Wstęp do programowania

Wstęp

Paweł Daniluk

Wydział Fizyki

Jesień 2013

Plan wykładu

1. Wstęp – komputery, algorytmy i programy
2. Podstawowe konstrukcje programistyczne
3. Tablice
4. Operacje na plikach, przetwarzanie napisów, wyrażenia regularne
5. Rekurencja
6. Złożoność obliczeniowa, dowodzenie poprawności programu
7. Struktury danych
8. Programowanie dynamiczne
9. Strategia dziel i zwyciężaj
10. Algorytmy zachłanne
11. Klasy złożoności obliczeniowej

$$
\begin{aligned}
a_1 &= 1 \\
a_2 &= 1 + 2 \\
a_3 &= 1 + 2 + 3 \\
&\dots \\
a_i &= 1 + 2 + 3 + \dots + i
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= 1 \\
a_2 &= 1 + 2 \\
a_3 &= 1 + 2 + 3 \\
&\dots \\
a_i &= 1 + 2 + 3 + \dots + i
\end{aligned}
$$

$$
a_i = \sum_{k=1}^{i} i
$$

$$a_1 = 1$$
$$a_2 = 1 + 2$$
$$a_3 = 1 + 2 + 3$$
$$\dots$$
$$a_i = 1 + 2 + 3 + \dots + i$$

$$a_i = \sum_{k=1}^{i} i$$

Definicja rekurencyjna

$$a_0 = 0$$
$$a_i = a_{i-1} + i$$

Liczby Fibonacci'ego

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

Liczby Fibonacci'ego

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

$$F_0 = 1$$
$$F_1 = 1$$
$$F_{n+1} = F_{n-1} + F_n$$

Liczby Fibonacci'ego

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

$$F_0 = 1$$
$$F_1 = 1$$
$$F_{n+1} = F_{n-1} + F_n$$

Definicja zwarta

$$F_n = \frac{\phi^n - \psi^n}{\phi - \psi}$$

gdzie:

$$\phi = \frac{1+\sqrt{5}}{2}, \psi = \frac{1-\sqrt{5}}{2}$$

Algorytm

Sposób postępowania prowadzący do pożądanego efektu.
Przykłady:

1. Przepisy kucharskie
2. Zapis nutowy muzyki
3. Instrukcje montażu
4. Opis dojazdu

Instrukcja szamponu

1. Nałożyć szampon
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1

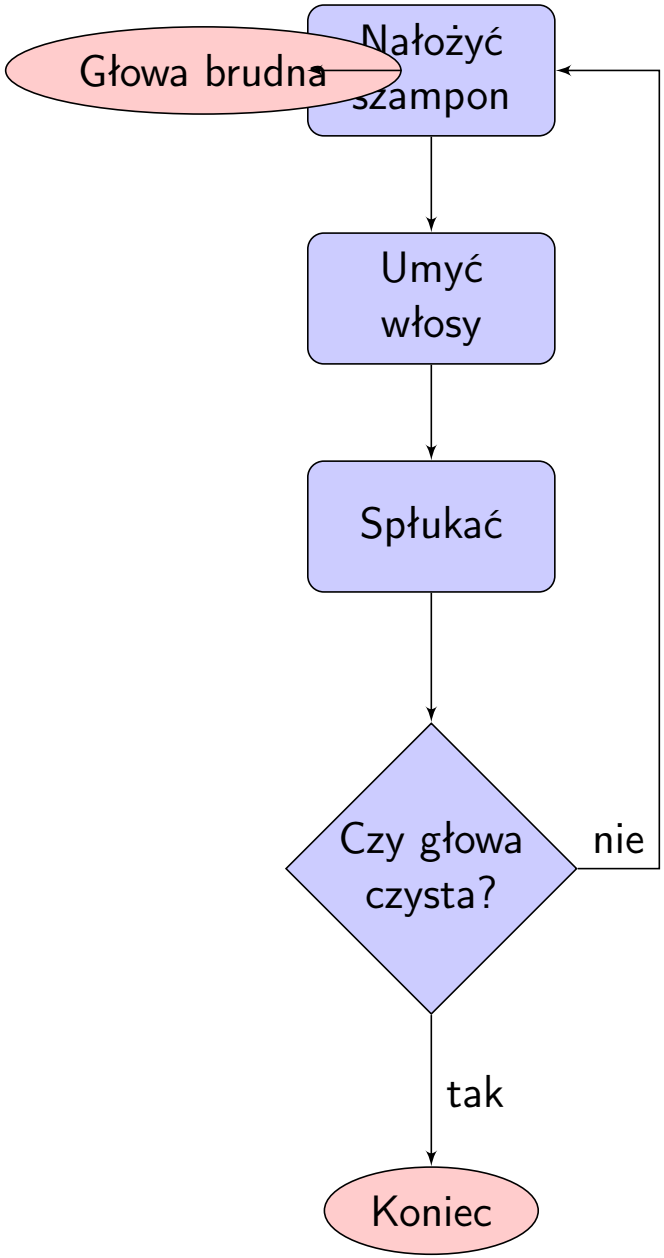
Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1
    1. Nałożyć szampon
    2. Umyć włosy
    3. Spłukać
    4. Czynność powtórzyć

Instrukcja szamponu

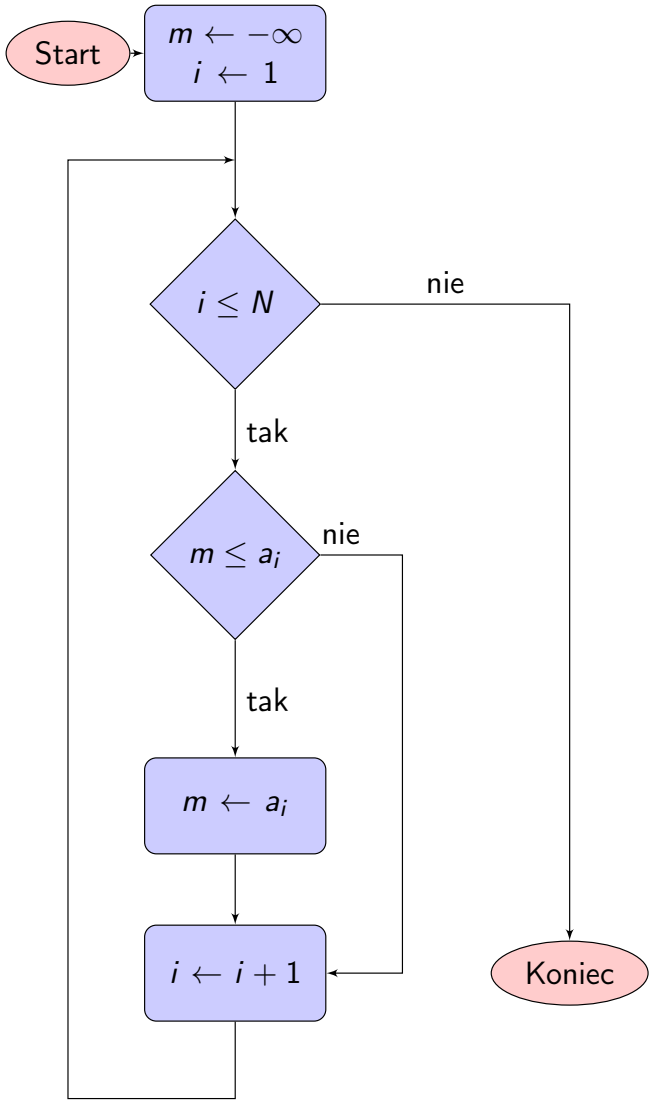
1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1
    1. Nałożyć szampon
    2. Umyć włosy
    3. Spłukać
    4. Czynność powtórzyć
        1. Nałożyć szampon
        2. Umyć włosy
        3. Spłukać
        4. Czynność powtórzyć

Schemat blokowy



Zadanie

Znaleźć największy wyraz ciągu $a_1, \ldots, a_N$.



Disclaimer

Informatyka jest nauką o komputerach w podobnym stopniu, jak astronomia jest nauką o teleskopach.

Jak działa komputer

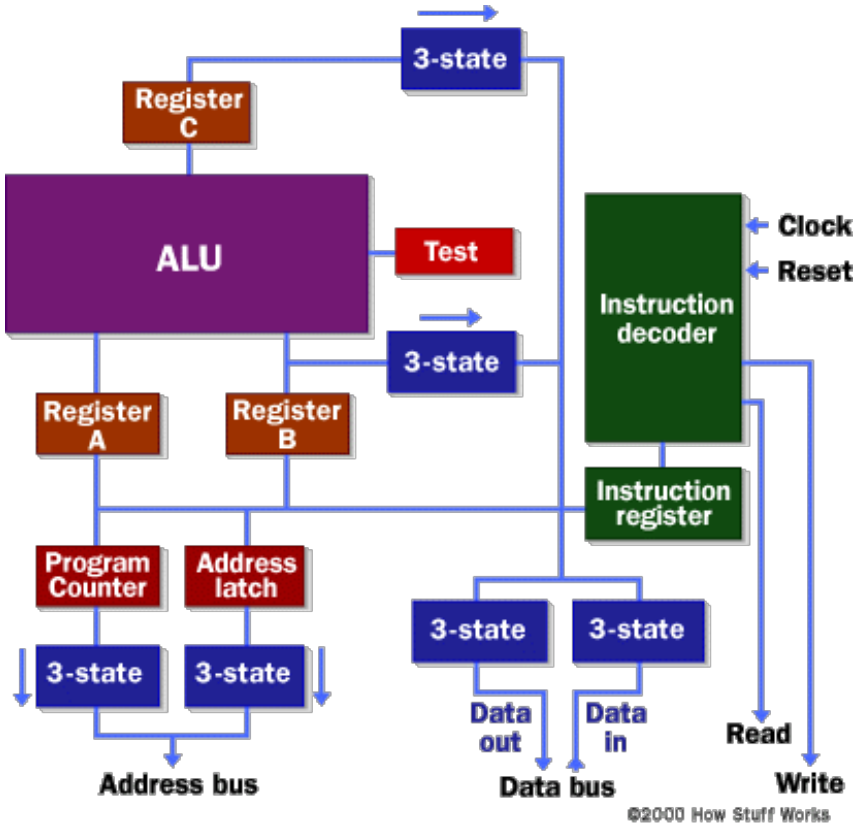
Procesor – Wczytuje rozkazy z pamięci i je wykonuje.

Pamięć – Przechowuje programy i dane.

UWAGA: Programy i dane są traktowane tak samo

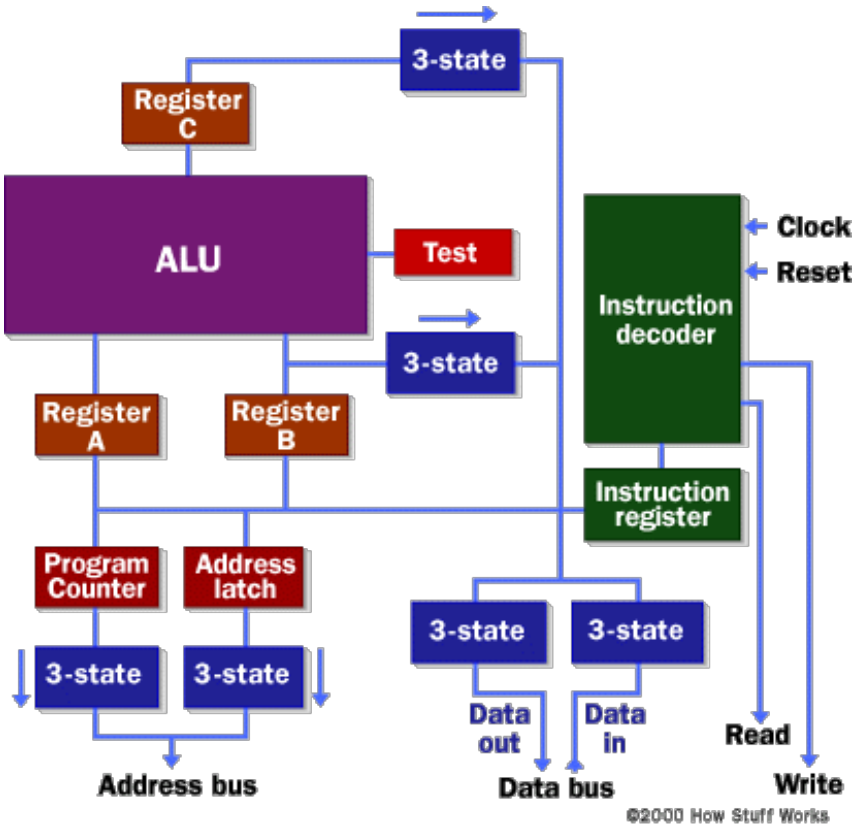
Magistrala – Łączy procesor, pamięć i inne elementy komputera.

Urządzenia wejścia/wyjścia – Procesor komunikuje się z nimi za pośrednictwem magistrali i dodatkowych interfejsów



Jak działa komputer

- ALU – jednostka arytmetyczno-logiczna
- Rejestry – A, B, C
- Rejestry specjalne – wskaźnik instrukcji, adresowy, instrukcji
- Przełączniki 3-stanowe
- Połączenia pomiędzy dekoderem instrukcji, a ALU, rejestrami i przełącznikami (nieuwidocznione)
- Magistrala – Szyna adresowa, danych i sterująca



Rozkazy

LOADA mem	Load register A from memory address
LOADB mem	Load register B from memory address
CONB con	Load a constant value into register B
SAVEB mem	Save register B to memory address
SAVEC mem	Save register C to memory address
ADD	Add A and B and store the result in C
SUB	Subtract A and B and store the result in C
MUL	Multiply A and B and store the result in C
DIV	Divide A and B and store the result in C
COM	Compare A and B and store the result in test
JUMP addr	Jump to an address
JEQ addr	Jump, if equal, to address
JNEQ addr	Jump, if not equal, to address
JG addr	Jump, if greater than, to address
JGE addr	Jump, if greater than or equal, to address
JL addr	Jump, if less than, to address
JLE addr	Jump, if less than or equal, to address
STOP	Stop execution

```
// Assume a is at address 128
// Assume F is at address 129
0 CONB 1 // a=1;
1 SAVEB 128
2 CONB 1 // f=1;
3 SAVEB 129
4 LOADA 128 // if a > 5 then jump to 17
5 CONB 5
6 COM
7 JG 17
8 LOADA 129 // f=f*a;
9 LOADB 128
10 MUL
11 SAVEC 129
12 LOADA 128 // a=a+1;
13 CONB 1
14 ADD
15 SAVEC 128
16 JUMP 4 // loop back to if
17 STOP
```

Opkody

LOADA	1
LOADB	2
CONB	3
SAVEB	4
SAVEC	5
ADD	6
SUB	7
MUL	8
DIV	9
COM	10
JUMP	11
JEQ	12
JNEQ	13
JG	14
JGE	15
JL	16
JLE	17
STOP	18

Kod maszynowy

0	3	CONB 1
1	1	
2	4	SAVEB 128
3	128	
4	3	CONB 1
5	1	
6	4	SAVEB 129
7	129	
8	1	LOADA 128
9	128	
10	3	CONB 5
11	5	
12	10	COM
13	14	JG 17
14	31	
15	1	LOADA 129
16	129	

17	2	LOADB 128
18	128	
19	8	MUL
20	5	SAVEC 129
21	129	
22	1	LOADA 128
23	128	
24	3	CONB 1
25	1	
26	6	ADD
27	5	SAVEC 128
28	128	
29	11	JUMP 4
30	8	
31	18	STOP

Paradygmaty programowania

Programowanie imperatywne

Algol

```
a:=1;
f:=1;
while a <= 5 do
    begin
        f := f * a;
        a := a + 1;
    end;
```

Paradygmaty programowania

Programowanie funkcyjne

Brak skutków ubocznych, stan się nie zmienia.

SML

```
let fun fac 0 = 1
      | fac n = n * fac (n - 1)
in
    fac(5)
end;
```

Paradygmaty programowania

Programowanie deklaratywne

Programista opisuje pożądany wynik bez precyzowania algorytmu.

Prolog

```
factorial(0,1).
factorial(N,F) :- N>0, N1 is N-1,
                  factorial(N1,F1), F is N * F1.

?- factorial(3,W).

W = 6 ;
```

Paradygmaty programowania

Programowanie obiektowe

Obiekty łączą w sobie dane (stan) i zachowanie (metody).

Smalltalk

```
factorial
    "Answer the factorial of the receiver."

    self = 0 ifTrue: [^ 1].
    self > 0 ifTrue: [^ self * (self - 1) factorial].
    self error: 'Not valid for negative integers'


5 factorial
```

Język programowania

1. Składnia – słowa i gramatyka
2. Semantyka – znaczenie
3. Typy danych
4. Biblioteki standardowe

Kompilować, czy interpretować

Języki kompilowane

Kod źródłowy jest kompilowany do maszynowego.

- szybkie
- podatne na błędy
- trudno poprawiać
- zależne od platformy

Kompilować, czy interpretować

Języki kompilowane

Kod źródłowy jest kompilowany do maszynowego.

- szybkie
- podatne na błędy
- trudno poprawiać
- zależne od platformy

Języki interpretowane

Interpreter analizuje kod źródłowy w miarę wykonywania.

- mało wydajne
- wygodne
- niezależne od platformy

Zadanie 1

Zapisz wzór rekurencyjny oraz w postaci zwartej dla:

1. sum kwadratów kolejnych liczb naturalnych,
2. sum kolejnych nieparzystych liczb naturalnych,
3. sum kolejnych potęg 2,
4. sumy iloczynów par liczb naturalnych mniejszych od n (dla $n > 1$).

Zadanie 2

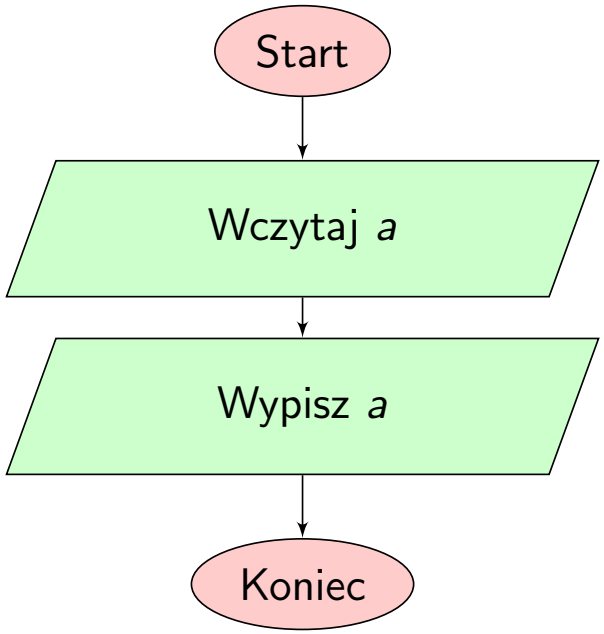
Wykonaj schemat blokowy algorytmu, który:

0. Wczytuje liczbę, a następnie ją wypisuje.

Zadanie 2

Wykonaj schemat blokowy algorytmu, który:

0. Wczytuje liczbę, a następnie ją wypisuje.



Zadanie 2 c.d.

Wykonaj schemat blokowy algorytmu, który:

1. Wczytuje dwie liczby i oblicza ich średnią arytmetyczną.
2. Wczytuje liczbę i sprawdza, czy jest dodatnia - algorytm powinien zwrócić odpowiedź TAK albo NIE.
3. Wczytuje dwie liczby i wyznacza najmniejszą z nich.
4. Wczytuje trzy liczby i wyznacza największą z nich.
5. Wczytuje dwie liczby a, n ($n \in N$) i oblicza a^n.
6. Wczytuje dwie liczby a, n ($n \in Z$) i oblicza a^n.
7. Wczytuje trzy liczby: a, b, c ($a, b, c \in R$) i wyznacza rozwiązania równania kwadratowego $ax^2 + bx + c = 0$. Wynikiem działania algorytmu powinna być w kolejności: liczba rozwiązań $(0, 1, 2)$, pierwsze rozwiązanie (gdy liczba rozwiązań > 0), drugie rozwiązanie (gdy liczba rozwiązań > 1).

Zadanie 3

Opracuj schemat blokowy programu, który zamienia dwie wartości miejscami.

Napisz program w kodzie maszynowym, który implementuje ten algorytm.

Zadanie 4

Przeanalizuj działanie przedstawionego na wykładzie programu w kodzie maszynowym.

Zastanów się, w jaki sposób można odczytać zawartość komórki pamięci o dowolnym (obliczonym w programie) adresie.

Zadanie 5

Opracuj schemat blokowy programu, który w szuka maksymalnego wyrazu w zadanym ciągu liczb $(a_{1\ldots N})$.

Napisz program w kodzie maszynowym, który implementuje ten algorytm.