Politechnika Wrocławska

Informatyka I — ARM031007
Wprowadzenie do informatyki — MCM032102

wer. 15 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2022-03-03 20:21:16 +0100

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Część I

Wprowadzenie

Politechnika Wrocławska

Program zajęć I

1. Wprowadzenie. Algorytm. Schematy blokowe. Idea programowania strukturalnego.
2. Struktura programów w C. Identyfikator, typy danych (typy fundamentalne: całkowite, rzeczywiste, znakowe, logiczny), deklaracja i inicjalizacja zmiennych, definiowanie stałych. Komunikacja poprzez konsolę. Operatory: arytmetyczne, logiczne, inkrementacji, dekrementacji, przypisania. Obliczanie wartości wyrażeń.
3. Struktury sterowania obliczeniami: rozgałęzienia i skoki, pętle pojedyncze i zagnieżdżone. Instrukcje proste i złożone; instrukcje warunkowe, wyrażenia warunkowe, instrukcje iteracyjne.
4. Preprocesor: dyrektywy, makrodefinicje.
5. Funkcje: budowa funkcji, argumenty funkcji, wynik wykonania funkcji, definicje i deklaracje globalne, argumenty funkcji main, rekurencja.

Politechnika Wrocławska

Program zajęć II

6. Tablice (tablice jedno i wielowymiarowe), łańcuchy znaków.
7. Wskaźniki. Pamięć dynamiczna.
8. **Kolokwium.**
9. Struktury danych, unie: deklaracja struktury, definiowanie zmiennej strukturalnej, tablice struktur, wskaźniki a struktury danych.
10. Operacje na plikach: otwieranie, zamykanie plików, czytanie i zapisywanie do plików.
11. Formatowanie w operacjach wejście/wyjście. Binarne wejście/wyjście.
12. Operacje na łańcuchach znaków.
13. Programowanie strukturalne w praktyce: podział programu na moduły, struktury danych, kompilacja.
14. Programy pomocnicze: diff, make, systemy rcs i cvs, debugger. Zarządzanie wersjami. Środowiska zintegrowane.
15. **Kolokwium zaliczeniowe.**

Politechnika Wrocławska

Po co uczyć się programowania? I

Pozwolę sobie przytoczyć tu 12 przewidywań na temat przyszłości programowania [**?**].

1. Procesory graficzne (GPU) będą naszymi następnymi procesorami.
2. Bardzo wiele przyszłego programowania dotyczyć będzie baz danych (*big data*).
3. JavaSript do wszystkiego.
4. Android na każdym urządzeniu.
5. Internet rzeczy — kolejne nowe platformy.
6. Open source na wiele sposobów.
7. Oparte na platformie WordPress aplikacje webowe.
8. Programy zostaną zastąpione przez „wtyczki” (plug-ins).
9. Niech żyją polecenia wydawane w terminalu!

Politechnika Wrocławska

Po co uczyć się programowania? II

10. Nie liczmy na dalsze upraszczanie językow programowania.
11. Programowaniem zajmować będą się programiści z krajów o najniższym koszcie pracy.
12. W dalszym ciagu szefostwo nie będzie rozumieć o co chodzi z tym programowaniem.

Politechnika Wrocławska

Slajdy

- Robotyka i Automatyzacja Procesów:
  https://myszka.kmim.wm.pwr.edu.pl/dydaktyka/c/rap/
- Mechatronika:
  https://myszka.kmim.wm.pwr.edu.pl/dydaktyka/c/mtr/

Bryk

Na podstawie slajdów powstał Bryk. Celowo nie nazywam go skryptem czy podręcznikiem. Można go znaleźć pod adresem: https://myszka.kmim.wm.pwr.edu.pl/dydaktyka/c/mtr/#bryk. Odsyłacz jest również na stronie wykładu.

Literatura I

- *A GNU Manual*, rozdział Nested Functions.
  Free Software Foundation, Inc., 2015.
- Programowanie w języku C, 2007.
  Wersja elektroniczna dostępna pod adresem:
  http://pl.wikibooks.org/wiki/Programowanie:C.
- David Griffiths, Dawn Griffiths.
  Head First C.
  Head First. O'Reilly, 2011.
  Są szanse, żeby ksiazka byłą dostępna pod tym adresem:
  http://proquestcombo.safaribooksonline.com/9781449335649.
  Można się o nią „dobijać" zgodnie z instrukcją na stronie biblioteki.

Politechnika Wrocławska

Literatura II

David Griffiths, Dawn Griffiths.
C. Rusz głową!
Helion, Gliwice, 2013.
Strona księgarni: http://helion.pl/ksiazki/c-rusz-glowa-david-griffiths-dawn-griffiths,cruszg.htm.

J. Grębosz.
Symfonia C++.
Kallimach, Kraków, 2000.

David Harel, Yishai Feldman.
Rzecz o istocie informatyki: algorytmika.
Klasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa, 2001, 2002, 2008.

Politechnika Wrocławska

Literatura III


George Heineman, Gary Pollice, Stanley Selkow.
Algorytmy. Almanach.
Helion, Gliwice, 2010.


Steve Holmes.
C programming.
http://www.imada.sdu.dk/~svalle/courses/dm14-2005/mirror/c/, 1995.


Ted Jensen.
A tutorial on pointers and arrays in C, Feb. 2000.
Dostępne jako
http://pweb.netcom.com/~tjensen/ptr/pointers.htm.

Politechnika Wrocławska

Literatura IV

- Rob Kendrick.
  Some dark corners of C, 2013.
- B. W. Kernighan, D. M. Ritchie.
  Język ANSI C.
  WNT, Warszawa, 2007.
- Ben Klemens.
  21st Century C.
  O'Reilly Media, 2012.
  http://shop.oreilly.com/product/0636920025108.do.

Politechnika Wrocławska

Literatura V


M. J. Kubiak.
Programuję w językach Turbo Pascal i C/C++: programowanie strukturalne z elementami programowania obiektowego.
Mikom, Warszawa, 2001.


Ciaran O'Riordan.
Learning GNU C.
Ciaran O'Riordan, 2002.


Nick Parlante.
Pointer basics, 1999.


Nick Parlante.
Pointers and memory, 2000.

Politechnika Wrocławska

Literatura VI

- Nick Parlante.
  Binary trees, 2001.
- Nick Parlante.
  The great tree-list recursion problem, 2001.
- Nick Parlante.
  Linked list problems, 2002.
- Nick Parlante.
  Essential C, 2003.
- Nick Parlante, Julie Zelenski.
  Unix programming tools, 2001.

Literatura VII


Richard M. Reese.
Understanding and Using C Pointers.
O'Reilly, 2013.


Richard M. Reese.
Wskaźniki w języku C: przewodnik.
Helion, Gliwice, 2014.
Dostęp po zalogowaniu w bazie NASBI.
http://biblioteka.pwr.wroc.pl/NASBI_Naukowa_Akademicka_Sieciowa_Biblioteka_Internetowa,161.dhtml.

Literatura VIII


Adam Sapek.
W głąb języka C.
Helion, Gliwice, lipiec 1993.
Wersja elektroniczna dostępna pod adresem:
`ftp://ftp.helion.pl/online/wglab/1-3.pdf`.


Herbert Schildt.
Leksykon C/C++.
Oficyna Wydawnicza LTP, Warszawa, 2002.


C. Sexton.
Język C to proste.
Wyd. RM, Warszawa, 2001.

Literatura IX


C. Sexton.
Programowanie w C++ — to proste.
RM, Warszawa, 2001.


Piotr Stańczyk.
Algorytmika praktyczna: Nie tylko dla mistrzów.
Wydawnictwo Naukowe PWN, Warszawa, 2009.


K. Stec.
Wybrane elementy języka C.
Wyd. Pol. Śląskiej, Gliwice, 2001.


Clovis L Tondo, Scott E Gimpel, Danuta Kruszewska.
Język ANSI C: Ćwiczenia I Rozwiązania.
Wydawnictwa Naukowo-Techniczne, Warszawa, wydanie wyd. 2, 2004.

Politechnika Wrocławska

Literatura X

- Peter Wayner.
  12 predictions for the future of programming, Luty 2014.
- N. Wirth.
  Algorytmy + struktury danych = programy.
  WNT, Warszawa, 2001.
- Piotr Wróblewski.
  Algorytmy: struktury danych i techniki programowania.
  Helion, Gliwice, 2010.

Politechnika Wrocławska

Część II

Część zasadnicza

Politechnika Wrocławska

Algorytm

Słowo „algorytm” jest bardzo nowe (w pewnym sensie).

Algorytm

Słowo „algorytm” jest bardzo nowe (w pewnym sensie).

Pochodzi od nazwiska Muḥammad ibn Mūsā al-Khwārizmī — perskiego matematyka (IX w) i pierwotnie oznaczało (każde) obliczenia w dziesiętnym systemie obliczeniowym.

Algorytm

Słowo „algorytm” jest bardzo nowe (w pewnym sensie).

Pochodzi od nazwiska Muḥammad ibn Mūsā al-Khwārizmī — perskiego matematyka (IX w) i pierwotnie oznaczało (każde) obliczenia w dziesiętnym systemie obliczeniowym.

Algorytm to jednoznaczny przepis przetworzenia w skończonym czasie pewnych danych wejściowych do pewnych danych wynikowych (Wikipedia). W tym znaczeniu pojawił się w latach pięćdziesiątych XX wieku.

Algorytm Euklidesa

Oto jedna z wersji algorytmu Euklidesa:

Algorytm Euklidesa

Oto jedna z wersji algorytmu Euklidesa:
Dane są dwie dodatnie liczby całkowite m i n, należy znaleźć ich ***największy wspólny dzielnik (NWD)*** *tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m jak i n.*

Politechnika Wrocławska

Algorytm Euklidesa

Oto jedna z wersji algorytmu Euklidesa:
Dane są dwie dodatnie liczby całkowite m i n, należy znaleźć ich ***największy wspólny dzielnik (NWD)*** *tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m jak i n.*

1. *[Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia. (Mamy* $0 \leq r < n$*.)*

Politechnika Wrocławska

Algorytm Euklidesa

Oto jedna z wersji algorytmu Euklidesa:
Dane są dwie dodatnie liczby całkowite m i n, należy znaleźć ich ***największy wspólny dzielnik (NWD)*** *tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m jak i n.*

1. *[Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia. (Mamy* $0 \leq r < n$*.)*
2. *[Czy wyszło zero?] Jeśli* $r = 0$ *zakończ algorytm; odpowiedzią jest n.*

Algorytm Euklidesa

Oto jedna z wersji algorytmu Euklidesa:
Dane są dwie dodatnie liczby całkowite m i n, należy znaleźć ich ***największy wspólny dzielnik (NWD)*** *tj. największą dodatnią liczbę całkowitą, która dzieli całkowicie zarówno m jak i n.*

1. *[Znajdowanie reszty] Podziel m przez n i niech r oznacza resztę z tego dzielenia. (Mamy* $0 \leq r < n$*.)*
2. *[Czy wyszło zero?] Jeśli* $r = 0$ *zakończ algorytm; odpowiedzią jest n.*
3. *[Upraszczanie] Wykonaj* $m \leftarrow n$, $n \leftarrow r$ *i wróć do kroku* **??**.

Cechy algorytmu I

Skończoność

Po pierwsze **powinien być skończony**; oznacza to, że po skończonej (być może bardzo dużej) liczbie kroków algorytm się zatrzyma[1]. Pytanie pomocnicze: Co gwarantuje, że algorytm Euklidesa zakończy się w skończonej liczbie kroków?

Procedura, która ma wszystkie cechy algorytmu poza skończonością nazywana jest *metodą obliczeniową*. Podaj przykłady metod obliczeniowych realizowanych przez rzeczywiste komputery.

[1] Ale sama skończoność to jednak za mało — z praktycznego punktu widzenia **dobry** algorytm powinien gwarantować, że obliczenia zostaną zakończone w skończonym ale **rozsądnym** czasie!

Cechy algorytmu II

Dobre zdefiniowanie

Po drugie **powinien być dobrze zdefiniowany**. Każdy krok algorytmu musi być opisany precyzyjnie. Wszystkie możliwe przypadki powinny być uwzględnione, a podejmowane akcje dobrze opisane[2]. Oczywiście język naturalny nie jest wystarczająco precyzyjny — może to prowadzić do nieporozumień. z tego powodu używa się bardziej formalnych sposobów zapisu algorytmów, aż po języki programowania…

[2] Zwracam też uwagę, że algorytmy kucharskie nie są odpowiednio precyzyjne: co to znaczy „lekko podgrzej”?

Politechnika Wrocławska

Cechy algorytmu III

Dane wejściowe

Po trzecie **powinien mieć precyzyjnie zdefiniowane dane wejściowe**. Pewne algorytmy mogą nie mieć danych wejściowych (mieć zero danych wejściowych). Dane wejściowe to wartości, które muszą być zdefiniowane zanim rozpocznie się wykonanie algorytmu.

Politechnika Wrocławska

Cechy algorytmu IV

Dane wyjściowe

Po czwarte **zdefiniowane dane wyjściowe**. Daną wyjściową algorytmu Euklidesa jest liczna *n* która jest naprawdę największym wspólnym dzielnikiem danych wejściowych. Osobną sprawą jest pokazanie skąd wynika, że wynik algorytmu Euklidesa jest rzeczywiście NWD liczb *m* i *n*.

Cechy algorytmu V

Efektywność

Po piąte algorytm **powinien być określony efektywnie** to znaczy jego operacje powinny być wystarczająco proste by można je (teoretycznie?) wykonać w skończonym czasie z wykorzystaniem kartki i ołówka.

Politechnika Wrocławska

Przykładowe algorytmy

1. Przepis kucharski,
2. Algorytm Euklidesa,
3. Wyszukiwanie osoby najwyższej/najniższej
4. Algorytm Euklidesa (drugi wariant — tak zwany „Algorytm B”)

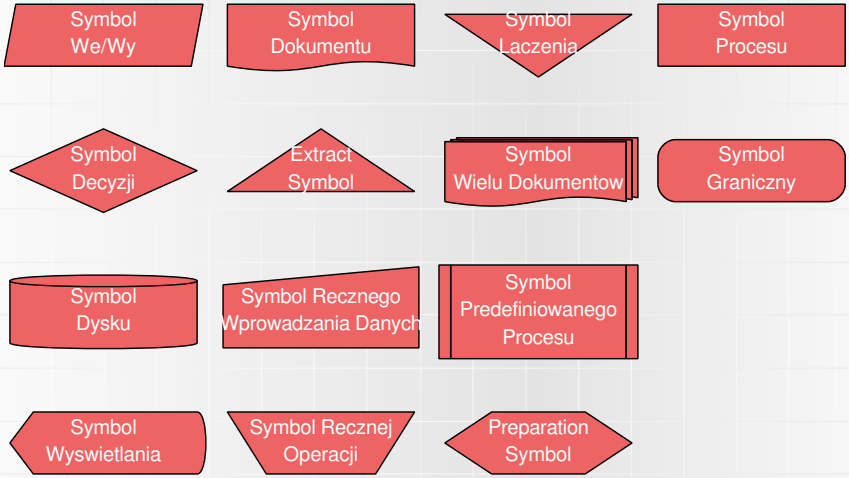
Politechnika Wrocławska

Sposób zapisu algorytmów

1. Język naturalny
2. Schemat blokowy
3. Tablica decyzyjna
4. Maszyna Turinga(???)
5. Język programowania

Politechnika Wrocławska

Schematy blokowe



Politechnika Wrocławska

UML — Unified Modeling Language

UML (*Unified Modeling Language* — ujednolicony język modelowania) to graficzny język do obrazowania, specyfikowania, tworzenia i dokumentowania elementów systemów informatycznych. Umożliwia standaryzację sposobu opracowywania przekrojów systemu, obejmujących obiekty pojęciowe, takie jak procesy przedsiębiorstwa i funkcje systemowe, a także obiekty konkretne, takie jak klasy zaprogramowane w ustalonym języku, schematy baz danych i komponenty programowe nadające się do ponownego użycia.

Programowanie

Prosty przykład

Mamy rozwiązać zadanie:

$$ax^2 + bx + c = 0$$

Programowanie

Prosty przykład

Mamy rozwiązać zadanie:

$$ax^2 + bx + c = 0$$

Metoda:

Programowanie

Prosty przykład

Mamy rozwiązać zadanie:

$$ax^2 + bx + c = 0$$

Metoda:

1. Wylicz $\Delta = b^2 - 4ac$

Programowanie

Prosty przykład

Mamy rozwiązać zadanie:

$$ax^2 + bx + c = 0$$

Metoda:

1. Wylicz $\Delta = b^2 - 4ac$
3. Oblicz $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$
4. Oblicz $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$

Programowanie

Prosty przykład

Mamy rozwiązać zadanie:

$$ax^2 + bx + c = 0$$

Metoda:

1. Wylicz $\Delta = b^2 - 4ac$
2. Jeżeli $\Delta < 0$ — nie ma pierwiastków rzeczywistych; koniec
3. Oblicz $x_1 = \dfrac{-b - \sqrt{\Delta}}{2a}$
4. Oblicz $x_2 = \dfrac{-b + \sqrt{\Delta}}{2a}$

Politechnika Wrocławska

Programowanie

Realizacja

1. **Wprowadź** a, b, c
2. **Wylicz** Delta=b * b – 4 * a * c
3. **Jeżeli** Delta <0 **to**
   - 3.1 **wypisz tekst** "Nie ma pierwiastków rzeczywistych"; **koniec**

   w przeciwnym razie
   - 3.1 x1=(–b – sqrt(Delta))/(2 * a)
   - 3.2 x2=(–b + sqrt(Delta))/(2 * a)
4. **Wypisz** "x1 = ", x1, " x2 = ", x2
5. **koniec**

Programowanie

Realizacja

1. **Wprowadź** a, b, c
2. **Wylicz** Delta=b * b – 4 * a * c
3. **Jeżeli** Delta <0 **to**
   - 3.1 **wypisz tekst** "Nie ma pierwiastków rzeczywistych"; **koniec**

   w przeciwnym razie
   - 3.1 x1=(–b – sqrt(Delta))/(2 * a)
   - 3.2 x2=(–b + sqrt(Delta))/(2 * a)
4. **Wypisz** "x1 = ", x1, " x2 = ", x2
5. **koniec**

Program nie jest fajny bo ma dwa końce…

Programowanie

Realizacja

1. **Wprowadź** a, b, c
2. **Wylicz** Delta=b * b – 4 * a * c
3. **Jeżeli** Delta <0 **to**
   - 3.1 **wypisz tekst** "Nie ma pierwiastków rzeczywistych"

   w przeciwnym razie
   - 3.1 x1=(–b – sqrt(Delta))/(2 * a)
   - 3.2 x2=(–b + sqrt(Delta))/(2 * a)
   - 3.3 **Wypisz** "x1 = ", x1, " x2 = ", x2
4. **koniec**

Politechnika Wrocławska

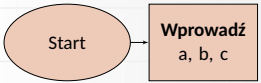
Programowanie

A schemat blokowy?

Politechnika Wrocławska

Programowanie

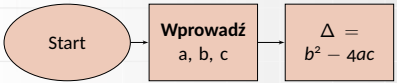
A schemat blokowy?



Politechnika Wrocławska

Programowanie

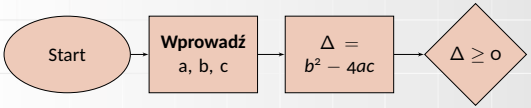
A schemat blokowy?



Politechnika Wrocławska

Programowanie

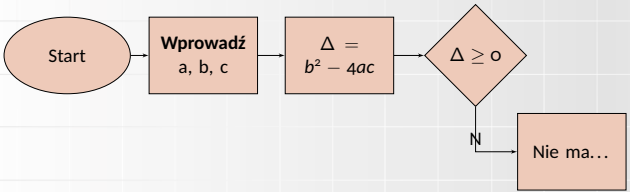
A schemat blokowy?



Politechnika Wrocławska

Programowanie

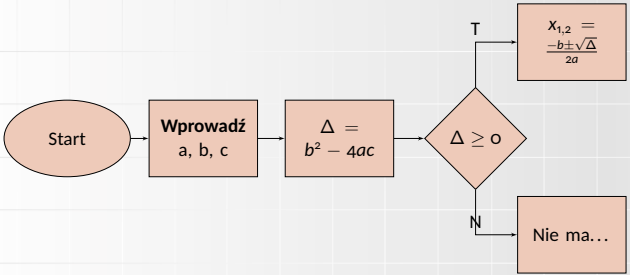
A schemat blokowy?



Politechnika Wrocławska

Programowanie

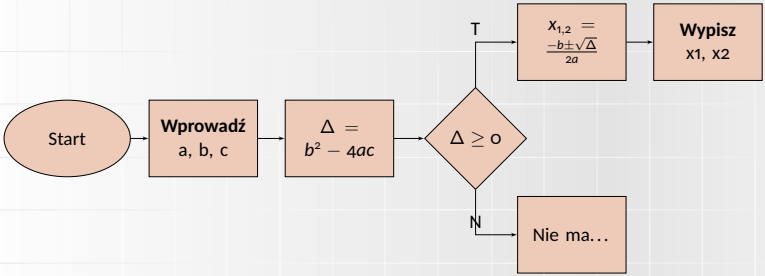
A schemat blokowy?



Politechnika Wrocławska

Programowanie

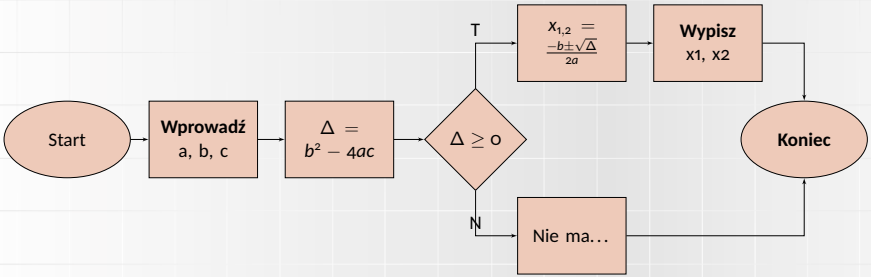
A schemat blokowy?



Politechnika Wrocławska

Programowanie

A schemat blokowy?



Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.

Politechnika Wrocławska

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.

Politechnika Wrocławska

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.
3. W pewnym uproszczeniu **programowanie strukturalne** to pewne rozszerzenie **programowania proceduralnego**.

Politechnika Wrocławska

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.
3. W pewnym uproszczeniu **programowanie strukturalne** to pewne rozszerzenie **programowania proceduralnego**.
4. Polega na po podziale kodu na dobrze zdefiniowane moduły (bloki).

Politechnika Wrocławska

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.
3. W pewnym uproszczeniu **programowanie strukturalne** to pewne rozszerzenie **programowania proceduralnego**.
4. Polega na po podziale kodu na dobrze zdefiniowane moduły (bloki).
5. Komunikacja między modułami odbywa się za pomocą precyzyjnie określonych interfejsów.

Politechnika Wrocławska

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.
3. W pewnym uproszczeniu **programowanie strukturalne** to pewne rozszerzenie **programowania proceduralnego**.
4. Polega na po podziale kodu na dobrze zdefiniowane moduły (bloki).
5. Komunikacja między modułami odbywa się za pomocą precyzyjnie określonych interfejsów.
6. Wedle innych to stosowanie konstrukcji języków programowania takich jak pętle i instrukcje warunkowe, oraz…

Idea programowania strukturalnego

1. Jest na ten temat piękna teoria — Dijkstra.
2. Wielu teoretyków programowanie strukturalne rozumie nieco inaczej niż Dijkstra.
3. W pewnym uproszczeniu **programowanie strukturalne** to pewne rozszerzenie **programowania proceduralnego**.
4. Polega na po podziale kodu na dobrze zdefiniowane moduły (bloki).
5. Komunikacja między modułami odbywa się za pomocą precyzyjnie określonych interfejsów.
6. Wedle innych to stosowanie konstrukcji języków programowania takich jak pętle i instrukcje warunkowe, oraz…
7. …unikanie instrukcji skoku (goto) oraz wielokrotnych punktów wejścia i wyjścia z kodu bloku (podprogramu).

Idea programowania strukturalnego

Postawienie problemu

Załóżmy, że mamy wyznaczyć pierwiastek stopnia n z liczby w, czyli znaleźć taką liczbę x, że:

$$x^n = w \tag{1}$$

lub inaczej:

$$x^n - w = 0 \tag{2}$$

Jeżeli oznaczymy $f(x) = x^n - w$ to zadanie to można zapisać ogólniej: należy znaleźć takie x, że $f(x) = 0$.

Idea programowania strukturalnego

Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia

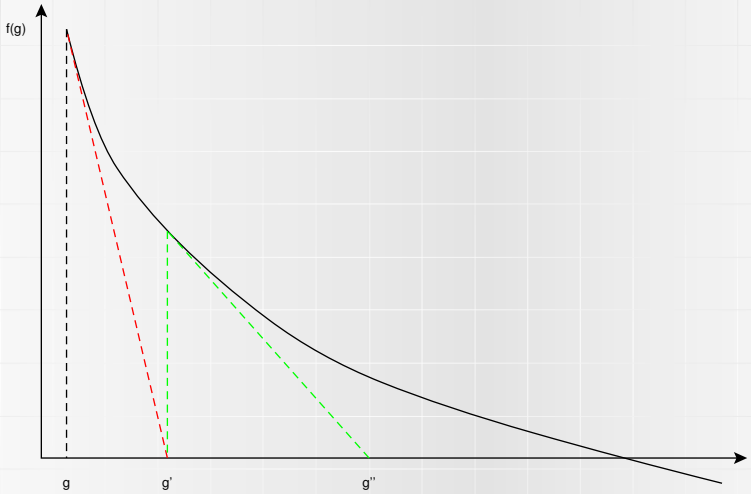
Jeżeli zadanie dodatkowo uprościmy zakładając:

- funkcja ma dokładnie jedno miejsce zerowe,
- jest różniczkowalna,
- jej pochodna w całym przedziale jest albo dodatnia albo ujemna;

to możemy naszkicować rysunek ilustrujący nasze zadanie.

Idea programowania strukturalnego

Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia



Idea programowania strukturalnego

Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia

Zaczynamy w punkcie g; wartość funkcji w tym punkcie wynosi $f(g)$. Musimy w jakiś sposób zdecydować w którym kierunku należy wykonać następny krok. Zauważmy, że możemy w tym celu wykorzystać pochodną (czerwona, przerywana linia na poprzednim rysunku). Jeżeli przybliżymy funkcję za pomocą pochodnej (stycznej do funkcji, przechodzącej przez punk $(g, f(g)$ to następnym przybliżeniem będzie punkt przecięcia stycznej z osią x.

Idea programowania strukturalnego

Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia

Z równania prostej mamy:

$$\frac{f(g) - 0}{g - g'} = f'(g) \tag{3}$$

czyli

$$\frac{f(g)}{f'(g)} = g - g' \tag{4}$$

i dalej

$$g' = g - \frac{f(g)}{f'(g)} \tag{5}$$

Politechnika Wrocławska

Idea programowania strukturalnego

Metoda Newtona-Raphsona: pierwiastek dowolnego stopnia

Jeżeli zauważymy, że $f(x) = x^n - w$ oraz, że $f'(x) = nx^{n-1}$ to kolejne przybliżenie wyliczane będzie ze wzoru:

$$g' = g - \frac{g^n - w}{ng^{n-1}} \tag{6}$$

albo

$$g' = \frac{ng^n - g^n + w}{ng^{n-1}} = \frac{(n-1)g^n + w}{ng^{n-1}} = \frac{1}{n}\left((n-1)g + \frac{w}{g^{n-1}}\right) \tag{7}$$

Gdy $n = 2$, wówczas

$$g' = \frac{1}{2}\left(g + \frac{w}{g}\right). \tag{8}$$

Umawiamy się, że program kończy pracę gdy kolejna poprawka g' nie różni się zbytnio od poprzednio wyliczonej wartości g, czyli $|g - g'| < \varepsilon$.

Idea programowania strukturalnego

Realizacja programowa

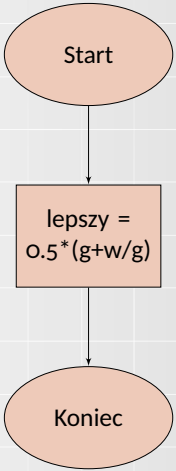
Program będzie się składał z trzech części:

1. `blisko(g, gprim)` — funkcja o wartościach logicznych sprawdzająca czy $|g - g'| \leq \varepsilon$,
2. `lepszy(n, w, g)` — funkcja rzeczywista wyliczająca następne, lepsze przybliżenie pierwiastka,
3. `pierwiastek(n, w, g)` — funkcja (rzeczywista) wyliczająca pierwiastek stopnia n z w zaczynając od przybliżenia g.

Uwaga: Dalszy przykład zakłada $n = 2$

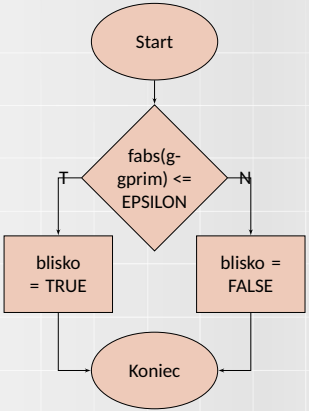
Realizacja programowa

`lepszy(w, g)`



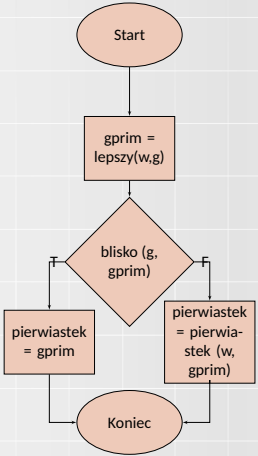
Realizacja programowa

`blisko(g, gprim)`



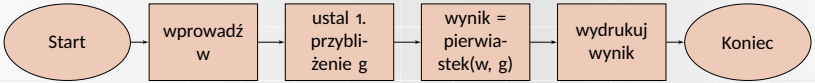
Realizacja programowa

`pierwiastek(w, g)`



Realizacja programowa

Program główny



Politechnika Wrocławska

Czemu C?

- Nie wiem!

Politechnika Wrocławska

Czemu C?

- Nie wiem!
- Jest bardzo popularny.

Politechnika Wrocławska

Czemu C?

- Nie wiem!
- Jest bardzo popularny.
- Stosunkowo prosty.

Czemu C?

- Nie wiem!
- Jest bardzo popularny.
- Stosunkowo prosty.
- C++ (jeden z podstawowych języków obiektowych) „wyrasta” z C.

Politechnika Wrocławska

Czemu C?

- Nie wiem!
- Jest bardzo popularny.
- Stosunkowo prosty.
- C++ (jeden z podstawowych języków obiektowych) „wyrasta” z C.
- Dosyć niskiego poziomu.

Politechnika Wrocławska

Czemu C?

- Nie wiem!
- Jest bardzo popularny.
- Stosunkowo prosty.
- C++ (jeden z podstawowych języków obiektowych) „wyrasta” z C.
- Dosyć niskiego poziomu.
- Deczko „hakerski”.

Hello World — wersja łatwa I

```
#!/usr/bin/env python

# example HelloWorld.py on terminal

print("Hello World!")
```

Hello World — wersja trudna I

```
#!/usr/bin/env python

# example helloworld.py

import pygtk
pygtk.require('2.0')
import gtk

class HelloWorld:

    # This is a callback function. The data arguments
    # are ignored in this example. More on callbacks
    # below.
    def hello(self, widget, data=None):
        print "Hello World"

    def delete_event(self, widget, event, data=None):
```

Hello World — wersja trudna II

```
        # If you return FALSE in the "delete_event"
        # signal handler, GTK will emit the "destroy"
        # signal. Returning TRUE means you don't want
        # the window to be destroyed. This is useful
        # for popping up 'are you sure you want to quit?'
        # type dialogs.
        print "delete event occurred"

        # Change FALSE to TRUE and the main window will
        # not be destroyed with a "delete_event".
        return False

    def destroy(self, widget, data=None):
        print "destroy signal occurred"
        gtk.main_quit()

    def __init__(self):
```

Hello World — wersja trudna III

```
        # create a new window
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)

        # When the window is given the "delete_event"
        # signal (this is given by the window manager,
        # usually by the "close" option, or on the
        # titlebar), we ask it to call the delete_event()
        # function as defined above. The data passed to
        # the callback function is NULL and is ignored
        # in the callback function.
        self.window.connect("delete_event", self.delete_event)

        # Here we connect the "destroy" event to a signal
        # handler. This event occurs when we
        # call gtk_widget_destroy() on the window,
        # or if we return FALSE in the "delete_event"
        # callback.
```

Hello World — wersja trudna IV

```
        self.window.connect("destroy", self.destroy)

        # Sets the border width of the window.
        self.window.set_border_width(10)

        # Creates a new button with the label "Hello
        # World".
        self.button = gtk.Button("Hello World")

        # When the button receives the "clicked" signal,
        # it will call the function hello() passing it
        # None as its argument.  The hello() function
        # is defined above.
        self.button.connect("clicked", self.hello, None)

        # This will cause the window to be destroyed by
        # calling gtk_widget_destroy(window) when
```

Hello World — wersja trudna V

```
        # "clicked". Again, the destroy signal could
        # come from here, or the window manager.
        self.button.connect_object("clicked",
                                   gtk.Widget.destroy, self.window)

        # This packs the button into the window (a GTK
        # container).
        self.window.add(self.button)

        # The final step is to display this newly
        # created widget.
        self.button.show()

        # and the window
        self.window.show()

    def main(self):
```

Hello World — wersja trudna VI

```
        # All PyGTK applications must have a gtk.main().
        # Control ends here and waits for an event to
        # occur (like a key press or mouse event).
        gtk.main()

# If the program is run directly or passed as an argument
# to the python interpreter then create a HelloWorld
# instance and show it
if __name__ == "__main__":
    hello = HelloWorld()
    hello.main()
```

Silnia na wiele sposobów I

- Newbie programmer

```
#Newbie programmer
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
print factorial(6)
```

Silnia na wiele sposobów II

- First year programmer, studied Pascal

```
#First year programmer, studied Pascal
def factorial(x):
    result = 1
    i = 2
    while i <= x:
        result = result * i
        i = i + 1
    return result
print factorial(6)
```

Silnia na wiele sposobów III

- First year programmer, studied C

```
#First year programmer, studied C
def fact(x): #{
    result = i = 1;
    while (i <= x): #{
        result *= i;
        i += 1;
    #}
    return result;
#}
print(fact(6))
```

Silnia na wiele sposobów IV

- First year programmer, SICP

```
#First year programmer, SICP
@tailcall
def fact(x, acc=1):
    if (x > 1): return (fact((x - 1), (acc * x)))
    else: return acc
print(fact(6))
```

SICP — Structure and Interpretation of Computer Programs

Silnia na wiele sposobów V

- First year programmer, Python

```
#First year programmer, Python
def Factorial(x):
    res = 1
    for i in xrange(2, x + 1):
        res *= i
    return res
print Factorial(6)
```

Silnia na wiele sposobów VI

- Lazy Python programmer

```
#Lazy Python programmer
def fact(x):
    return x > 1 and x * fact(x - 1) or 1
print fact(6)
```

Silnia na wiele sposobów VII

- Lazier Python programmer

```
#Lazier Python programmer
f = lambda x: x and x * f(x — 1) or 1
print f(6)
```

Politechnika Wrocławska

Silnia na wiele sposobów VIII

- Python expert programmer

```
#Python expert programmer
import operator as op
import functional as f
fact = lambda x: f.foldl(op.mul, 1, xrange(2, x + 1))
print fact(6)
```

Silnia na wiele sposobów IX

- Python hacker

```
#Python hacker
import sys
@tailcall
def fact(x, acc=1):
    if x: return fact(x.__sub__(1), acc.__mul__(x))
    return acc
sys.stdout.write(str(fact(6)) + '\n')
```

Politechnika Wrocławska

Silnia na wiele sposobów X

- EXPERT PROGRAMMER

```
#EXPERT PROGRAMMER
import c_math
fact = c_math.fact
print fact(6)
```

Silnia na wiele sposobów XI

- ENGLISH EXPERT PROGRAMMER

```
#ENGLISH EXPERT PROGRAMMER
import c_maths
fact = c_maths.fact
print fact(6)
```

Silnia na wiele sposobów XII

- Web designer

```
#Web designer
def factorial(x):
    #————————————————————————————————————————————
    #——— Code snippet from The Math Vault ———
    #——— Calculate factorial (C) Arthur Smith 1999 ———
    #————————————————————————————————————————————
    result = str(1)
    i = 1 #Thanks Adam
    while i <= x:
        #result = result * i #It's faster to use *=
        #result = str(result * result + i)
            #result = int(result *= i) #??????
        result str(int(result) * i)
        #result = int(str(result) * i)
        i = i + 1
```

Silnia na wiele sposobów XIII

```
    return result
print factorial(6)
```

Silnia na wiele sposobów XIV

- Unix programmer

```
#Unix programmer
import os
def fact(x):
    os.system('factorial ' + str(x))
fact(6)
```

Silnia na wiele sposobów XV

- Windows programmer

```
#Windows programmer
NULL = None
def CalculateAndPrintFactorialEx(dwNumber,
                                 hOutputDevice,
                                 lpLparam,
                                 lpWparam,
                                 lpsscSecurity,
                                 *dwReserved):
    if lpsscSecurity != NULL:
        return NULL #Not implemented
    dwResult = dwCounter = 1
    while dwCounter <= dwNumber:
        dwResult *= dwCounter
        dwCounter += 1
    hOutputDevice.write(str(dwResult))
```

Silnia na wiele sposobów XVI

```
    hOutputDevice.write('\n')
    return 1
import sys
CalculateAndPrintFactorialEx(6, sys.stdout, NULL, NULL,
                             NULL, NULL, NULL, NULL,
                             NULL, NULL, NULL, NULL,
                             NULL, NULL, NULL, NULL,
                                               NULL, NULL)
```

Silnia na wiele sposobów XVII

- Enterprise programmer

```
#Enterprise programmer
def new(cls, *args, **kwargs):
    return cls(*args, **kwargs)

class Number(object):
    pass

class IntegralNumber(int, Number):
    def toInt(self):
        return new (int, self)

class InternalBase(object):
    def __init__(self, base):
        self.base = base.toInt()
```

Silnia na wiele sposobów XVIII

```
    def getBase(self):
        return new(IntegralNumber, self.base)

class MathematicsSystem(object):
    def __init__(self, ibase):
        Abstract

    @classmethod
    def getInstance(cls, ibase):
        try:
            cls.__instance
        except AttributeError:
            cls.__instance = new(cls, ibase)
        return cls.__instance

class StandardMathematicsSystem(MathematicsSystem):
    def __init__(self, ibase):
```

Silnia na wiele sposobów XIX

```
        if ibase.getBase() != new (IntegralNumber, 2):
            raise NotImplementedError
        self.base = ibase.getBase()

    def calculateFactorial(self, target):
        result = new (IntegralNumber, 1)
        i = new (IntegralNumber, 2)
        while i <= target:
            result = result * i
            i = i + new (IntegralNumber, 1)
        return result

print StandardMathematicsSystem.getInstance(
            new (InternalBase,
            new (IntegralNumber, 2))).calculateFactorial(
            new (IntegralNumber, 6))
```

Politechnika Wrocławska

Ewolucja języków programowania I

- 1980: C

```
printf("%10.2f", x);
```

- 1988: C++

```
cout << setw(10) << setprecision(2) << showpoint << x;
```

- 1996: Java

```
    java.text.NumberFormat formatter =
    java.text.NumberFormat.getNumberInstance();
    formatter.setMinimumFractionDigits(2);
    formatter.setMaximumFractionDigits(2);
    String s = formatter.format(x);
    for (int i = s.length(); i < 10; i++)
                        System.out.print('␣');
    System.out.print(s);
```

Politechnika Wrocławska

Ewolucja języków programowania II

- 2004: Java

```
System.out.printf("%10.2f", x);
```

- 2008: Scala and Groovy

```
printf("%10.2f", x)
```

Proces tworzenia programu

1. Potrzeba (Zadanie).

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.
4. Program.

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.
4. Program.
5. Kompilacja (przetworzenie z postaci kodu źródłowego do postaci pośredniej).

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.
4. Program.
5. Kompilacja (przetworzenie z postaci kodu źródłowego do postaci pośredniej).
6. Uruchomienie.

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.
4. Program.
5. Kompilacja (przetworzenie z postaci kodu źródłowego do postaci pośredniej).
6. Uruchomienie.
7. (Ewentualnie) odpluskwianie…

Politechnika Wrocławska

Proces tworzenia programu

1. Potrzeba (Zadanie).
2. Pomysł.
3. Algorytm.
4. Program.
5. Kompilacja (przetworzenie z postaci kodu źródłowego do postaci pośredniej).
6. Uruchomienie.
7. (Ewentualnie) odpluskwianie…
8. Poprawa błędów (ewentualnie zmiana algorytmu)

Politechnika Wrocławska

Zadania domowe

1. Co gwarantuje, że algorytm Euklidesa zakończy się w skończonej liczbie kroków?
2. Podaj przykłady metod obliczeniowych realizowanych przez rzeczywiste komputery.
3. Przerysować schematy blokowe Metody Newtona dla przypadku ogólnego (to znaczy dla dowolnego n).
4. Narysować schemat blokowy „Algorytmu B” (następny slajd!)

Politechnika Wrocławska

Algorytm B

Algorytm B

Dla danych dodatnich liczb całkowitych u i v algorytm ten znajduje **największy wspólny dzielnik**.

B1 Przyjmij $k \leftarrow 0$, a następnie powtarzaj operacje: $k \leftarrow k+1$, $u \leftarrow u/2$, $v \leftarrow v/2$ zero lub więcej razy do chwili gdy przynajmniej jedna z liczb u i v przestanie być parzysta.

B2 Jeśli u jest nieparzyste to przyjmij $t \leftarrow -v$ i przejdź do kroku B4. W przeciwnym razie przyjmij $t \leftarrow u$.

B3 (W tym miejscu t jest parzyste i różne od zera). Przyjmij $t \leftarrow t/2$.

B4 Jeśli t jest parzyste to przejdź do B3.

B5 Jeśli $t > 0$, to przyjmij $u \leftarrow t$, w przeciwnym razie przyjmij $v \leftarrow -t$.

B6 Przyjmij $t \leftarrow u - v$. Jeśli $t \neq 0$ to wróć do kroku B3. W przeciwnym razie algorytm zatrzymuje się z wynikiem $u \cdot 2^k$.