Programowanie I. Struktury sterujące: sekwencja i abstrakcja

Aleksander Denisiuk
Uniwersytet Warmińsko-Mazurski
Olsztyn, ul. Słoneczna 54
denisjuk@matman.uwm.edu.pl

7 lutego 2018

Struktury sterujące: sekwencja i abstrakcja

Najnowsza wersja tego dokumentu dostępna jest pod adresem

http://wmii.uwm.edu.pl/~denisjuk/uwm

Struktury sterujące

Podstawowe Struktury

- Sekwencja
- Abstrakcja
- Wybór
- Powtórzenie

Mus czekoladowy

Wziąć cztery duże albo pięć średnich jajek. Rozbić jajko i umieścić białko w szklance. Powtórzyć dla pozostałych jajek.
Ubić białka na pianę.
Dodać łyżeczkę cukru i dokładnie wymieszaj. Dodać drugą łyżeczkę i ponownie wymieszać. Powtarzać, aż się doda całość (50 gr).
Czekoladę z kawą rozpuścić w kąpieli wodnej, dodać masło i wymieszać, aby składniki się połączyły.
Przestudzoną (to ważne!) czekoladę delikatnie wymieszać z żółtkami.
Przełożyć do czego kto lubi i schłodzić.

Przepis jako algorytm

1. Jeżeli jajka są duże, wziąć 4 sztuki. Inaczej wziąć 5.
2. Rozbić jajko i umieścić białko w szklance.
3. Powtórzyć dla pozostałych jajek.
4. Ubić białka na pianę.
5. Dodać łyżeczkę cukru i dokładnie wymieszaj.
6. Dodać drugą łyżeczkę i ponownie wymieszać.
7. Powtarzać, aż się doda całość (50 gr).
8. Czekoladę z kawą rozpuścić w kąpieli wodnej, dodać masło i wymieszać, aby składniki się połączyły.
9. Przestudzoną (to ważne!) czekoladę delikatnie wymieszać z żółtkami.
10. Przełożyć do czego kto lubi i schłodzić.

Sekwencja

Sekewncja

- Ciąg operatorów, wykonywanych jeden po drugim
- Początek kolejnego operatora zgadza się z końcem poprzedniego
- W C operatory są oddzielane średnikiem

Abstrakcja

Abstrakcja w algorytmach

- Rozbić skomplikowany algorytm na mniejsze algorytmy *szczegółowe*

Przykład. Wyświetlić ludzika

Pierwszy krok projektowania


```
/* Wyświetlić głowę */
/* Wyświetlić ręce */
/* Wyświetlić tułów */
/* Wyświetlić nogi */
/* Wyświetlić stopy */
```

Drugi krok


```
/* Wyświetlić głowę */
   printf("          *****\n");
   printf("          *   *\n");
   printf("          *****\n");
/* Wyświetlić ręce */
   printf("      *       *       *\n");
   printf("       **     *      **\n");
   printf("         ********\n");
/* Wyświetlić tułów */
   printf("            ***\n");
   printf("            ***\n");
   printf("            ***\n");
   printf("            ***\n");
/* Wyświetlić nogi */
/* Wyświetlić stopy */
```

Drugi krok z wykorzystaniem abstrakcji


```
/* Wyświetlić głowę */
   PrintHead();
/* Wyświetlić ręce */
   PrintArms();
/* Wyświetlić tułów */
   PrintBody();
/* Wyświetlić nogi */
   PrintLegs();
/* Wyświetlić stopy */
   PrintFeet();
```

Algorytmy szczegółowe


```
void PrintHead(void){
    printf("          *****\n");
    printf("          *   *\n");
    printf("          *****\n");
}

void PrintArms(void){
    printf("      *       *       *\n");
    printf("       **     *     **\n");
    printf("         *******\n");
}

void PrintBody(void){
    printf("            ***\n");
    printf("            ***\n");
    printf("            ***\n");
    printf("            ***\n");
}
```

Wykorzystane abstrakcji

- Zmniejszyć skomplikowaność algorytmu głównego
- Długość algorytmów szczegółowych zależy od długości algorytmu głównego
- Zazwyczaj długość algorytmu szczegółowego nie przekracza 50–60 linijek
- Jeżeli nietrywialne zagadnienie jest rozwiązywane w kilka miejscach, zrobić algorytm szczegółowy

Procedury

Procedury bez parametrów

- Kod algorytmu szczegółowego
- Funkcje typu **void**

- Wywołanie procedury: `NazwaProcedury();`
- Przykład

```
void PrintHead(void){
    printf("          *****\n");
    printf("          *   *\n");
    printf("          *****\n");
}

int main(void){
    PrintHead();
    return 0;
}
```

Wcześniejsza deklaracja

- Procedura powinna zostać opisana zanim będzie wykorzystywana
- Wcześniejsza deklaracja

- Przykład

```
void PrintHead(void);

int main(void){
    PrintHead();
    return 0;
}
```

- Implementacja później

Moduły (Biblioteki)

- Plik nagłówkowy (`*.h`)
- Plik z implementacja (`*.c`)
- Ta sama nazwa
- Wykorzystanie modułów standardowych:

```
#include <module.h>
```

- Wykorzystanie modułów własnych:

```
#include "module.h"
```

Ludzik, `painter.h`


```
#ifndef PAINTER_H
#define PAINTER_H

void PrintHead(void);
void PrintArms(void);
void PrintBody(void);
void PrintLegs(void);
void PrintFeet(void);

#endif // PAINTER_H
```

Ludzik, painter.c


```c
#include <stdio.h>

#include "painter.h"

void PrintHead(void){
   printf("          *****\n");
   printf("          *   *\n");
   printf("          *****\n");
}

void PrintArms(void){
   printf("      *       *       *\n");
   printf("        **    *     **\n");
   printf("          ********\n");
}
```

Ludzik, `painter.c`, cd


```
void PrintBody(void){
    printf("            ***\n");
    printf("            ***\n");
    printf("            ***\n");
    printf("            ***\n");
}
void PrintLegs(void){
    printf("             *    *\n");
    printf("             *    *\n");
    printf("           *       *\n");
    printf("           *       *\n");
}
void PrintFeet(void){
    printf("         ***      ***\n");
    printf("         ***      ***\n");
    printf("         ***      ***\n");
}
```

Ludzik, figure.c


```c
#include "painter.h"

int main(void){
    PrintHead();
    PrintArms();
    PrintBody();
    PrintLegs();
    PrintFeet();
    return 0;
}
```

Kompilacja


```
gcc -Wall -g figure.c painter.c -o figure
```

Makefile


```
CC=gcc
CFLAGS=-Wall -g
SOURCES = $(wildcard  *.c)
EXECUTABLE = figure

$(EXECUTABLE): $(SOURCES)
        $(CC) $^ -o $@

clean:
        rm -f $(EXECUTABLE) $(EXECUTABLE).exe
```

- `$^` — wszystkie zależności
- `$@` — cel

Uwagi

- Nazwa procedury powinna być sformatowana w stylu Pascala
- Nazwa powinna podpowiadać, co procedura robi
- Przy opisaniu procedur podaje się dwa warunki:
  1. Założenia wstępne — powinny być spełnione przed procedurą (może nie być)
  2. Wynik — warunek spełniony po procedurze
- Przykład:

```
void PrintHead(void){
/// Wynik: została wyświetlona głowa
   printf("          *****\n");
   printf("          *   *\n");
   printf("          *****\n");
}
```

Case Study

Raport tygodniowy

Opisanie

Trzyosobowa firma w odpowiedzi na wymagania urzędu skarbowego potrzebuje programu, który by wygenerował tygodniowy raport, zawierający: stanowisko pracownika, inicjały, pesel, ilość wypracowanych godzin oraz tygodniowe wynagrodzenie.

Dane wejściowe

Informacja wejściowa składa się z trzech wierszy:
w pierwszym podaje się informacja o prezesie firmy,
w drugim — o wiceprezesie, w trzecim — o księgowym.
Każdy wierz ma następujący format:

1. Cztery symbole na inicjały (pierwsza litera imienia, kropka, pierwsza litera nazwiska, kropka).
2. Jedna lub więcej spacji
3. Cztery ostatnie cyfry pesela.
4. Jedna lub więcej spacji
5. Liczba naturalna — stawka godzinowa w złotych.
6. Jedna lub więcej spacji
7. Liczba naturalna — ilość przepracowanych godzin.
8. Jedna lub więcej spacji

Dane wyjściowe

- Przed trzema wierszami wejściowymi wyświetla się informacja o następującej treści:

```
Podaj informację o prezesie, wiceprezesie
i księgowym w trzech wierszach. Każdy wiersz
powinien zawierać inicjały (cztery symbole),
cztery ostatnie cyfry pesela, stawkę godzinową
w złotych oraz ilość przepracowanych godzin.
Po każdym wierszu zostanie wyświetlony raport.
```

- Po tej informacji wyświetla się pusta linia, a potem trzy zachęty do wprowadzenia danych:

  `Podaj informację o <stanowisko>`

  gdzie <stanowisko> zamienia się na `prezesie`, `wiceprezesie`, `księgowym`

Dane wyjściowe, cd

- Po każdym wejściowym wierszu wyświetla się pusta linia, po czym raport w następującym formacie

```
inicjały:                    <inicjały>
pesel:                       <pesel>
stawka godzinowa:            <stawka godzinowa>
przepracowanych godzin:      <godzin>
zarobek tygodzniowy:         <zarobek tygodniowy>
<pusta linia>
<pusta linia>
```

Błędne Dane

Przy naruszeniu formatu wejściowych danych zachowanie niezdefiniowane

Przykład

Podaj informację o prezesie, wiceprezesie i księgowym w trzech wierszach. Każdy wiersz powinien zawierać inicjały (cztery symbole), cztery ostatnie cyfry pesela, stawkę godzinową w złotych oraz ilość przepracowanych godzin. Po każdym wierszu zostanie wyświetlony raport.

```
Podaj informację o prezesie
R.R. 7234 123 21

inicjały:                  R.R.
pesel:                     7234
stawka godzinowa:          123
przepracowanych godzin:    21
zarobek tygodzniowy:       2583
```

Przykład, cd


```
Podaj informację o wiceprezesie
G.B. 2348 27 35

inicjały:                  G.B.
pesel:                     2348
stawka godzinowa:          35
przepracowanych godzin:    27
zarobek tygodzniowy:       945

Podaj informację o księgowym
H.B. 3896 20 10

inicjały:                  H.B.
pesel:                     2348
stawka godzinowa:          20
przepracowanych godzin:    10
zarobek tygodzniowy:       200
```

Pierwszy krok projektowania


```
/* Wyświetlenie informacji wstępnej */

/* Wprowadzenie i opracowanie
   informacji o prezesie */

/* Wprowadzenie i opracowanie
   informacji o wicprezesie */

/* Wprowadzenie i opracowanie
   informacji o księgowym */
```

Drugi krok projektowania


```
printf("Podaj informację o prezesie, wiceprezesie\n");
printf("i księgowym w trzech wierszach. Każdy wiersz\n");
printf("powinien zawierać inicjały (cztery symbole),\n");
printf("cztery ostatnie cyfry pesela, stawkę godzinową\n");
printf("w złotych oraz ilość przepracowanych godzin.\n");
printf("Po każdym wierszu zostanie wyświetlony raport.");

/* Wprowadzenie i opracowanie informacji o prezesie */
   printf("Podaj informację o prezesie\n");
   ProcessPerson();

/* Wprowadzenie i opracowanie informacji o wicprezesie */
   printf("Podaj informację o wiceprezesie\n");
   ProcessPerson();

/* Wprowadzenie i opracowanie informacji o księgowym */
   printf("Podaj informację o księgowym\n");
   ProcessPerson();
```

Procedura `ProcessPerson`. Zmienne


```
void ProcessPerson(void){
    char first_initial;
    char dot;
    char second_initial;
    char pesel1, pesel2, pesel3, pesel4;
    unsigned int wage_per_hour; // stawka
    unsigned int hours;
    unsigned int weekly_wage;
```

Wczytywanie danych


```
scanf(" %c%c%c%c %c%c%c%c %u %u",
      &first_initial,
      &dot,
      &second_initial,
      &dot,
      &pesel1,
      &pesel2,
      &pesel3,
      &pesel4,
      &wage_per_hour,
      &hours
      );
```

Obliczenie i wyświetlanie


```
weekly_wage = wage_per_hour * hours;

printf("inicjały:\t\t%c.%c.\n", first_initial,
                                second_initial);
printf("pesel:\t\t\t%c%c%c%c\n", pesel1, pesel2,
                                pesel3, pesel4);
printf("stawka godzinowa:\t%u\n",
                                wage_per_hour);
printf("przepracowanych godzin:\t%u\n", hours);
printf("zarobek tygodzniowy:\t%u\n\n\n",
                                weekly_wage);
```

- Łamanie wierszy tylko na potrzeby prezentacji!

Zmienne lokalne

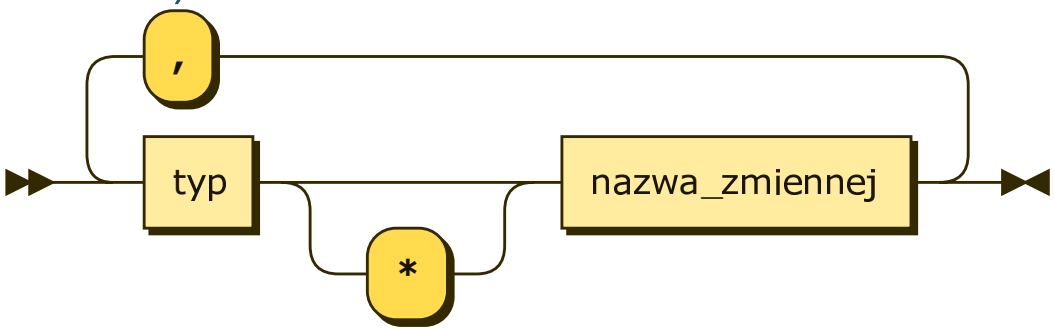
- Zmienne określone wewnątrz procedury (lokalne) nie są dostępne poza procedurą
- Zmienne globalne mogą zostać zmienione wszędzie
- Unikać globalnych zmiennych
- Możliwe użycie:
  - ✦ globalne stałe
  - ✦ skomplikowane unikatowe obiekty, używane w wielu procedurach

Parametry

Procedury z parametrami

- Deklaracja procedury z parametrami:

- Opisanie parametrów (typ, sposób przekazywania, nazwa):



- Przykład:

```
void ReadData(int number, double * score);
```

Przez wartość

- Bez gwiazdki
- W procedurze tworzy się zmienna lokalna
- Przy wywołaniu można podać zmienna, stałą, wyrażenie

```
void PrintSquare(int number){
    printf("%d\n", number*number);
}
int main(void){
    int n = 0;
    PrintSquare(0);
    PrintSquare(n);
    PrintSquare(sqrt(n++)+n); // Niezdefiniowan
    return 0;
}
```

Przez wskaźnik (referencję)

- Z gwiazdką
- Przy wywołaniu podaje się zmienna (poprzedzona znakiem `&` (bądź wskaźnik))
- Operuje się na tej zmiennej (odwołanie się poprzez parameter z gwiazdką)
- Zmiany są przekazywane na zewnątrz

```
void Sqr(int * number){
    *number = (*number) * (*number);
}

int main(void){
    int n = 2;
    Sqr(&n);
    printf("%u\n", n);
    return 0;
}
```

Modyfikator `const`

- Argument, który nie jest zmieniany w procedurze.
- Weryfikacja podczas kompilacji
- W szczególności, duże struktury danych

```
void Output(const VeryBigType * big_data);
```

Case Study

Darniowanie i ogrodzenie

Opisanie

Firma zajmuje się darniowaniem i ogrodzeniem działek. Jeden zespół obsługuje dwie działki tygodniowo (układa darń i stawi płot). Program pyta o wymiary dwóch działek i drukuje ogólny i średni obwód oraz pole powierzchni dwóch działek. Program zostanie używany do zamówienia materiałów.

Dane Wejściowe

Użytkownik dwa razy jest proszony o wprowadzenie dwóch liczb. Każda para wprowadzanych liczb — to długość i szerokość działki.

Dane wyjściowe

- Przed każdą parą liczb wyświetla się podpowiedź
  `Podaj długość i szerokość działki`
- Po wprowadzonej informacji wyświetla się pusta linia, po której następujące cztery wiersze:

```
Ogólna długość ogrodzenia: <OD>
Ogólna powierzchnia do układania darni: <OP>
Średnia długość ogrodzenia: <ŚD>
Średnia powierzchnia do układania darni: <ŚP>
```

- gdzie `<OD>`, `<OP>`, `<ŚD>` i `<ŚP>` są liczbami całkowitymi. Przy obliczeniu `<ŚD>` oraz `<ŚP>` część ułamkowa jest odrzucana.

Błędne Dane

Przy naruszeniu formatu wejściowych danych zachowanie niezdefiniowane

Przykład


```
Podaj długość i szerokość działki
50 100
Podaj długość i szerokość działki
101 99

Ogólna długość ogrodzenia: 700
Ogólna powierzchnia do układania darni: 1499
Średnia długość ogrodzenia: 350
Średnia powierzchnia do układania darni: 7499
```

Pierwszy krok projektowania


```
void GetAreaPerimeter(int *area, int *perimeter);
/// Wynik: zostały wprowadzone rozmiary działki
/// area jest polem powierzchni,
/// perimeter -- obwodem

int total_area = 0;
int total_perimeter = 0;

GetAreaPerimeter(&area, &perimeter);
/* Aktuallizować total_area i total_perimeter */
GetAreaPerimeter(&area, &perimeter);
/* Aktuallizować total_area i total_perimeter */
/* Wydruk wyników */
```

Drugi etap


```
int total_area = 0;
int total_perimeter = 0;
int area, perimeter; //Pole i obwód działki

GetAreaPerimeter(&area, &perimeter);
/* Aktuallizować total_area i total_perimeter */
   total_area += area;
   total_perimeter += perimeter;
GetAreaPerimeter(&area, &perimeter);
/* Aktuallizować total_area i total_perimeter */
   total_area += area;
   total_perimeter += perimeter;
/* Stwierdzenie: total_area jest
   sumaryczną powierzchnią, total_perimeter
   jest sumarycznym obwodem   */
```

Wydruk wyników


```
/* Wydruk wyników */
printf("\n");
printf("Ogólna długość ogrodzenia: %d\n",
       total_perimeter);
printf("Ogólna powierzchnia do układania \
       darni: %d\n", total_area);
printf("Średnia długość ogrodzenia: %d\n",
       total_perimeter/2);
printf("Średnia powierzchnia do układania \
       darni: %d\n", total_area/2);
```

Procedura GetAreaPerimeter


```
void GetAreaPerimeter(int *area, int *perimeter)
/// Wynik: zostały wprowadzone rozmiary działki:
/// długość i szerokość.
/// area jest polem powierzchni,
/// perimeter -- obwodem

    int width, height;

    printf("Podaj długość i szerokość działki\n");
    scanf("%d %d", &width, &height);
    *area = width*height;
    *perimeter = 2*(width + height);
}
```

Testowanie

Zestaw danych testowych

- Tworzy się plik danych wejściowych `test.input` oraz odpowiedni plik danych wyjściowych `test.output`
- Porównuje się wyjście programu na danych testowych z zawartością pliku `test.output`
- Wczytywanie z pliku:

```
./program < test.input
```

- Przekierowanie wyścia do pliku

```
./program < test.input > output.test
```

- Porównywanie dwóch plików

```
diff -y output.test test.output
```

Ogrodzenie i darniowanie

- `test.input`:

```
50 100
101 99
```

test.output


```
Podaj długość i szerokość działki
Podaj długość i szerokość działki

Ogólna długość ogrodzenia: 700
Ogólna powierzchnia do układania darni: 14999
Średnia długość ogrodzenia: 350
Średnia powierzchnia do układania darni: 7499
```

Wynik pozytywny


```
diff -y output.test test.output
```

```
Podaj długość i szerokość działki                 Podaj długość i szerokoś
Podaj długość i szerokość działki                 Podaj długość i szerokoś

Ogólna długość ogrodzenia: 700                    Ogólna długość ogrodzenia
Ogólna powierzchnia do układania darni: 14999     Ogólna powierzchnia do uk
Średnia długość ogrodzenia: 350                   Średnia długość ogrodzeni
Średnia powierzchnia do układania darni: 7499     Średnia powierzchnia do u
```

Brak pustej linii


```
diff -y output.test test.output
```

```
Podaj długość i szerokość działki                    Podaj długość i szerokoś
Podaj długość i szerokość działki                    Podaj długość i szerokoś
                                                   >
Ogólna długość ogrodzenia: 700                       Ogólna długość ogrodzenia
Ogólna powierzchnia do układania darni: 14999        Ogólna powierzchnia do uk
Średnia długość ogrodzenia: 350                      Średnia długość ogrodzeni
Średnia powierzchnia do układania darni: 7499        Średnia powierzchnia do u
```

Zła odpowiedź


```
diff -y output.test test.output
```

```
Podaj długość i szerokość działki                    Podaj długość i szerokoś
Podaj długość i szerokość działki                    Podaj długość i szerokoś

Ogólna długość ogrodzenia: 700                       Ogólna długość ogrodzenia
Ogólna powierzchnia do układania darni: 15000      | Ogólna powierzchnia do uk
Średnia długość ogrodzenia: 350                      Średnia długość ogrodzeni
Średnia powierzchnia do układania darni: 7499        Średnia powierzchnia do u
```