
Test procesu zapisu i odczytu danych z pliku tekstowego. Procedura z
apisu zostanie wykonana po wciśnięciu klawisza ENTER na klawiaturze.

Koniec danych.

Press any key to continue . . .
```

```
31      char c;
32      while(load.get(c))
33          cout << c;
34      cout << endl << "Koniec danych." << endl;
35      load.close();
36  }
```

Kontrola dostępu

- Kontrola poprawności otwarcia dostępu do pliku jest zagadnieniem które powinno być pod kontrolą
- Sprawdzenie stanu (statusu) otwarcia dostępu do pliku realizują metody:
  - `.fail();`
  - `.good();`
  - `.is_open();`
- Zalecane jest stosowanie metody `.is_open()` ze względu na szerszy zakres wykrywanych problemów

Kontrole dostępu

```
17  void zapis(string nazwa)
18  {
19      string bufor;
20      ofstream save;
21      save.open(nazwa);
22      cout << "save.fail(): " << save.fail() << endl;
23
24                                                    l;
25
26
27
28
29
30
31
32
33  }
```

```
C:\WINDOWS\system32\cmd.exe
Podaj nazwę pliku:Plik testowy
save.fail(): false
save.good(): true
save.is_open(): true
Podaj dane do zapis:Dane testowe do zapisu.
Zapis ukończony!
Odczyt danych z pliku:
load.fail(): false
load.good(): true
load.is_open(): true
Dane testowe do zapisu.
Koniec danych.

Press any key to continue . . .
```

Tryby dostępu

ios::app	Otwiera plik tylko do zapisu, ustawia wskaźnik na jego końcu i tam zapisuje dane.
ios::ate	Ustawianie wskaźnika na koniec pliku.
ios::binary	Otwarcie pliku w trybie binarnym, tzn. dane traktowane są jak binarne, a nie tekstowe.
ios::in	Otwarcie pliku w trybie do odczytu.
ios::out	Otwarcie pliku w trybie do zapisu.
ios::trunc	Podczas otwierania plik jest czyszczony

Przykład

```
17  void zapis(string nazwa)
18  {
19      string bufor;
20      ofstream save;
21      save.open(nazwa, ios_base::app);
22      cout << "save.fail(): " << save.fail() << endl;
23      cout << "save.good(): " << save.good() << endl;
24      cout << "save.is_open(): " << save.is_open() << endl;
25      if(save.is_open()){
26          cout << "Podaj dane do zapis:";
27          getline(cin,bufor);
28          save << bufor << endl;;
29          cout << "Zapis ukończony!" << endl;
30      }else
31          cout << "Zapis danych niemożliwy!" << endl;
32      save.close();
33  }
```

Wybór trybu dostępu

- Utworzenie obiektu FILE klasy ifstream lub ofstream powoduje że dostęp realizowany jest w trybie odczytu lub zapisu (nadpisywania)
- Klasa fstream dziedziczy po ifstream i ofstream
- Obiekt klasy fstream pozwala na określenie sposobu dostępu do pliku, możliwość zmiany trybu dostępu
- Metody klasy ios_base

Tryb dostępu

```
fstream plik;
plik.open(nazwa, ios_base::in);
```

- **Otwarcie w trybie odczytu**

```
plik.open(nazwa, ios_base::out);
```

- **Otwarcie w trybie zapisu**
- **Korzystając z łącznika bitowego "|" można łączyć tryby**

```
plik.open(nazwa, ios_base::in |
                 ios_base::out);
```

Tryby dostępu

- W przypadku ustawienia jednoczesnego trybu dostępu do zapisu i odczytu należy analizować pozycję w pliku.
- Po zapisaniu danych ”kursor” umiejscowiony jest na końcu pliku.
- Próba odczytania danych nie zakończy się powodzeniem
- Konieczne jest przeniesienie kursora na początek pliku

Obsługa strumienia

- Zapis i odczyt danych można w trybie tekstowym realizować analogicznie jak dla standardowych strumieni wejścia i wyjścia z wykorzystaniem operatorów kierunkowania strumienia

```
plik << dane;
plik >> dane;
```

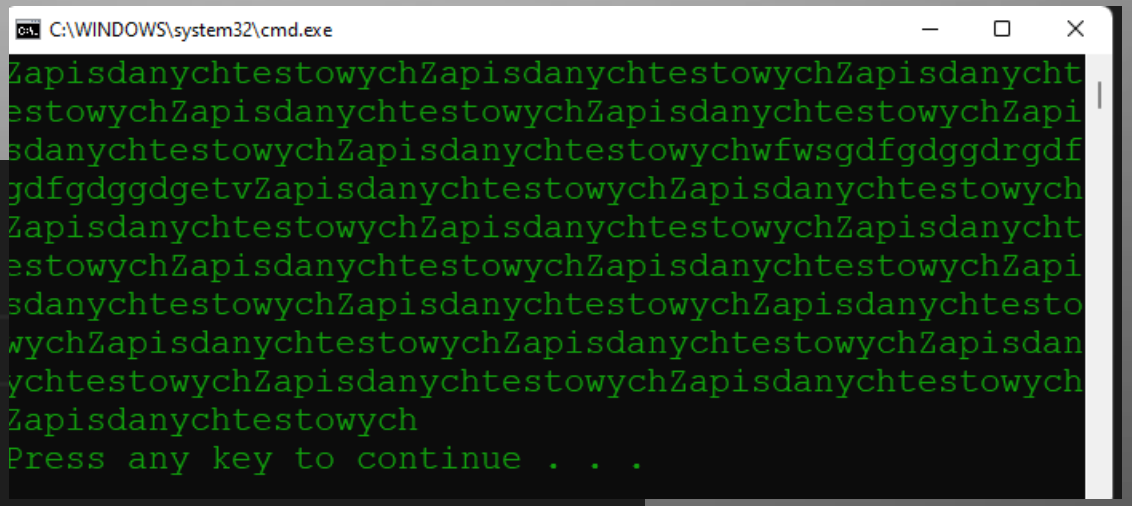
- Pomijane znaki spacji, tabulacji i przejścia do nowego wiersza.

Przykład

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    fstream plik;
    plik.open("nowy.txt", ios_base::in);
    string bufor;
    while(plik >> bufor)
        cout << bufor;
    plik.close();
}
```



Pozycja w pliku

- Określanie pozycji w pliku

```
.tellp(); //zapis
.tellg(); //odczyt
```

- Metody działają praktycznie identycznie, zwracają numer pozycji którą aktualnie będziemy zapisywać lub odczytywać.
- Przy odczycie metoda `.tellg()` gdy dojdziemy do końca pliku zwróci wartość `'-1'`, a `.tellp()` numer pozycji

Przykład

C:\WINDOWS\system32\cmd.exe

```
"Tekst nowy. Koniec."
 01234567890123456789

Zapis:
        Pozycja: 0
        Pozycja: 11 Zapisano: "Tekst nowy."
        Pozycja: 19 Zapisano: " Koniec."
Odczyt:
        Pozycja: 0
        Pozycja: 5
        Pozycja: 11
        Pozycja: -1
Odczytano: Tekstnowy.Koniec.

Press any key to continue . . .
```

```
32      cout << "Odczytano: " << tekst << endl;
33  }
```

Ustawianie pozycji

- Przemieszczanie kursora po pliku umożliwiają funkcje:

```
.seekp(liczba,pozycja); // zapis
.seekg(liczba,pozycja); // odczyt
```

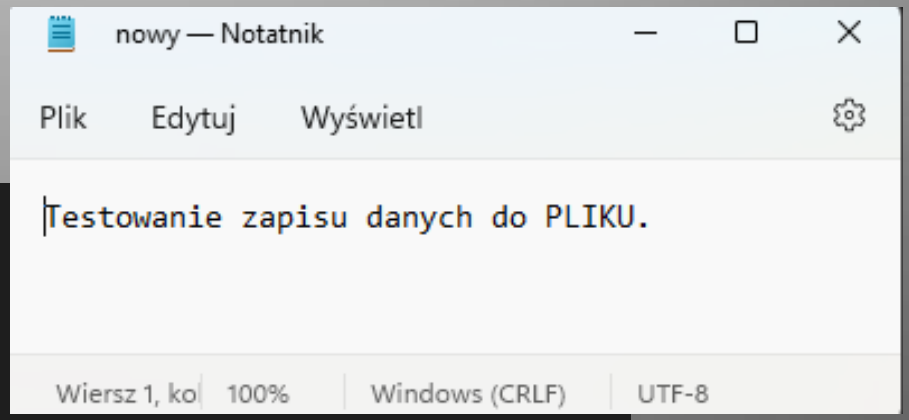
- Liczba określa przesunięcie względem podanej pozycji (+/-).
- Przy zapisie dla trybu **ios_base::out**
- Pozycję określa się za pomocą trzech metod:
  - **ios_base::beg** - początek pliku (*domyślna*)
  - **ios_base::end** - koniec pliku
  - **ios_base::cur** - bieżąca pozycja

10:47

Przykład

```cpp
#include <iostream>
#include <fstream>

using namespace std;
void zapis();
void odczyt();
int main()
{
    fstream plik;
    string tekst1 = "Testowanie zapisu danych do pliku.";
    string tekst2 = "PLIKU.";
    plik.open("nowy.txt", ios_base::out);
    plik << tekst1;
    plik.seekp(-6,ios_base::cur);
    plik << tekst2;
    plik.close();
}
```



Metody strumienia fstream

- **Za pomocą metod stosowanych do obsługi strumienia standardowego**
- **Pliki tekstowe**
  - `.put()` `//typ char`
  - `.get()` `//typ char`
  - `.getline()` `//tablica char[]`
  - `getline()` `//string`
  - `.write()` `//typ char lub char[]`
  - `.read()` `//typ char lub char[]`
- **Pliki binarne**
  - `.write()`
  - `.read()`

Identyfikacja końca pliku

- Przy odczycie danych z pliku istotne jest rozpoznanie kiedy przy odczycie dotrzemy do końca pliku
- `getline(plik,string)` zwraca wartość `false` gdy nie może odczytać danych z pliku tekstowego
- `.eof()` zwraca wartość `true` gdy dotrzemy przy odczytywaniu do końca pliku
- `.clear()` – czyści flagi błędów

Plik tekstowy

C:\WINDOWS\system32\cmd.exe

```
.getline()
Testowy zestaw danych do zapisu.
Testowy zestaw danych do zapisu.

getline()
Testowy zestaw danych do zapisu.
Testowy zestaw danych do zapisu.
.get()
Testowy zestaw danych do zapisu.
Testowy zestaw danych do zapisu.
.read()
Testowy zestaw danych do zapisu.
Testowy zestaw danych do zapisu.

Press any key to continue . . .
```

```
su.
su.

UTF-8

base::in);
su.\n";
```

Konwersja na liczbę

- Dane odczytywane z pliku metodą `getline()` są typu string (łańcuchem znakowym)
- Odczytywanie danych liczbowych wymaga ich konwersji do formatu `int` lub `float`
- Funkcje `atoi()` i `atof()` konwertują dane string do formatu liczbowego

```
atoi(tekst.c_str());
atof(tekst.c_str());
```

10:47

Plik binarny

C:\WINDOWS\system32\cmd.exe

```
 0  1  2  3  4  5  6  7  8  9
 1  2  3  4  5  6  7  8  9 10
 2  3  4  5  6  7  8  9 10 11
 3  4  5  6  7  8  9 10 11 12
 4  5  6  7  8  9 10 11 12 13
 5  6  7  8  9 10 11 12 13 14
 6  7  8  9 10 11 12 13 14 15
 7  8  9 10 11 12 13 14 15 16
 8  9 10 11 12 13 14 15 16 17
 9 10 11 12 13 14 15 16 17 18
-----------ZAPISANO----------
----Otwarto plik binarny.----
 0  1  2  3  4  5  6  7  8  9
 1  2  3  4  5  6  7  8  9 10
 2  3  4  5  6  7  8  9 10 11
 3  4  5  6  7  8  9 10 11 12
 4  5  6  7  8  9 10 11 12 13
 5  6  7  8  9 10 11 12 13 14
 6  7  8  9 10 11 12 13 14 15
 7  8  9 10 11 12 13 14 15 16
 8  9 10 11 12 13 14 15 16 17
 9 10 11 12 13 14 15 16 17 18

Press any key to continue . . .
```

testowy — Notatnik

Plik Edytuj Wyświetl

Wiersz 1, kolumna 1 220% Unix (LF) UTF-8

10:47

Plik binarny

```
12  void odczyt(int n)
```

C:\WINDOWS\system32\cmd.exe

```
----------ZAPISANO----------
Zapisano 54 znaków
```

testowy — Notatnik

Plik Edytuj Wyświetl

Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.Testowy zapis danych do pliku binarnego. Dane tekstowe.

Wiersz 1, kolumna 1 | 100% | Windows (CRLF) | UTF-8

```
30      cout << "----------ZAPISANO----------" << endl;
31      plik.close();
32      return dane.length();
33  }
```

Dynamiczna alokacja

- Dynamiczna alokacja umożliwia żądanie w czasie działania programu oczekiwanej ilości pamięci dla zmiennych.
- Program z standardowo zadeklarowanymi zmiennymi ma przydzielony stały obszar pamięci
- Dynamiczne alokowanie pamięci umożliwia zajęcie przez zmienne miejsc poza tym obszarem

Dynamiczna alokacja

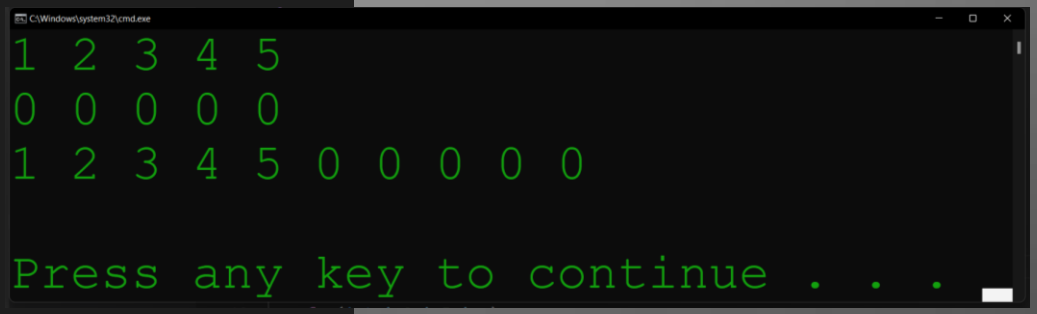
- W języku C++ można korzystać z narzędzi znanych z język C

```
T = (typ*)calloc(liczba, rozmiar);
T = (typ*)malloc(liczba * rozmiar);
N =(typ*)realloc(T,liczba*rozmiar);
free(T);
```

- Dynamiczna alokacja jest stosowana w odniesieniu do danych tablicowych
- Metoda może być stosowana do prostych typów danych (int, float, char)

Przykład

```
1  #include <iostream>
2  using namespace std;
3  void show(int *X, int n)
4  {
5      for(int i=0;i<n;i++)
6          cout << X[i] << " ";
7      cout << endl;
8  }
9  int main()
10 {
11     int *B, *C;
12     int *A = (int*) malloc(5*sizeof(*A));
13     for(int i=0;i<5;i++)
14         A[i] = i+1;
15     show(A,5);
16     B = (int*) calloc(5,sizeof(int));
17     show(B,5);
18     C = (int*) realloc(A,10*sizeof(*C));
19     show(C,10);
20     free(A);free(B);free(C);
21 }
```



Dynamiczna alokacja

- W języku C++ dostępna jest dedykowana konstrukcja alokująca za pomocą operatora new

```
int *zmienna = new int;
```

- W przypadku braku wolnej pamięci zostanie zgłoszony wyjątek
- W przypadku współczesnych komputerów sytuacja taka jest mało prawdopodobna
- Gdy zmienna przestaje być potrzeba należy zwolnić pamięć

```
delete zmienna;
```

Dynamiczna alokacja

- Za pomocą operatora new można alokować zmienne, tablice jedno i wielowymiarowe
- W przypadku tablicy jednowymiarowej należy określić liczbę elementów

```
int *T = new int[rozmiar];
```

- Kasowanie tablicy dynamiczne realizuje funkcja delete

```
delete [] T;
```

Dynamiczna alokacja

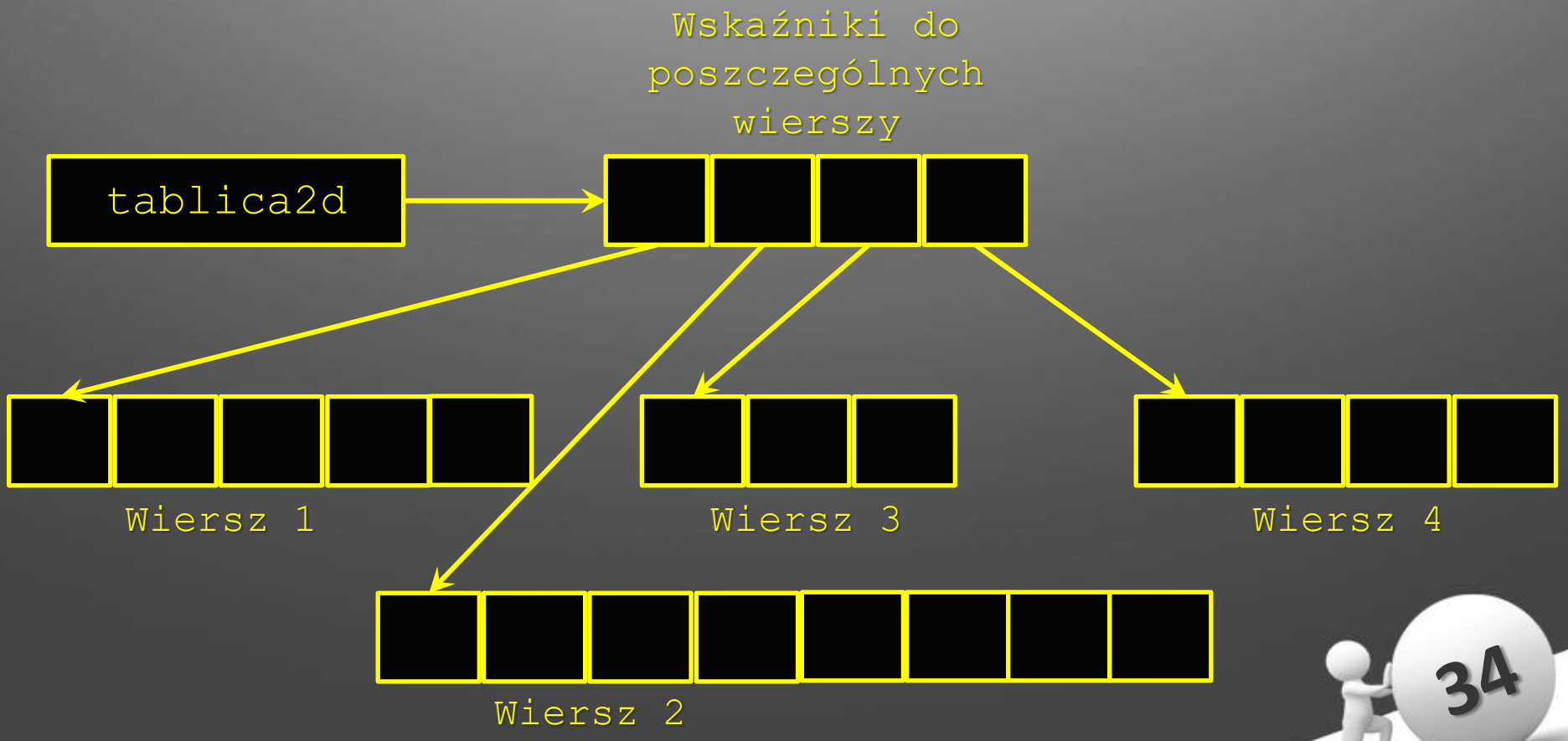
- Tablice dwuwymiarowe przy alokacji dynamicznej wymagają zastosowanie konstrukcji wskaźnika do wskaźnika

```
int **tablica2d;
```

- Procedura ta wynika ze sposobu organizacji przestrzeni w pamięci dla tablicy dwuwymiarowej
- W pierwszym etapie alokuje się wiersze macierzy
- W drugim etapie komórki każdego z wierszy

Dynamiczna alokacja

- Graficznie cały proces można przedstawić za pomocą diagramu:



Dynamiczna alokacja

- Inicjacja wskaźnika do wskaźnika, początek deklaracji tablicy dwuwymiarowej

```
int **tablica2d;
```

- Alokacja tablicy wskaźników, określenie liczby elementów w wierszu

```
tablica2d = new int*[n];
```

- Każdy z wskaźników tablicy jest zalążkiem kolejnej tablicy

Dynamiczna alokacja

- Dla każdego wskaźnika utworzonego w poprzedni kroku trzeba alokować nową tablicę

- Przydzielić obszar w pamięci dla każdego wiersza tablicy 2d

```
for (int i=0; i<n;i++){
  tablica2d[i] = new int[m];
}
```

Dynamiczna alokacja

- Zwalnianie pamięci przez tablicę dwuwymiarową jest procesem przeprowadzanym w kolejności odwrotnej do procedury alokowania.
- Najpierw należy zwolnić pamięć każdego wiersza
- Następnie można zwolnić tablicę alokacyjną tablicy wierszy

Przykład

```
C:\Windows\system32\cmd.exe
Zmienna A=10
Tablica B:1 2 3 4 5
Tablica C:
0
1 2
2 3 4
3 4 5 6
4 5 6 7 8

Press any key to continue . . .
```

```
42      delete [] C;
43  }
13              cout << X[i][j] << " ";
14          cout << endl;
15      }
16  }
```

Zmiana rozmiaru

- Nie ma dedykowanej funkcji modyfikującej rozmiar tablicy dynamicznej (new)
- Konieczne jest napisanie dedykowanej funkcji
  - Tymczasowa tablica o nowym rozmiarze
  - Kopiowanie danych do tablicy tymczasowej
  - delete i new dla nowego rozmiaru
  - Kopiowanie z tymczasowej do tablicy o nowym rozmiarze
  - Funkcja zwracająca wskaźnik lub wskaźnik globalny

Przykład

- **Kopiowanie danych pomiędzy tablicami**

```
memcpy(*new,*old,size);
memmove(*new,*old,size);
for(){}  //sctruct union class
```

- **Ustawianie danych w tablicy <cstring>**

```
memset(*tab,val,size);
```

- **Funkcja zwracająca wskaźnik**

```
typ* nazwa(typ,…)
{
      return *tab;
}
```

Przykład 1

```
C:\Windows\system32\cmd.exe
Rozmiar tablicy:10
2 2 6 7 7 4 5 5 4 9
Rozmiar tablicy:25
2 2 6 7 7 4 5 5 4 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 6 7 7 4 5 5 4 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Rozmiar tablicy:25
6 6 7 7 4 0 7 1 2 9 9 1 8 6 2 2 3 9 5 3 5 6 2 9 3
Rozmiar tablicy:5
6 6 7 7 4
6 6 7 7 4

Press any key to continue . . .
```

```
48      delete [] newTab;
49      return tab;
50  }
```

bR)

Przykład 2

C:\Windows\system32\cmd.exe

```
Punkt(0)[0,0]
Punkt(1)[1,2]
Punkt(2)[2,4]
Punkt(3)[3,6]
Punkt(4)[4,8]
Punkt(5)[5,10]
Punkt(6)[6,12]
Punkt(7)[7,14]
Punkt(8)[8,16]
Punkt(9)[9,18]
Punkt(10)[10,20]
Punkt(11)[11,22]
Punkt(12)[12,24]
Punkt(13)[13,26]
Punkt(14)[14,28]
Punkt(0)[0,0]
Punkt(1)[1,2]
Punkt(2)[2,4]
Punkt(3)[3,6]
Punkt(4)[4,8]
Punkt(5)[5,10]
Punkt(6)[6,12]

Press any key to continue . . .
```

```
& size,int newSize)

punkt[newSize];

++)                    endl;
i];

;
++)
i];
```

Klasy

Konstruując klasę należy trzymać się kilku zasad:

- Właściwej wykorzystywać deklarację private i public
- Dokładnie trzeba przemyśleć interfejs klasy
- Łatwo można przenieść metody czy pól danych z implementacji prywatnej do publicznej, w drugą stronę jest to często bardzo skomplikowane
- Zaleca się aby pola danych zawsze definiować jako prywatne

Klasy

- Metody domyślne definiować jako prywatne i przenosić do interfejsu publicznego wtedy gdy jest to konieczne
- W sytuacji gdy jest konieczne nadanie dostępu do pola danych (strefa prywatna) najlepiej zrealizować to poprzez metodę
  - *get()* - odczytującą pole danych
  - *set()* - zapisującą do pola danych

Klasy

- Metoda ukrywania pól danych jako prywatnych (klasy) to **HERMETYZACJA**
- Polega ona na ukrywaniu implementacji (ich budowy i sposobów wykorzystania)
- Użytkownicy uzyskują do nich pośredni dostęp dzięki interfejsowi klasy i przeznaczonym do tego metodom

Klasy

- Tworząc nową klasę i deklarując w niej pola danych (zmienne) nie należy tych zmiennych inicjować, nie można przypisywać do niej wartości domyślnej
- Inicjację pól klasy należy realizować korzystając z **KONSTRUKTORA**
- Konstruktor jest metodą która jest uruchamiana w wyłącznie momencie utworzenie nowego obiektu na klasie

Konstruktor

- Przy braku konstruktora kompilator tworzy konstruktor domyślny
- Konstruktor taki jest pusty i nic nie wykonuje
- Konstruktor może ustawiać wartości domyślne cech lub pobierać je poprzez zapytania lub parametry wywołania
- W klasie można zdefiniować kilka konstruktorów

Konstruktor

- Wartości domyślne podaje się w przy deklaracji konstruktora
- Tworząc nowy obiekt z konstruktorem zawierającym wartości domyślne nie trzeba podawać wartości startowych
- Można podać wszystkie lub część z nich
- Parametry inne niż domyślne muszą być podawane w kolejności ich zgłoszenia od lewej

Konstruktor

- Głównym zadaniem konstruktora jest inicjowanie zmiennych (obiektów swojej klasy)
- Często w procesie tym w przypadku inicjacji obiektu poprzez metodę konstruktora wraz z parametrami wejściowymi istnieje konieczność sprawdzenia poprawności kompatybilności tych danych
- Warunek ten określa się mianem NIEZMIENNIKA KLASY

Konstruktor

- Jest metodą o tej samej nazwie co klasa
- Deklaracja konstruktora może zawierać listę parametrów
- Nie ma określonego typu zwracanego
- Żadna inna metoda zdefiniowana w klasie nie może mieć takiej samej nazwy jak klasa

Konstruktor

- **Utworzenie obiektu z konstruktorem bez parametrów lub z parametrami domyslnymi**

```
nazwaKlasy nazwaObiektu;
nazwaKlasy *nazwaObiektu = new nazwaKlasy;
```

- **Utworzenie obiektu z konstruktorem posiadającym parametr wejściowym**

```
nazwaKlasy nazwaObiektu(par);
nazwaKlasy *nazwaObiektu = new nazwaKlasy(par);
```

Niezmiennik klasy

- Definiuje się go jako warunek testujący parametr(y) wejściowy
- Jeżeli warunek nie zostanie spełniony niezmiennik:
  - Zgłasza wyjątek
  - Obiekt nie jest tworzony
  - Zwalniane są wszelkie zasoby jakie przy tworzeniu obiektu zostały utworzone

Destruktor

- DESTRUKTOR jest metodą o nazwie takiej samej jak klasa, ale poprzedzona znakiem tyldy '~'
- Metoda ta **nie może mieć** żadnych parametrów wejściowych
- DESTRUKTOR ma za zadanie „posprzątanie" w chwili usuwania obiektu (wyjście z funkcji, koniec pracy programu)

Przykład 1

```
23      void show(){
24          cout << nazwa
25      }
5   private:
6       int x,y;
7       string nazwa;
8   public:
9       punkt() ···
18      punkt(string nazwa
23      void show(){ ···
26  };
27
28  int main()
29  {
30      punkt A;
31      A.show();
32      punkt B("Alfa",23,54);
33      B.show();
34  }
```

```
21
22
15          cout << nazwa <<  .y= ;
16          cin >> y;
17      }
```

```
C:\Windows\system32\cmd.exe
Nazwa punktu:Beta
Beta.x=11
Beta.y=23
Beta[11,23]
Alfa[23,54]

Press any key to continue . . .
```

Przykład 1

```
 9      void set()
10      {
11          cout << "Nazwa punktu:";
12          getline(cin,nazwa);
13          cout << nazwa << ".x=";
14          cin >> x;
15          cout << nazwa << ".y=";
16          cin >> y;
17      }
18      punkt(string nazwa="Alfa", int x=0, int y=0){
19          this->nazwa = nazwa;
20          this->x = x;
21          this->y = y;
22      }
```

Przykład 1

kod9.cpp | punkt.hpp | punkt.cpp

punkt.cpp > ...

```
1   #include "punkt.hpp"
2
3   void punkt::set() ···
12  punkt::punkt(string nazwa, int x, int y)
13  {
14      this->nazwa = nazwa;
15      this->x = x;
16      this->y = y;
17  }
18  void punkt::show() ···
13      void show();
14  };
15  #endif
```

Przykład 2

```
C:\WINDOWS\system32\cmd.exe
Konstruktor 1 obiektu 0x7ed5fffd00
Nie podano rozmiaru tablicy!
Podaj rozmiar tablicy:4
Podaj wartość [0] elementu:1
Podaj wartość [1] elementu:2
Podaj wartość [2] elementu:3
Podaj wartość [3] elementu:4
Dane tablicy: 1 2 3 4
Konstruktor 2 obiektu 0x21977a86c80
Podaj wartość [0] elementu:3
Podaj wartość [1] elementu:5
Podaj wartość [2] elementu:6
Podaj wartość [3] elementu:4
Podaj wartość [4] elementu:2
Dane tablicy: 3 5 6 4 2
Destruktor obiektu 0x21977a86c80
Obiekt jest kasowany!
Konstruktor 1 obiektu 0x21977a86c80
Nie podano rozmiaru tablicy!
Podaj rozmiar tablicy:3
Podaj wartość [0] elementu:1
Podaj wartość [1] elementu:2
Podaj wartość [2] elementu:3
Dane tablicy: 1 2 3
Destruktor obiektu 0x7ed5fffd00
Obiekt jest kasowany!

Press any key to continue . . .
```

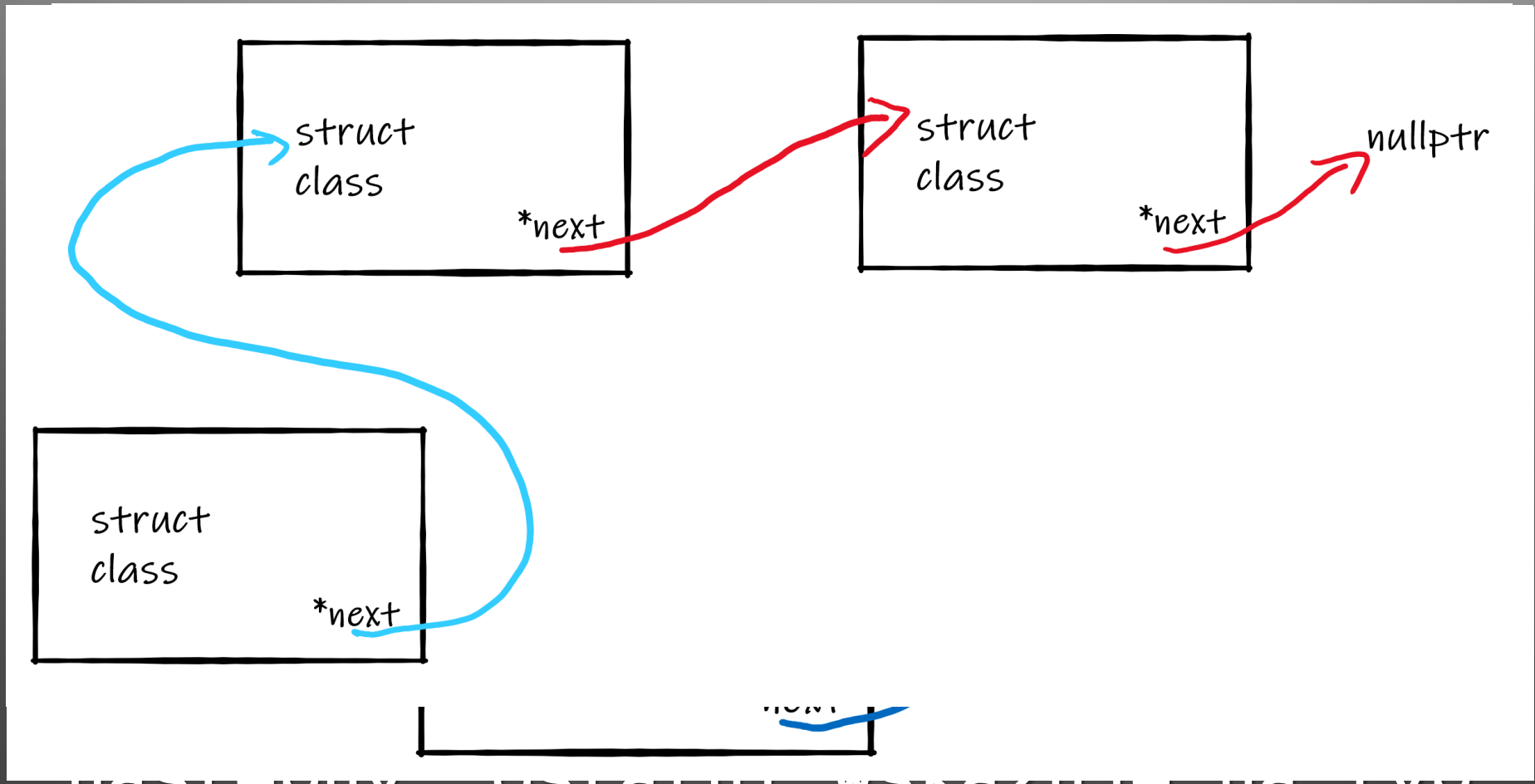
```
[" << i << "] elementu:";
';
```

Dynamiczna alokacja a konstruktor

- Dynamicznie alokując obiekt lub tablicę obiektów z konstruktorem wymagającym podania listy parametrów związane jest z przekazaniem tych parametrów.

- Parametry w nawiasie ()
- Tablica parametrów {}

Linked list



następny, ostatni wskazuje na typ nullptr

Linked list

- W przypadku klasy pole wskazujące na element następny jest typu public
- Można zaprogramować odpowiednie metody gdy pole wskazujące jest prywatne.
- W programie konieczne jest zaprogramowanie dedykowanych funkcji wstawiających lub kasujących obiekty listy

Przykład

```
32 › int main() ···
50 › void dodajP(punkt**pier) ···
57 › void dodajO(punkt**pier) ···
71 › void dodaj(punkt*P) ···
78 › void dodaj(punkt *P,string nazwa) ···
89   void drukuj(punkt *n)
90   {
91       cout << "-----------------------" << endl;
92       while(n!=nullptr){
93           n->show();
94           n=n->kolejny;
95       }
96       cout << "-----------------------" << endl;
97   }
93 › void drukuj(punkt *n) ···
```

Przykład

C:\Windows\system32\cmd.exe

```
Podaj nazwę:minu
minu.x=-5
minu.y=-10
----------------------
Alfa[-10,-20]
minu[-5,-10]
pierwszy[0,0]
ostatni[10,10]
Beta[20,30]
----------------------
Podaj nazwę punktu po którym dodać nowy punkt:minus
Nie ma takiego punktu!
----------------------
Alfa[-10,-20]
minu[-5,-10]
pierwszy[0,0]
ostatni[10,10]
Beta[20,30]
----------------------
Podaj nazwę punktu po którym dodać nowy punkt:pierwszy
Podaj nazwę:plus
plus.x=5
plus.y=5
----------------------
Alfa[-10,-20]
minu[-5,-10]
pierwszy[0,0]
plus[5,5]
ostatni[10,10]
Beta[20,30]
----------------------

Press any key to continue . . .
```

Double linked list

- Konstrukcja analogiczna do **linked list**
- Obiekt posiada dodatkowo wskaźnik na wcześniejszy element
- Pierwszy element wskazuje na drugi (do przodu) i `nullptr` (do tyłu)