Szkoła Gospodarstwa Wiejskiego
Wydział Zastosowań Informatyki i Matematyki

PARADYGMATY PROGRAMOWANIA

Jednostka 2 – Proceduralny i strukturalny

Kierunek: Informatyka, semestr 5
Prowadzący: dr inż. Krzysztof Malczewski

1. Cel i zakres zajęć

Celem ćwiczeń jest:

- zrozumienie zasad **paradygmatu proceduralnego** i **strukturalnego**,
- umiejętność **wydzielania funkcji** w celu modularnego organizowania kodu,
- stosowanie instrukcji sterujących w sposób strukturalny, bez „skoków” i `goto`,
- przepisanie programu imperatywnego do wersji proceduralnej,
- porównanie i analiza korzyści.

Zakres tematyczny:

- definicje i charakterystyka paradygmatów proceduralnego i strukturalnego,
- dekompozycja programu na funkcje,
- przekazywanie parametrów i zwracanie wartości,
- instrukcje sterujące: `if`, `else`, `switch`, `for`, `while`, `break`, `continue`,
- zasada „jednego wejścia i jednego wyjścia” z bloków,
- refaktoryzacja programu z jednostki 1.

---

2. Wprowadzenie teoretyczne

2.1 Paradygmat proceduralny

Paradygmat proceduralny jest rozwinięciem imperatywnego.
Program jest **zbiorem procedur (funkcji)**, z których każda realizuje określone zadanie.
Zamiast jednego „ciągu instrukcji”, mamy **hierarchię wywołań funkcji**.

Cechy:

- Dekompozycja problemu na funkcje,
- Przekazywanie danych poprzez parametry,
- Unikanie powtórzeń kodu,
- Ułatwienie testowania i ponownego użycia.

2.2 Paradygmat strukturalny

Strukturalne programowanie to styl pisania kodu, który:

- eliminuje nieuporządkowane skoki (`goto`),
- stosuje **czytelne instrukcje sterujące**,
- ma **jedno wejście i jedno wyjście** z bloków,
- organizuje kod w logiczne, zagnieżdżone bloki.

Korzyści:

- Łatwiejsze rozumienie przepływu,
- Mniejsze ryzyko błędów logicznych,
- Lepsza czytelność i konserwowalność.

---

3. Tutorial 1 – Refaktoryzacja programu imperatywnego

3.1 Punkt wyjścia

W jednostce 1 studenci napisali program w stylu imperatywnym obliczający średnią i sortujący dane. Teraz dokonamy **refaktoryzacji do wersji proceduralno-strukturalnej**.

3.2 Projekt funkcji

Dekompozycja:

- `wczytaj_dane(...)` — pobiera dane i zapisuje do tablicy, zwraca liczbę elementów,
- `oblicz_sume(...)` — sumuje elementy,
- `oblicz_srednia(...)` — zwraca średnią,
- `znajdz_min_max(...)` — znajduje min i max,
- `bubble_sort(...)` — sortuje dane,
- `drukuj_wyniki(...)` — wypisuje wszystko na ekran.

3.3 Przykład – język C

```
#include <stdio.h>

#define MAX_N 1000

int wczytaj_dane(double dane[], int max_n);
double oblicz_sume(const double dane[], int n);
double oblicz_srednia(const double dane[], int n);
void znajdz_min_max(const double dane[], int n, double *min, double *max);
void bubble_sort(double dane[], int n);
void drukuj_wyniki(const double dane[], int n);

int main(void) {
    double dane[MAX_N];
    int n = wczytaj_dane(dane, MAX_N);
    if (n == 0) {
        printf("Brak danych.\n");
        return 0;
    }
```

```
    double suma = oblicz_sume(dane, n);
    double srednia = oblicz_srednia(dane, n);
    double min, max;
    znajdz_min_max(dane, n, &min, &max);

    bubble_sort(dane, n);
    drukuj_wyniki(dane, n);

    printf("Suma: %.2f, Średnia: %.2f, Min: %.2f, Max: %.2f\n",
           suma, srednia, min, max);
    return 0;
}
```

Każda funkcja powinna być krótka, mieć jasny cel i nie mieć efektów ubocznych poza tymi, które są jawne.

3.4 Przykład – Python

```
def wczytaj_dane():
    n = int(input("Podaj liczbę elementów: "))
    dane = []
    for i in range(n):
        x = float(input(f"Podaj liczbę {i+1}: "))
        dane.append(x)
    return dane

def oblicz_sume(dane):
    return sum(dane)

def oblicz_srednia(dane):
    return sum(dane)/len(dane) if dane else 0

def znajdz_min_max(dane):
    return (min(dane), max(dane)) if dane else (None, None)

def bubble_sort(dane):
    dane = dane[:]  # kopia
    while True:
        swapped = False
        for i in range(len(dane)-1):
            if dane[i] > dane[i+1]:
                dane[i], dane[i+1] = dane[i+1], dane[i]
                swapped = True
        if not swapped:
            break
    return dane

def drukuj_wyniki(dane):
    print("Posortowane dane:", dane)

def main():
    dane = wczytaj_dane()
    if not dane:
        print("Brak danych.")
        return
    suma = oblicz_sume(dane)
    srednia = oblicz_srednia(dane)
```

```
    min_val, max_val = znajdz_min_max(dane)
    posortowane = bubble_sort(dane)
    drukuj_wyniki(posortowane)
    print(f"Suma: {suma}, Średnia: {srednia}, Min: {min_val}, Max:
{max_val}")

if __name__ == "__main__":
    main()
```

---

4. Tutorial 2 – Strukturalne instrukcje sterujące

4.1 Instrukcje warunkowe

- `if`, `else if`, `else` — zagnieżdżone w logiczny sposób, bez „skoków”
- `switch` / `match` dla wyborów wielu przypadków

4.2 Pętle

- `for` — dla znanej liczby iteracji
- `while` — dla warunków nieznanej liczby iteracji
- unikanie „niekończących się” pętli

4.3 Zasada jednego wejścia i jednego wyjścia

Każdy blok (funkcja, pętla, warunek) powinien mieć **jedno miejsce wejścia i jedno wyjście**. Zwiększa to czytelność i poprawność formalną.

---

5. Zadania do samodzielnego wykonania

Zadanie 1 (obowiązkowe)

Przepisz program imperatywny z jednostki 1 do **proceduralnej wersji** z funkcjami. Podziel odpowiedzialności wg wzoru z tutorialu. Dodaj czytelne instrukcje warunkowe i pętle.

Zadanie 2 (obowiązkowe)

Rozszerz program o funkcję `znajdz_mediane(...)` i porównaj medianę z średnią dla różnych zestawów danych.

Zadanie 3 (dodatkowe)

Napisz program strukturalny rozwiązujący prosty problem:

Wczytaj zestaw ocen studentów (0–100), oblicz liczbę ocen powyżej średniej i wypisz je w kolejności malejącej.

---

6. Pytania kontrolne

1. Na czym polega główna różnica między paradygmatem imperatywnym a proceduralnym?
2. Co daje podział programu na funkcje?
3. Dlaczego unikanie `goto` zwiększa czytelność kodu?
4. Co to znaczy, że funkcja ma jedno wejście i jedno wyjście?
5. Jakie zalety ma program strukturalny w porównaniu z nieuporządkowanym przepływem?

---

7. Checklisty

Proceduralny

- Kod podzielony na funkcje.
- Każda funkcja ma jasno określoną odpowiedzialność.
- Parametry i wartości zwracane są poprawnie użyte.

Strukturalny

- Użyto `if`, `for`, `while` zamiast `goto`.
- Każdy blok ma jedno wejście i jedno wyjście.
- Kod jest wcięty i czytelny.

---

8. Wzór sprawozdania

Struktura jak w jednostce 1 + dodatkowo:

- **Schemat wywołań funkcji** (diagram lub opis).
- Porównanie wersji imperatywnej i proceduralnej.
- Komentarz o poprawie czytelności i konserwowalności.

---

9. Aneks – Szablony kodu

C – Deklaracje funkcji

```
int wczytaj_dane(double dane[], int max_n);
double oblicz_sume(const double dane[], int n);
double oblicz_srednia(const double dane[], int n);
void znajdz_min_max(const double dane[], int n, double *min, double *max);
void bubble_sort(double dane[], int n);
void drukuj_wyniki(const double dane[], int n);
```

Python – Szkielet

```
def wczytaj_dane():
    ...

def oblicz_sume(dane):
    ...

def oblicz_srednia(dane):
    ...

def znajdz_min_max(dane):
    ...

def bubble_sort(dane):
    ...

def drukuj_wyniki(dane):
    ...

def main():
    dane = wczytaj_dane()
    # ...
```

Zakończenie

Po wykonaniu ćwiczenia student powinien:

- rozumieć różnicę między imperatywnym, proceduralnym i strukturalnym podejściem,
- potrafić dekomponować problem na funkcje,
- stosować strukturalne instrukcje sterujące,
- przygotować dokumentację i analizę programu w stylu proceduralnym.