Paradygmaty Programowania

Marcin Bączyk

Wykład 3

8 marca 2021

Treść dzisiejszego wykładu

- wstęp do architektury oprogramowania
- przegląd sposobów programowania
  - strukturalne
  - obiektowe
  - funkcyjne
- programowanie funkcyjne
  - programowanie deklaratywne
  - charakterystyka
  - funkcje w ujęciu matematycznym
  - funkcje w ujęciu programistycznym
  - funkcje wyższych rzędów
  - rachunek λ i domknięcia
  - rekurencja

Literatura

- Mark Richards, Neal Ford: "Podstawy architektury oprogramowania dla inżynierów"
- Robert C. Martin: "Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów"
- Joshua Backfield: "Programowanie funkcyjne. Krok po kroku"
- Čukić Ivan: "Programowanie funkcyjne w języku C++. Tworzenie lepszych aplikacji"

Architektura oprogramowania

Architektura oprogramowania

– podstawowa organizacja systemu wraz z jego komponentami, wzajemnymi powiązaniami, środowiskiem pracy i regułami ustanawiającymi sposób jej budowy i rozwoju.

"Architektura jest tym czego nie można znaleźć w Google".

Architektura oprogramowania składa się z:

- **struktury systemu**, oraz
- **parametrów architektury**, wspieranych przez
- **decyzje architektoniczne** oraz
- **zasady projektowania**.

Architektura oprogramowania

"Architektura reprezentuje ważną decyzję projektową, która wpływa na kształt systemu przy czym waga decyzji mierzona jest kosztami zmian, które wprowadza". Grady Booch

"Jeżeli uważasz, że dobra architektura jest droga, to wypróbuj złą architekturę". Brian Foote i Joseph Yoder

Celem architektury oprogramowania jest zminimalizowanie liczby ludzi (ogólniej środków) wymaganych do zbudowania i utrzymywania danego systemu.

Prawa architektury oprogramowania

1. W architekturze oprogramowania wszystko jest **kompromisem**.
2. **Dlaczego** jest ważniejsze niż **jak**.

Architektura oprogramowania

Funkcjonalność (zachowanie)

- Spełnienie określonego zadania przez tworzony program: obliczenie wartości funkcji albo umożliwienie zarządzania dużym i złożonym systemem (np. lotnisko).
- Decyduje o wartości tworzonego oprogramowania.

Architektura

- Nie zapewnia spełnienia ani jednego zadania stawianego przed tworzonym programem.
- Użytkownicy nie interesują się strukturą działającego oprogramowania; zazwyczaj nie mają dostępu do kodu źródłowego i nie wiedzą jak system jest zbudowany.
- Nie ma dużego wpływu na wartość oprogramowania w danej chwili.

Architektura oprogramowania

Co jest ważniejsze : dostarczenie funkcjonalności czy zapewnienie dobrej architektury ?

- Oprogramowanie w kontrze do sprzętu powinno mieć możliwość ciągłej modyfikacji.
- Zmiany wymuszone są czynnikami zarówno wewnętrznymi jak i zewnętrznymi.
- Łatwość wprowadzania zmian decyduje o przewadze konkurencyjnej.
- Wraz z rozwojem systemu rośnie koszt utrzymywania i rozwijania.

- Wprowadzając ciągłe zmiany do rozbudowanego systemu bez odpowiedniej architektury skutkuje wzrostem kosztów wprowadzania jednostkowej zmiany
- W konsekwencji system dąży do stanu w którym jakakolwiek zmiana jest nieopłacalna (koszt $>$ potencjalnego zysku).
- Taki system należy porzucić lub chociaż częściowo przepisać od początku (duże koszty).

Architektura oprogramowania

Problemy ważne i pilne

"Te pilne nie są ważne, a ważne nie są pilne". Dwight D. Eisenhower

- Funkcjonalność oprogramowania jest pilna ale zazwyczaj nie jest szczególnie ważna.
- Architektura oprogramowania jest ważna ale zazwyczaj nie jest szczególnie pilna.

Hierarchia problemów:

1. ważne i pilne
2. ważne ale niezbyt pilne
3. pilne ale nieważne
4. niepilne i nieważne

Istotne jest aby nie przenosić punktów 3 i 4 na szczyt listy.

Przegląd sposobów tworzenia oprogramowania

Programowanie strukturalne

Skoki zastąpiono instrukcjami **if/then/else** oraz **for/do/while/until**.

"Programowanie strukturalne wymusza dyscyplinę bezpośredniego przekazywania sterowania".

Stosowane jest jako algorytmiczna podstawa tworzonych modułów.

Programowanie obiektowe

Stos wywołań funkcji można przenieść na stertę, a zmienne lokalne zadeklarowane w funkcji mogą istnieć po jej zakończeniu.

"Programowanie obiektowe wymusza dyscyplinę pośredniego przekazywania sterowania".

Polimorfizm wykorzystywany jako metoda przekraczania granic poszczególnych modułów.

Przegląd sposobów tworzenia oprogramowania

Programowanie funkcyjne

Niezmienność jako fundamentalna cecha rachunku lambda.

"Programowanie funkcyjne wymusza dyscyplinę podczas przypisywania wartości".

Programowanie funkcyjne wykorzystywane do zaprowadzenia dyscypliny z dostępie do danych.

Każdy ze sposobów programowania nakłada dodatkowe ograniczenia nie dając nic w zamian. Uwaga skupiona jest na tym czego nie robić. Rzadko udaje się uzyskać informację na temat co należy robić. - **doświadczenie**.

Decydując się na któryś ze sposobów programowania zmuszeni jesteśmy do **kompromisu**.

Programowanie strukturalne

Cechy charakterystyczne:

- struktury kontrolne
  - sekwencja
  - wybór
  - iteracja
- podprogramy
- bloki

Konsekwencją stosowania podejścia czysto strukturalnego jest pojedyncze wyjście.
W ramach wykładu jako pojedyncze wyjście oznaczać będzie sytuacja w której funkcja / procedura zawiera pojedyncza instrukcję zwracającą obliczony wcześniej wynik.

Programowanie strukturalne

Przy pomocy struktur kontrolnym oraz podprogramów możliwe jest stworzenie dowolnego oprogramowania.

Jednakże programowanie jest trudne, a duża złożoność systemów informatycznych skutkuje dużą podatnością na pojawianie się błędów.

W związku z tym, aby zmniejszyć lokalnie złożoność systemu program dzieli się na podprogramy.

Aby zapewnić "bezbłędność" oprogramowania dowodzi się poprawności każdego z fragmentów programu. Zmniejszenie ilości kodu każdego podprogramu mniejsza złożoność dowodu.

Programowanie strukturalne

Dekompozycja funkcyjna

- Istota programowania strukturalnego;
- Rekursywna dekompozycja modułów aż do osiągnięcia poziomu dla którego możliwe jest przeprowadzenie dowodu;
- Matematyczne dowodzenie poprawności kodu jest pracochłonne i trudne; obecnie praktycznie nie stosowane.

Testowanie

- Metoda naukowa;
- Sprawdzanie poprawności kodu odbywa się poprzez próbę stwierdzenia ich niepoprawności;
- W przypadku niemożliwości stwierdzenia niepoprawności kodu przyjmuje się, że jest on pozbawionych błędów.

Programowanie strukturalne

Tworzenie oprogramowania nie jest zagadnieniem matematycznym.

W programowaniu strukturalnym:

- Podstawą są instrukcje sterujące przepływem programu i bloki instrukcji często organizowane w podprogramy
- Problem (program) dekomponowany jest na dostatecznie małe podprogramy, dla których można przeprowadzić dowód poprawności lub jeżeli dowiedzenie jest niemożliwe, lub nie da się dowieść ich niepoprawności.
- Główny nacisk kładziony jest na to **w jaki sposób** wykonać dane zadanie.

Cechy charakterystyczne:

- abstrakcja
- hermetyzacja
- dziedziczenie
- polimorfizm

Konsekwencją stosowania podejścia czysto obiektowego jest przypisanie każdej operacji w systemie konkretnemu obiektowy i przekazanie mu odpowiedzialności za tę operację.

Programowanie obiektowe

Programowanie obiektowe jest odpowiedzią na potrzebę modelowania świata rzeczywistego w możliwie wierny sposób (z dokładnością do poziomu abstrakcji). Podstawowe cechy programowania obiektowego ułatwiają tworzenie oprogramowania dla konkretnej dziedziny.

Żadna z podstawowych cech programowania obiektowego nie musi być wspierana przez dany język aby móc napisać w nim program obiektowy.

Systemy projektowane obiektowo składają się z dużej ilości "małych" obiektów współpracujących ze sobą. Interfejs publiczny, oznaczający usługi jakie obiekt może świadczyć swoim klientów informuje jedynie o tym co może być zrobione. Klienci poszczególnych obiektów nie ingerują w to w jaki sposób dana usługa jest świadczona.

Systemy obiektowe projektowane są w taki sposób aby obiekt odpowiedzialny za daną funkcjonalność mógł być zastąpiony innym bez zmiany kodu klienta.

Programowanie obiektowe

W programowaniu obiektowym:

- Podstawą systemu są współpracujące ze sobą obiekty;
- Obiekty mogą być przekazywane pomiędzy modułami bez wprowadzania dodatkowych (zbędnych) zależności;
- Klienci danego obiektu nie wiedzą w jaki sposób realizowana jest jego odpowiedzialność;
- Moduły wysokiego poziomu nie zależą od modułów niskiego poziomu;
- Rozwój (dodawanie nowych) funkcjonalności odbywa się bez zmiany wysokopoziomowych reguł biznesowych;
- Główny nacisk kładziony jest na to **kto ma** wykonać dane zadanie.

Cechy charakterystyczne:

- brak przypisań
- brak pętli
- czyste funkcje

Konsekwencją stosowania podejścia czysto funkcyjnego jest brak jakiegokolwiek modyfikowalnego stanu wewnętrznego programu.

Programowanie funkcyjne

System projektowany funkcyjnie składa się z funkcji, które przyjmują argumenty i zwracają wyniki obliczeń. Funkcje wchodzące w skład systemu składają się z innych funkcji, aż do pojedynczych bloków instrukcji - mechanizm dekompozycji funkcyjnej.

Podobnie do programowania obiektowego istotniejsze jest to co robi dana funkcja niż to w jaki sposób to robi. Często obiekty (niezmienne!) są wykorzystywane jako obiekty funkcyjne - funkcje posiadające pewien niemodyfikowalny stan wewnętrzny.

Programowanie funkcyjne obecnie jest odpowiedzią na problemy pojawiające się w aplikacjach współbieżnych. Niemożliwość zmiany stanu, któregokolwiek z elementów działającego programu usuwa możliwość pojawienia się wyścigów i zakleszczeń.

Programowanie funkcyjne

W programowaniu funkcyjnym:

- Minimalizowana jest liczba elementów które mogą ulegać zmianie, a mechanizmy zmiany podlegają ścisłej kontroli
- Zakłada się, że czas niezbędny na ponowne wykonanie obliczeń jest pomijalnie mały i w związku z tym wartości zmiennych nie są zapamiętywane i modyfikowane
- Zakłada się, że ilość pamięci potrzebnej do przechowywania wszystkich niezbędnych informacji jest nieograniczona
- Główny nacisk kładziony jest na to **co ma być** zrobione.

Wnioski

Poszczególne sposoby programowania nakładają w programie różne ograniczenia:

- strukturalne – na bezpośrednie przekazywanie sterowania,
- obiektowe – na pośrednie przekazywania sterowania,
- funkcyjne – na przypisywanie wartości zmiennym,

przy czym nie oferują niczego dodatkowego.

Wszystkie przedstawione sposoby tworzenia oprogramowania znane były już w latach 60-tych XX wieku. Ze względu na rożne ograniczenia w danej epoce rożne sposoby programowania stawały się dominującymi.

Natomiast od momentu stworzenia pierwszych programów reguły programowania nie uległy zasadniczej zmianie. Programy składają się z sekwencji, selekcji, iteracji i pośrednictwa.

Wnioski

wniosek nadrzędny

Poza pewnymi skrajnymi przypadkami nie istnieje podejście czysto strukturalne, obiektowe czy funkcyjne do wytwarzania oprogramowania.

Duże systemy informatyczne składają się z wielu modułów, które mogą być napisane z wykorzystaniem rożnych sposobów programowania.

Programowanie deklaratywne

- Programista opisuje (deklaruje) co ma być zrobione (jaki ma być efekt działania programu), a nie w jaki sposób ma to być zrobione (jaka jest sekwencja działań prowadząca do celu).
- Charakteryzuje się minimalną ilością skutków ubocznych (co w znaczący sposób upraszcza opracowywanie programów współbieżnych).

Program deklaratywny

- Jest niezależny — wynik końcowy nie zależy od żadnego zewnętrznego stanu.
- Jest bezstanowy — nie posiada stanu wewnętrznego, który który mógłby się zmieniać między wywołaniami.
- Jest deterministyczny — dla takich samych argumentów wejściowych **zawsze** zwraca ten sam wynik.

programowanie funkcyjne

Programowanie funkcyjne jest przykładem programowania deklaratywnego. Problem (lub ogólniej świat) modelowany jest przy pomocy pojęć matematycznych. Wraz z zaawansowaniem modelu wykorzystywane struktury matematyczne, przyjmujące postać stałych i funkcji również stają się coraz bardziej zaawansowane.

Czyste programowanie funkcyjne polega tworzeniu coraz bardziej złożonych funkcji i stałych oraz wyznaczaniu wartości zmiennych.

Charakterystyka programowania funkcyjnego

- brak przypisań $->$ stałe i funkcje
- brak pętli $->$ rekurencja
- relatywnie łatwo napisać poprawny program
- łatwiej wykazać poprawność programu
- czyste funkcje
- funkcje wyższych rzędów

Deklaratywny charakter

- Nacisk kładziony jest na opisanie wyniku działania każdego fragmentu programu.
- Często konkretny problem przedstawiany jest jako szczególny przypadek bardziej ogólnego zadania.

Przykład - wyznaczenia sumy elementów w kolekcji

Podejście imperatywne:

- Dodawanie kolejnych elementów do wyniku
- Zgodne ze sposobem w jaki działają współczesne procesory, które wykorzystują głównie proste operacje arytmetyczne oraz instrukcje skoków.

Podejście deklaratywne:

- Suma dowolnego elementu zbioru oraz sumy pozostały jego elementów.
- Mimo, że wydaje się być mniej intuicyjna, to de facto jest bliższa rzeczywistości.

Języki funkcyjne i programowanie funkcyjne

- Posiadają instrukcje pozwalające w łatwy sposób wyrażać idee wysokiego poziomu w oderwaniu od konkretnej architektury systemu.
- Programy ze względu na wykorzystywane konstrukcje zazwyczaj są wykonują się wolniej niż ich imperatywne odpowiedniki.
- Pierwotnie programowanie funkcyjne nie cieszyło się one dużą popularnością i stanowiło raczej ciekawe zagadnienie do badania dla naukowców.
- Obecnie coraz częściej stosowane do wielu zagadnień biznesowych (np. w architekturze bazującej na mikrousługach).
- Na przestrzeni lat powstało wiele udanych implementacji języków funkcyjnych.
- Programy napisane fukcyjnie charakteryzują się dużą czytelnością i efektywnością powstawania kodu.
- Czas wykonywania programu i pamięć niezbędna do jego uruchomienia, ze względu na olbrzymi rozwój systemów komputerowych, nie jest głównym ograniczeniem z jakie należy rozważać.

Funkcje w ujęciu matematycznym

- funkcja prosta

$$f(x) = ax^2 + bx + c$$

- funkcja złożona

$$f(x, g, h) = g(x) \cdot h(x)$$

- funkcja rekurencyjna, instrukcja warunkowa

$$n! := \begin{cases} 1, & \text{dla } n = 0 \\ n \cdot (n-1)!, & \text{dla } n >= 1. \end{cases}$$

- suma / iloczyn, pętla

$$n! = \prod_{k=1}^{n} k$$

Funkcje w ujęciu programistycznym

Funkcje (lub procedury) to inaczej podprogramy odpowiedzialne za wyodrębniony element programu. Podobnie do matematycznych odpowiedników w programowaniu również wyróżnia się kilka rodzajów funkcji:

- funkcja prosta - funkcja złożona z instrukcji wykonywanych sekwencyjnie.
- funkcja złożona - funkcja wywołująca inne funkcje w celu wyznaczenia niezbędnych wartości
- funkcja rekurencyjna - funkcja, wywołująca samą siebie, niezbędna jest instrukcja warunkowa w celu zakończenia ciągu wywołań

- Wiele funkcji, nie tylko matematycznych, daje się przedstawić zarówno w postaci rekurencyjnej jak i sumy lub iloczynu wyrażeń.
- W programowaniu funkcyjnym ze względu na prostotę wyrażenia i łatwość dowodzenia poprawności kodu preferowana jest postać rekurencyjna.

$$n! = \prod_{k=1}^{n} k$$

Przedstawienie wzoru na obliczenie silni liczby naturalnej ($n!$) w postaci iloczynu kolejnych liczb dodatnich mniejszych lub równych n jest bliższe imperatywnemu podejścu do programowania. Definicja w tej postaci dokładnie prezentuje w jaki sposób należy wykonać obliczenia w celu wyznaczenia wyrażenia. W trakcie jej obliczania aktualizowana jest wartość iloczynu, co jest sprzeczne z założeniami programowania funkcyjnego (bezstanowy). Stanem w tym przypadku można określić wartość zmiennej przechowującej kolejne wartości iloczynu.

$$n! := \begin{cases} 1, & \text{dla } n = 0 \\ n \cdot (n-1)!, & \text{dla } n >= 1. \end{cases}$$

Z kolei przedstawienie wzoru na obliczenie silni liczby naturalnej ($n!$) w postaci iloczynu wartości $(n-1)!$ oraz n jest bliższe deklaratywnemu podejścu do programowania. Wyrażenie tej postaci odwołuje się do matematycznej definicji czym jest silnia. Warto zwrócić uwagę, że bezpośrednio w definicji nie jest przedstawiony sposób obliczenia wyważenia $(n-1)!$. Program nie ma też zmiennej, która mogłaby być aktualizowana.

funkcje wyższego rzędu

W programowaniu funkcyjnym funkcje są takimi samymi wartościami jak wszystkie inne. Oznacza to, że funkcje mogą być:

- argumentami innych funkcji oraz
- wynikami zwracanymi przez inne funkcje.

Funkcje, które operują na innych funkcjach nazywamy funkcjami wyższych rzędów.

polimorfizm funkcji

Zarówno funkcje przekazywane jako argumenty do innych funkcji jak i funkcje przez nie zwracane mogą mieć różną postać. Na przykład istnieje nieskończenie wiele funkcji mapujących zbiór liczb rzeczywistych w zbiór liczb zespolonych ($g : \mathbb{R} \mapsto \mathbb{C}$). Jeżeli dana funkcja $f(x, g) : \mathbb{R} \mapsto \mathbb{R}$ oczekuje takiej funkcji przekazanej jako argument to może ona być dowolnej postaci. Taką sytuację nazywamy **polimorfizmem** – wielopostaciowością.

Rachunek λ

Mając daną funkcję złożoną

$$f(x, g, h) = g(x) \cdot h(x)$$

możemy chcieć w miejsce argumentów g lub h przekazać funkcję bez jej formalnego definiowania W tym celu wykorzystywany jest rachunek lambda . Funkcje f, g oraz h są formalnie zdefiniowane, mają swoją nazwę i prawdopodobnie adres w pamięci fizycznej. Często w przypadku bardzo prostych wyrażeń, takich jak na przykład wielomiany niewielkich stopni, lepiej jest nie definiować funkcji a utworzyć są ad hoc i przekazać jako argument wprost w wywołaniu. Matematycznie można zapisać to w następujący sposób:

$$f(10, y \mapsto y^2, y \mapsto 1 - \frac{1}{y}) = (y \mapsto y^2)(10) \cdot (y \mapsto 1 - \frac{1}{y})(10)$$

Domknięcia

Domknięciem nazywamy funkcję λ, która odwołuje się do zmiennych spoza zakresu funkcji:

$$f(10, y \mapsto y^2, y \mapsto 1 - \frac{1}{y} + c) = (y \mapsto y^2)(10) \cdot (y \mapsto 1 - \frac{1}{y} + c)(10).$$

W powyższym przykładzie zmienna c nie występuje jako argument funkcji λ. Nie została też zdefiniowana wewnątrz jej ciała. Zmienna ta została przechwycona, a utworzona funkcja wywoła się w sposób poprawny.

Funkcje λ oraz domknięcia nie są implementowane we wszystkich językach, zwłaszcza niefunkcyjnych. W niektórych językach programowania funkcjonalności te mogą być mocno ograniczone zwłaszcza w przypadku zwracania funkcji jako argumentu.

Należy zwrócić uwagę czy odwołanie do przechwytywanej zmiennej (lub funkcji) w danej funkcji λ odbywa się przez kopiowanie czy referencję. Nie w każdym języku możliwe jest utrzymanie zmiennej poza ciałem funkcji w przypadku istniejących do niej innych odwołać (np. przez domknięcie).

Zastosowania procedur wyższych rzędów

- Niektóre pojęcia matematyczne w sposób naturalny operują na funkcja wyższych rzędów. *Przykład* : Funkcja wyznaczająca pochodną funkcji jako argument przyjmuje funkcję i zwraca funkcję w innej postaci.
- W wielu miejscach w programie powtarzany jest ten sam kod, różniący się jedynie fragmentami. Fragmenty te mogą być ujęte w postaci procedur, które następnie będą parametrami funkcji bardziej ogólnych.
  Przykład : Procedura sortowania rekordów może się różnić fragmentem dotyczącym porównania dwóch elementów, zaś sama implementacja algorytmu będzie taka sama we wszystkich przypadkach. Mechanizm porównywania wartości można przekazać do procedury w postaci osobnej funkcji.

Skutki uboczne

Jednym z podstawowych założeń programowania funkcyjnego jest minimalizacja skutków ubocznych działania wszystkich funkcji. Z punktu widzenia programowania funkcyjnego niepożądane skutki uboczne jakie mogą się zdarzyć to między innymi:

- zmiana wartości zmienne globalnej,
- zmiana wartości argumentu,
- przydzielenie zasobu, i jego późniejsze niezwrócenie.

Wszystkie powyższe przypadki, to niekorzystne skutki uboczne, utrudniające czytanie i rozumienie kodu programu. Odczytując instrukcję zakładamy, że nie zmienia ona stanu programu, który mógłby zaburzyć działanie kolejnych instrukcji.

Skutki uboczne, jakimi są zmiany stanu systemu powodują, również, że dwukrotne wywołanie tej samej funkcji nie musi zakończyć się tym samym wynikiem dla tych samych danych wejściowych.

Skutki uboczne

Jednak programy z definicji nie mogą istnieć bez skutków ubocznych. Nie wszystkie skutki uboczne są niepożądane. Poniżej kilka przykładów „przydatnych" skutków ubocznych działania funkcji:

- wyświetlenie informacji na ekranie lub wyczyszczenie ekranu,
- zapisanie danych do pliku, bazy danych lub wysłanie przez sieć,
- zmiana pola w rekordzie danych.

Funkcje i procedury, które zmieniają stan systemu są niezbędne do poprawnego działania programu. Bez jakiejkolwiek formy wyświetlenia wyniku, sam wynik jest bezużyteczny. Należy jednak zwrócić uwagę by funkcje, które mają swoje skutki uboczne, informowały o tym poprzez swoją nazwę, np. *wyświetl*, *wyczyśćEkran* lub *zapiszWynikNaDysku*.

Zmienne niemutowalne

Zmienne w systemie komputerowym mogę być:

- mutowalne oraz
- niemutowalne.

Zazwyczaj poprzez pojęcie zmiennej rozumie się wartość, która może ulega zmianie w trakcie swojego życia. Natomiast w programowaniu funkcyjnym przyjmuje się zmienne są niemutowalne. Oznacza to, że:

- zmienne się nie zmieniają w czasie swojego życia oraz, że
- jedynie zmienne globalne mogą zmieniać swoje referencje.

W przypadku zmiany referencji zmiennej globalnej poprzedni obiekt zostaje zniszczony, a w jego miejsce utworzony nowy.

Niemutowanie zmiennych ułatwia utrzymanie ich prawidłowego stanu programu. Zmienne utworzone w prawidłowych sposób i zawierające prawidłowe dane nie zmienią tego stanu.

Przykład

Wśród grupy studentów, ze wszystkich występujących imion i nazwisk, tylko niektóre ich kombinacje oznaczają konkretne osoby. Przez funkcję starosty rozumiemy osobę występującą w imieniu całej grupy studentów. W tym przykładzie funkcję tę należy rozumieć jako referencję do zmiennej globalnej, gdyż nie jest to cecha żadnej z osób w grupie.

Jeżeli zmianie ulegać będą kolejno imię i nazwisko starosty w pewnym etapie (dokładnie pomiędzy zmianą imienia a nazwiska) program znajdzie się w nieprawidłowym stanie, gdyż jako starosta będzie przypisana osoba, która nie istnieje – imię i nazwisko nie będzie identyfikowało żadnej osoby z grupy.

Natomiast, jeżeli utworzymy nowy rekord (wraz z imieniem i nazwiskiem nowego starosty) a następnie zostanie on wskazany jako nowy starosta program przez cały czas będzie w stanie prawidłowym.

Rekurencja

Ponieważ w programowaniu funkcyjnym zmienne są niemutowalne to konstrukcje takiej jak pętle nie mają zastosowania w prost. Dzieje się tak, że zazwyczaj pętle w trakcie kolejnych iteracji aktualizują zmienne lokalne, by następnie zwrócić wynik. Aby móc rozwiązać ten problem należy posłużyć się rekurencją.

Listy

W przypadku rekurencji i operacji na zbiorach istotnym pojęciem są listy. Z punktu widzenia programowania funkcyjnego listą może być dowolny typ zbioru, który umożliwia wyznaczenie:

- głowy listy oraz
- ogona listy.

Poprzez głowę listy będziemy rozumieli pierwszy jej element, zaś jako ogon listy wszystkie pozostałe.

Listy i rekurencja

Wykorzystując pojęcie listy, operacje wykonywane na zbiorach elementów można przedstawić w postaci ogólnego algorytmu:

1. Weź pierwszy element zbioru. Czasami może to być też dowolny element tego zbioru.
2. Wykonaj żądaną operację na elemencie.
3. Scal wynik operacji z wynikiem wywołania algorytmu dla pozostałych elementów ze zbioru.

Przykład

Sumę zbioru liczbowego można przedstawić jako sumę dowolnego jego elementu oraz sumę wszystkich pozostałych. Podobnie znajdowanie elementu maksymalnego można przedstawić jako wynik porównania danego, dowolnie wybranego elementy z wartością maksymalną pozostałej części zbioru.

Przykład

Filtrowanie zbioru elementów, ze względu na funkcję celu można przedstawić jako scalenie wyniku zwracanego dla pojedynczego wyniku, który może być zbiorem pustym, z wynikiem zwracanym dla pozostałych elementów zbioru, który również może być pusty.

Przykładowy problem programistyczny

Wyznaczyć średnią ilość godzin w tygodniu poświęconych na naukę wśród studentów, którzy mają średnią ważoną ze wszystkich przedmiotów wyższą lub równą 4.0.