Paradygmaty Programowania

mgr inż. Marcin Bączyk (prowadzący) dr inż. Adam Wojtasik(autor prezentacji)

Wykład 1

22 lutego 2021

Treść dzisiejszego wykładu

- omówienie regulaminu przedmiotu
- struktury danych
  - struktury tablicowe
  - struktury łączowe
  - struktury tablicowo-łączowe
- typy danych
  - typy proste
  - typy złożone
  - abstrakcyjne typy danych
  - kolejki uogólnione

Regulamin przedmiotu

Zajęcia

- wykład (10 spotkań po 2 x 45 minut) - MS Teams, kanał wykład
- laboratoria (5 spotkań po 2 x 45 + 15 minut) - MS Teams, odpowiednia grupa laboratoryjna
- *egzamin* (120 minut) - ??

Obecność

Obecność na wykładach nie jest obowiązkowa. Obecność na zajęciach laboratoryjnych jest obowiązkowa. Nieusprawiedliwiona nieobecność na zajęciach laboratoryjnych skutkuje otrzymaniem zera punktów za te zajęcia. Nieobecność na zajęciach należy usprawiedliwić w przeciągu 7 dni od ustania przyczyny nieobecności na zajęciach.

Regulamin przedmiotu

Metody weryfikacji osiągnięcia efektów uczenia się

- dwa krótkie (15 minut) testy odbywające się w trakcie zajęć wykładowych po 10 punktów każdy
- pięć zajęć laboratoryjnych po 6 punktów każde
  - 1,5 punkta - "wejściówka"
  - 4,5 punkta - wykonanie zadania i oddanie protokołu
  - aby całe ćwiczenie mogło być zaliczone należy zdobyć co najmniej 0,5 punkta z "wejściówki"
- egzamin końcowy za 50 punktów

Poprawa, odrabianie zajęć laboratoryjnych

Poprawa testów odbywających się w czasie wykładu będzie możliwa w innym terminie. Zajęcia laboratoryjne nie mogą być poprawiane. Odrabianie usprawiedliwionej nieobecności na zajęciach laboratoryjnych należy uzgadniać z prowadzącym daną grupę. Wykonywanie pojedynczych zajęć z inną grupą należy uzgodnić z dwoma prowadzącymi.

Regulamin przedmiotu

Forma "wejściówek" oraz protokołu zależy od konkretnego prowadzącego zajęcia. Testy wykładowe będą wykorzystywały dostępne narzędzia na platformach dopuszczonych przez PW. Forma egzaminu zostanie ustalona w trakcie semestru na podstawie sytuacji epidemiologicznej w kraju i rekomendacji uczelni w tej sprawie.

W trakcie wszystkich sprawdzianów i egzaminów dozwolone jest wykorzystywanie wszystkich dostępnych materiałów (książki, Internet, własne notatki, itp.) nienaruszających samodzielności. Niedopuszczalne jest komunikowanie się z kimkolwiek w trakcie sprawdzianu.

Regulamin przedmiotu

Zaliczenie przedmiotu

Aby zaliczyć przedmiot należy zdobyć łącznie co najmniej 51 punktów z obu testów wykładowych, laboratoriów i egzaminu oraz zaliczyć co najmniej **trzech** z pięciu zajęć laboratoryjnych. Zaliczenie zajęć laboratoryjnych oznacza zdobycie co najmniej połowy możliwych do uzyskania punktów.

W trakcie wszystkich sprawdzianów i egzaminów dozwolone jest wykorzystywanie wszystkich dostępnych materiałów (książki, Internet, własne notatki, itp.) nienaruszających zasady samodzielności. Niedopuszczalne jest komunikowanie się z kimkolwiek w trakcie sprawdzianów.

Regulamin przedmiotu

Skala ocen

- 0-50,(9) - 2
- 51 - 60,(9) - 3
- 61 - 70,(9) - 3,5
- 71 - 80,(9) - 4
- 81 - 90,(9) - 4,5
- 91 - 100 - 5

Zdobycie co najmniej 45 punktów ze wszystkich laboratoriów oraz testów wykładowych upoważnia do zwolnienia z pisania egzaminu i otrzymania oceny 5.

Testy, raporty z wykonanych w trakcie zajęć laboratoryjnych ćwiczeń będą ocenione w czasie 14 dni. Egzamin zostanie oceniony w ciągu kilku dni, przed terminem poprawkowym.

Cel i zakres merytoryczny przedmiotu PAPRO

Celem przedmiotu jest:

- zaznajomienie studentów z pojęciem paradygmatu w kontekście nauk informatycznych;
- przedstawienie obecnie najpopularniejszych paradygmatów programowania wraz z ich zaletami, wadami oraz obszarami stosowania.

paradygmat (SJP)

1. «przyjęty sposób widzenia rzeczywistości w danej dziedzinie, doktrynie itp.»
2. «zespół form fleksyjnych danego wyrazu lub końcówek właściwych danej grupie wyrazów»

programowanie (SJP)

1. «tworzenie (pisanie) programów komputerowych»

Literatura

- Robert C. Martin, *Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów*
- Witold Malina, Piotr Mironowicz, *Wykłady z informatyki. Programowanie strukturalne. Trendy programowania.*
- Steven F. Lott, *Python. Programowanie funkcyjne*
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku*
- Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Erich Gamma, *Refaktoryzacja. Ulepszanie struktury istniejącego kodu*

Informacje inne

Dane teleadresowe

- pokój nr 449 w Gmachu Elektroniki (4 piętro, skrzydło C)
- *m.k.baczyk@elka.pw.edu.pl* lub *marcin.baczyk@pw.edu.pl*
- Konto MS Teams powiązane z adresem e-mail *marcin.baczyk@pw.edu.pl*
- strona przedmiotu
  http://staff.elka.pw.edu.pl/ mbaczyk1/PAPRO_2021L/index.html

Konsultacje

- kanał ogólny zespołu 2021L_103A-ELxxx-ISP-PAPRO_WYK_1
- prywatny / grupowy czat lub rozmowa głosowa MS Teams
- termin dowolny po wcześniejszym umówieniu / preferowane godziny poranne i popołudniowe

Struktura danych

Mechanizm organizacji informacji (danych), który implikuje sposób dostępu do tych informacji i przeprowadzania na nich operacji

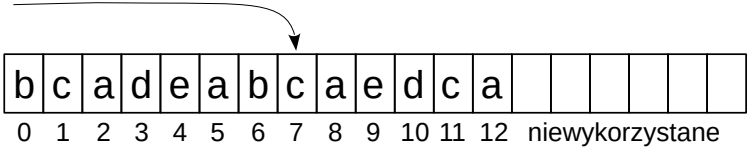
---

Jedyne dwie elementarne zasady budowania struktury danych

- Struktura tablicowa – elementy umieszczone są kolejno obok siebie i mają kolejne numery (indeksy).
- Struktura z łączami – elementy mogą być rozrzucone w pamięci, ale każdy z nich wyposażony jest w dodatkową informację mówiącą o tym, gdzie leży następny element.

Wszystkie stosowane struktury danych opierają się na stosowaniu jednej lub obu tych zasad

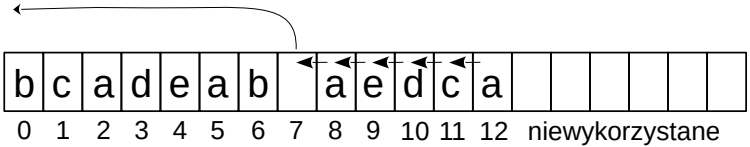
Tablicowa struktura danych



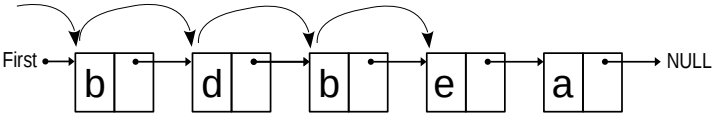
- Łatwy i szybki dostęp (równoległy),

Ale:

- Często część zarezerwowanej pamięci jest niewykorzystana.
- Budowa jest sztywna, trudna do modyfikacji czy reorganizacji.



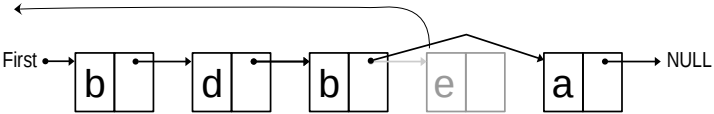
Łączowa struktura danych



- Niezbyt szybki dostęp (sekwencyjny, szeregowy).
- Każdy element musi zajmować dodatkową pamięć na łącza.

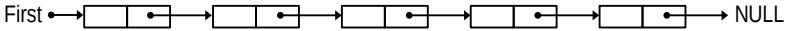
Ale:

- Nie ma „pustej” pamięci.
- Budowa jest elastyczna, łatwa do do reorganizacji czy modyfikacji.

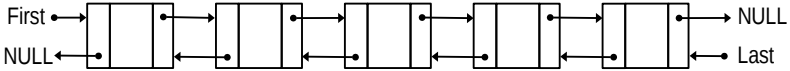


Łączowe struktury danych – przykłady

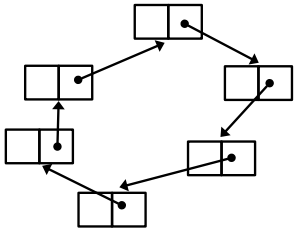
Lista:



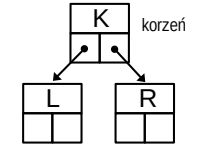
Lista dwukierunkowa:



Lista cykliczna:



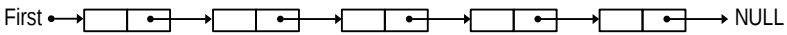
Drzewo binarne:



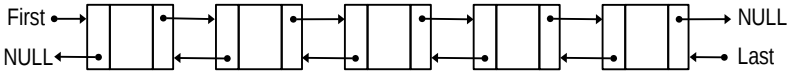
Potomkowie leżący na danym poziomie drzewa są korzeniami dla następnego poziomu

Łączowe struktury danych – przykłady

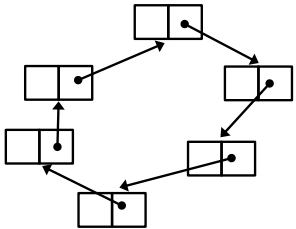
Lista:



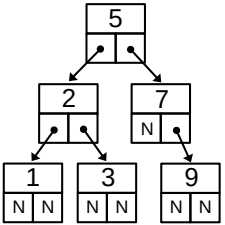
Lista dwukierunkowa:



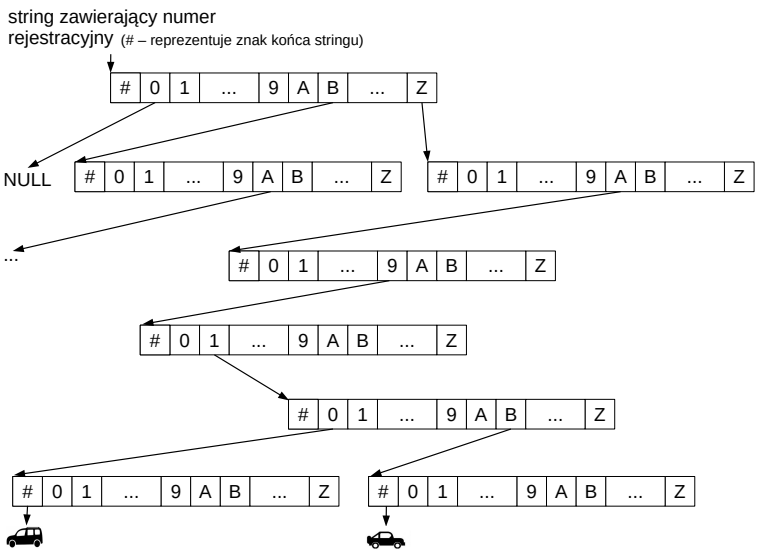
Lista cykliczna:



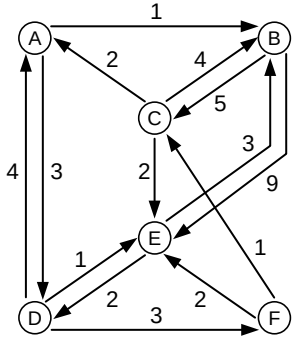
Drzewo poszukiwań binarnych (BST):



Tablicowo-łączowe struktury danych – przykład: B-drzewo

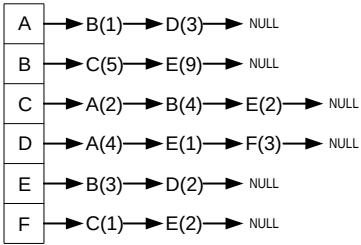


Alternatywy w wyborze struktury danych



	A	B	C	D	E	F
A	–	1	–	3	–	–
B	–	–	5	–	9	–
C	2	4	–	–	2	–
D	4	–	–	–	1	3
E	–	3	–	2	–	–
F	–	–	1	–	2	–

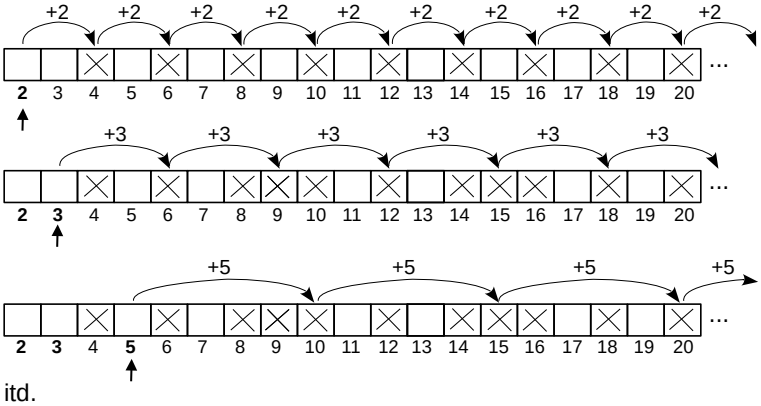
Różne struktury
przechowujące taki
sam graf skierowany



Algorytm a struktura danych

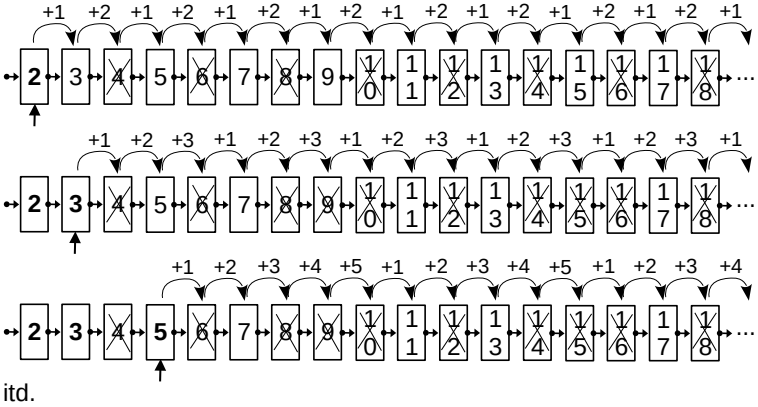
Algorytm i struktura danych są ze sobą mocno związane: wybór algorytmu implikuje wybór struktury danych i na odwrót

Algorytm a struktura danych – sito Eratostenesa



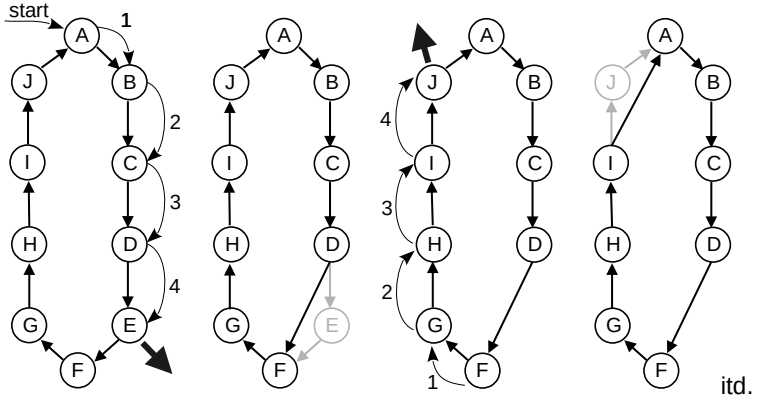
Użycie tablicy jest naturalne: implementacja jest łatwa (kolejne liczby to po prostu indeksy tablicy), działanie przebiega sprawnie.

Algorytm a struktura danych – sito Eratostenesa



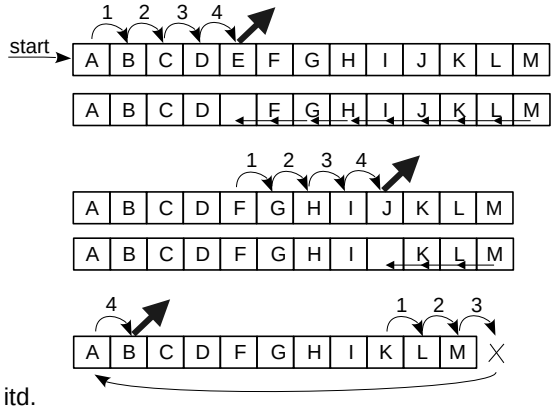
Użycie listy jest możliwe, ale implementacja mniej wygodna (trzeba stworzyć listę i wypełnić jej elementy kolejnymi liczbami naturalnymi), działanie jest mało efektywne szybkościowo.

Algorytm a struktura danych – eliminacja Josephusa



Użycie listy cyklicznej pozwala na łatwość implementacji i dużą sprawność działania.

Algorytm a struktura danych – eliminacja Josephusa



Użycie tablicy jest możliwe, ale niewygodne – wymuszające konieczność wykonywania dodatkowych czasochłonnych czynności (lub przy innej implementacji – alokacji dodatkowych informacji).

Typ danych

Zbiór wartości (informacji) oraz zbiór operacji, jakie można na tych wartościach wykonać

Typ danych

- **prosty**
  Integer 32-bitowy: –32 768 ÷ 32767 oraz dodawanie, odejmowanie, mnożenie itd.
- **złożony (agregacyjny, zenkapsulowany)**
  np. tzw. krotka {dana1, dana2, dana3}: wszystkie kombinacje dopuszczalnych wartości danych 1, 2, 3 oraz jakieś operacje na tych danych.

Instancja typu danych (instancja danych)

Pojedynczy egzemplarz informacji należący do danego typu danych

Implementacja typu danych

Typ danych
jest pojęciem na wysokim poziomie abstrakcji (teoretycznym).
Jest to ogólnie pojęta informacja + ogólnie pojęte operacje.
Praktyczne użycie typu danych wymaga jego implementacji.

- Implementacja prostych typów danych wbudowana jest zazwyczaj w język programowania[1].
- Niektóre „standardowe" agregacyjne typy danych mogą być wbudowane w języki programowania[1] i/lub dostarczane przez biblioteki i zręby (frameworki).
- Programista może sam tworzyć potrzebne mu agregacyjne typy danych.

[1] W wielu językach używa się więc pojęcia „typ wbudowany".

Implementacja typu danych

- Do wewnętrznej organizacji informacji w implementacji typu danych stosowane są jakieś struktury danych (mniej lub bardziej złożone).
- Programista korzystający z wbudowanego lub bibliotecznego typu danych nie musi znać zastosowanej struktury (i na ogół jej nie zna – nie jest to potrzebne).

Abstrakcyjny typ danych

Interfejs

Operacje (procedury) stowarzyszone z typem danych, dzięki którym program ma dostęp do informacji zawartych w danej instancji i może na nich operować[a]

[a] Zatem definicja typu danych mogłaby brzmieć: Jest to zbiór wartości i jego interface.

Abstrakcyjny typ danych

Typ danych, dla którego informacje zawarte w instancjach są dostępne wyłącznie za pośrednictwem interfejsu (występuje tu tzw. hermetyzacja)[a]

[a] Programista nie bezpośredniego dostępu do wewnętrznej struktury danych.

W pełni **zhermetyzowany** typ danych oznacza, że instancja tych danych jest **czarną skrzynką** o nieznanej strukturze wewnętrznej, więc „świat zewnętrzny" nie może bez „zgody" typu czegokolwiek tam zmienić

Konsekwencja hermetyzacji

Wewnętrzna budowa abstrakcyjnego typu danych jest niezależna od reszty programu, Można zupełnie przeorganizować tę budowę (czyli użytą strukturę danych) i zmienić zastosowane wewnętrzne algorytmy nie zmieniając interfejsu. Wtedy, mimo tak ogromnych zmian, cały „zewnętrzny" program stosujący ten typ danych nie musi być modyfikowany – będzie działał tak samo.

Podstawowy abstrakcyjny typ danych

Kolejka uogólniona

Składa się z danych oraz co najmniej dwóch operacji interfejsu: wstaw (włóż, push, insert, put) i usuń (wyjmij, pull, remove, get)

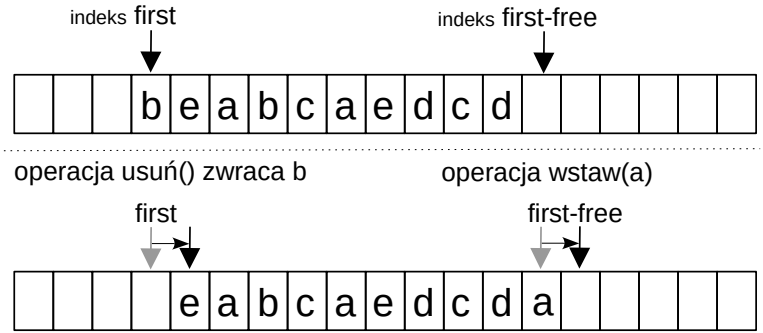
Najważniejsze rodzaje kolejek uogólnionych

- **stos (kolejka LIFO)**
- **kolejka (kolejka FIFO)**
- **kolejka priorytetowa**
- **słownik**
- **kolejka losowa**

Inne przykładowe typowe operacje obecne w interfejsach kolejek: wyszukaj (search), wybierz (select), uporządkuj (sort), połącz (join)

Kolejki – przykłady implementacji

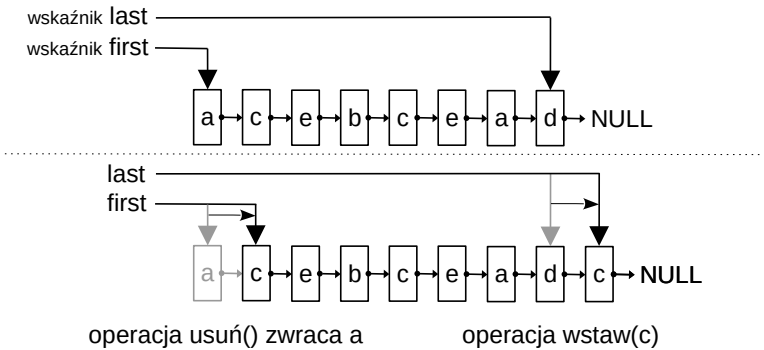
- **Kolejka FIFO – implementacja z użyciem tablicy:**



Kiedy indeks first-free wyjdzie poza rozmiar tablicy należy go „zresetować" na pierwszy element tablicy.
Kolejka działa poprawnie dla liczby elementów mniejszej o jeden od rozmiaru tablicy.
Można tego uniknąć implementując dodatkowo procedurę powiększenia (realokacji) tablicy w przypadku przekroczenia jej rozmiaru.

Kolejki – przykłady implementacji

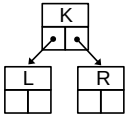
- **Kolejka FIFO – implementacja z użyciem listy:**



Liczba elementów w kolejce jest nieograniczona.

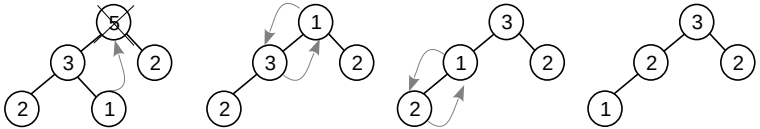
Kolejki – przykłady implementacji

- **Kolejka priorytetowa – implementacja z użyciem kopca:**



Drzewo binarne, w którym zawsze klucze węzłow spełniają warunki: K >= L oraz K >= P.
Zatem w korzeniu całego drzewa (na najwyższym poziomie) jest klucz o największej wartości.

Pu usunięciu korzenia trzeba przywrócić własność kopca:



Podobna procedura pozwala przywrócić własność kopca po wstawieniu nowego węzła (na pierwsze wolne miejsce).

Uwaga: Mimo, że jest to struktura łączowa kopiec wygodnie jest implementować w tablicy! Umieszcza się w niej kolejne węzły z kolejnych poziomów – wtedy dla korzenia o indeksie i lewy potomek ma indeks $2i$, a prawy $2i + 1$.

Kolekcja standardowa (kontener standardowy)

Narzędzie programistyczne (będące elementem języka programowania lub dostarczane jako biblioteczne), które jest kolejką uogólnioną do przechowywania instancji danych dowolnego rodzaju, mające cechy funkcjonalne związane z jakąś strukturą danych i mające jakąś charakterystyczną funkcjonalność

Uwaga

W ogólności kontener (kolekcja) to każdy typ zagregowany mający cechy kolejki uogólnionej (zarówno wbudowany w język lub biblioteczny, jak i zdefiniowany przez programistę na potrzeby jego programu)

Podstawowe rodzaje kolekcji – 1

Uwaga: W poszczególnych językach programowania niektóre typy kolekcji nie muszą być dostępne.
Uwaga: Dla różnych języków programowania kolekcje danego rodzaju mogą różnić się funkcjonalnością.

Tablica (Array)

Interfejs gwarantuje, że zawarte w niej elementy są uporządkowane jeden za drugim. Dostęp do nich uzyskuje się podając ich numer kolejny (wartość indeksu).

Zbiór (Set)

Interfejs nie gwarantuje żadnego uporządkowania elementów. Kolejność dostępu do poszczególnych instancji jest przypadkowa

Podstawowe rodzaje kolekcji – 2

Lista (List)

Interfejs gwarantuje, że zawarte w niej elementy są uporządkowane jeden za drugim. Kolejność dostępu do poszczególnych instancji jest sekwencyjna – przeszukiwanie zaczyna się zawsze od pierwszego elementu

Słownik (Dictionary)

Elementy mają przyporządkowane unikatowe słowa kluczowe (etykiety). Interfejs gwarantuje dostęp do poszukiwanej instancji po podaniu takiego słowa.

Uwaga: W różnych językach programowania takie same kolekcje mogą mieć różne nazwy. Także takie same funkcjonalnie procedury interfejsu mogą być inaczej nazywane.
Uwaga: W poszczególnych językach programowania dane rodzaje kolekcji mogą występować w kilku odmianach różniących się nieco funkcjonalnością i/lub właściwościami.