POLITECHNIKA KRAKOWSKA – WIEiK

KATEDRA AUTOMATYKI I TECHNIK INFORMACYJNYCH

Metody Programowania

www.pk.edu.pl/~zk/MP_HP.html

Wykładowca:
dr hab. inż. Zbigniew Kokosiński
zk@pk.edu.pl

Wykład 1: P r o g r a m o w a n i e

- Definicja programowania
- Początki programowania w starożytności
- Pierwsza maszyna licząca i język programowania
- Komputer von Neumanna i ENIAC
- Dualizm programowo-sprzętowy
- Języki, środowiska programowania
- Hipoteza Sapira-Whorfa
- Programowanie – klasyfikacje
- Kompilacja kodu źródłowego
- Wymagania jakościowe
- Metody programowania

Definicja programowania

Programowanie (kodowanie) jest procesem obejmującym tworzenie, testowanie, usuwanie błędów oraz bieżące utrzymywanie kodu źródłowego programów komputerowych.

Kod źródłowy jest zapisem wybranego algorytmu, przetwarzającego w pożądany sposób dane wejściowe w dane wyjściowe, w pewnym języku programowania.

Algorytm to metoda rozwiązania problemu algorytmicznego (istnieją także problemy niealgorytmiczne i prawdopodobnie niealgorytmiczne).

Dane przechowywane są zwykle w postaci struktur. Wybór struktur danych zależy od problemu, własności języka programowania oraz samego programisty.

Programy = algorytmy + struktury danych (*Niklaus Wirth*)

Programowanie – rzemiosło czy sztuka ?

Przykładowe nazwy podręczników dla programistów:

The art of computer programming (*Donald Knuth*)

Algorithmics. The spirit of computing (*David Harel*)

Programming perls (*Jon Bentley*)

Algorithms + Data Structures = programs (*Niklaus Wirth*)

The practice of programming (*Brian Kernighan, Robert Pike*)

Podstawy i praktyka programowania mikroprocesorów
(*Jerzy Grabowski, Stanisław Koślacz*)

The pragmatic programmer. From Journeyman to Master
(*Andrew Hunt, David Thomas*)

Pierwsze algorytmy – procedury administracyjne

~2340–2000, Mezopotamia – procedury administracyjne w okresie III dynastii babilońskiej, pierwsze algorytmy

~IX w. p.n.e., Kartagina – legendarna założycielka miasta i jego pierwsza królowa Dydona, wytyczyła obszar o największej powierzchni przy najmniejszym obwodzie (w kształcie koła), mając do jego wyznaczenia jedynie wołową skórę.

Współczesne narzędzia:

1. **procedury przetwarzania danych,**
2. **procedury numeryczne,**
3. **optymalizacja, badania operacyjne**

Algorytm babiloński

– Algorytm oblicza wielkość pól I i II, mając daną sumę gruntów S, zbiór z 30 jednostek pola I, zbiór z 30 jednostek pola II oraz nadwyżkę zbioru z pola I.

"30, sumę gruntów, rozbij na dwie części, weź 15 i 15.
Utwórz odwrotność 30, pola, jest $\frac{1}{30}$.
$\frac{1}{30}$ pomnóż przez zboże 20, któreś zebrał, $\frac{20}{30}$ jest zbożem, które wypadło.
Pomnóż przez 15, jest 10, i zachowaj w głowie.
Odwrotność 30 utwórz, i jest $\frac{1}{30}$.
$\frac{1}{30}$ przez 15, zboże, któreś zebrał, pomnóż.
$\frac{15}{30}$ jest to zboże tymczasowe.
Pomnóż przez 15, jest $7\ \frac{1}{2}$.
10, które zachowała twoja głowa, o ile wychodzi poza $7\ \frac{1}{2}$?
O $2\ \frac{1}{2}$ wychodzi.
$2\ \frac{1}{2}$ od $8\ \frac{1}{3}$, zboża, odejmij i zostanie ci $5\ \frac{5}{6}$.
$\frac{2}{3}$ i $\frac{1}{2}$ dodaj, i jest $1\ \frac{1}{6}$.
Co trzeba pomnożyć przez $1\ \frac{1}{6}$, aby mieć $5\ \frac{5}{6}$, które zachowała twoja głowa?
5 pomnożone przez $1\ \frac{1}{6}$, da ci $5\ \frac{5}{6}$.
5, któreś uzyskał, do jednego z dwóch 15 dodaj, od drugiego odejmij i masz z pierwszego 20, z drugiego 10.
20 jest powierzchnia pierwszego gruntu, 10 drugiego".

Algorytm babiloński – współczesny program

Wejście: S – suma gruntów; Z1 – zbiór z 30 jednostek pola I; Z2 – zbiór z 30 jednostek pola II; NW – nadwyżka zbioru z pola I.
Wyjście: P1 – wielkość pola I; P2 – wielkość pola II.

1. wczytaj S=30; Z1=20; Z2=15; NW=8 1/3;
2. PS=S/2; // oblicz połowę S
3. OS=1/S; // oblicz odwrotność S
4. ZW1=OS*Z1; // oblicz zbiór na jednostkę pola I
5. ZG1=ZW1*PS; // oblicz zbiór z pierwszej połowy pola S
6. ZW2=OS*Z2; // oblicz zbiór na jednostkę pola II
7. ZG2=ZW2*PS; // oblicz zbiór z drugiej połowy pola S
8. R=ZG1-ZG2; // oblicz różnicę zbiorów
9. R: RZ=NW-R; // oblicz różnicę nadwyżki NW i różnicy R
10. SP=ZW1+ZW2; // oblicz sumę zbiorów na jednostkę
11. X=RZ/SP; // oblicz X
12. P1=PS+X; // oblicz P1
13. P2=PS-X; // oblicz P2
14. wyprowadź P1,P2.

Egipski podręcznik algebry i geometrii

~1750 p.n.e. Pisarz Ahmes (Ahmose) wyjaśnia obliczanie pól powierzchnii zamknietych i objętości (np. piramid i spichlerzy, pisze o ułamkach, ciągach arytmetycznych (np. 2,4,6,8,10, ...) i geometrycznych (np. 2,4,8,16,32, ...), rozwiązuje równania z jedną niewiadomą.

Przebieg typowego obliczenia wykonanego przez Ahmesa:

"Kiedy wiadomo, że kawałek ziemi ma formę uciętego trójkąta (trapez) z dwoma bokami o długości 20 khet, podstawę o długości 6 khet, a przekatną o długości 4 khet, to jaka jest jego powierzchnia? Dodaj przekątną do podstawy. Otrzymasz 10. Weź połowę z tego, będzie to 5. Pomnóż 5x20. Wynik 100 kwadratowych khetów."

Algorytmy Greków: algorytm Euklidesa

(twórca: Eudoksos z Knidos)

Algorytm Euklidesa (metoda kolejnych dzieleń) to algorytm znajdowania największego wspólnego dzielnika (NWD) dwóch różnych liczb naturalnych. Nie wymaga rozkładania liczb na czynniki pierwsze.

Przebieg algorytmu Euklidesa obliczania NWD liczb a i b :

1. Oblicz c jako resztę z dzielenia a przez b.
2. Zastąp a przez b, zaś b przez c.
3. Jeżeli b = 0, to szukane NWD = a,
   w przeciwnym wypadku przejdź do 1.

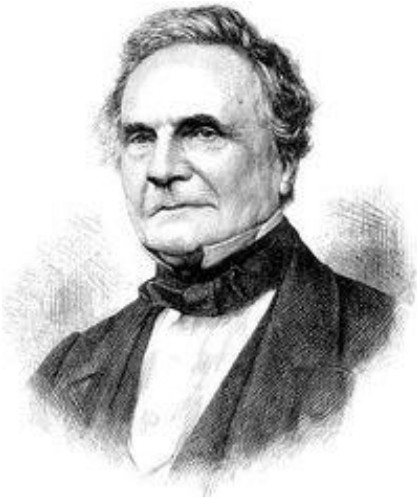
Algorytmy Greków: algorytm Eratostenesa

Sito Eratostenesa to algorytm kolejnego wybierania liczb pierwszych z ciągu liczb naturalnych.

Przebieg algorytmu „sito Eratostenesa” :

1. Ze zbioru liczb naturalnych większych od jedności wybieramy najmniejszą i wykreślamy wszystkie jej wielokrotności większe od niej samej.
2. Powtarzamy krok 1 dla kolejnych liczb, które nie zostały wykreślone.
3. Liczby niewykreślone w danym przedziale stanowią rozwiązanie.

Pierwsza maszyna licząca i język programowania



Charles Babbage (1791-1871)

koncepcja komputera mechanicznego - „maszyny analitycznej”

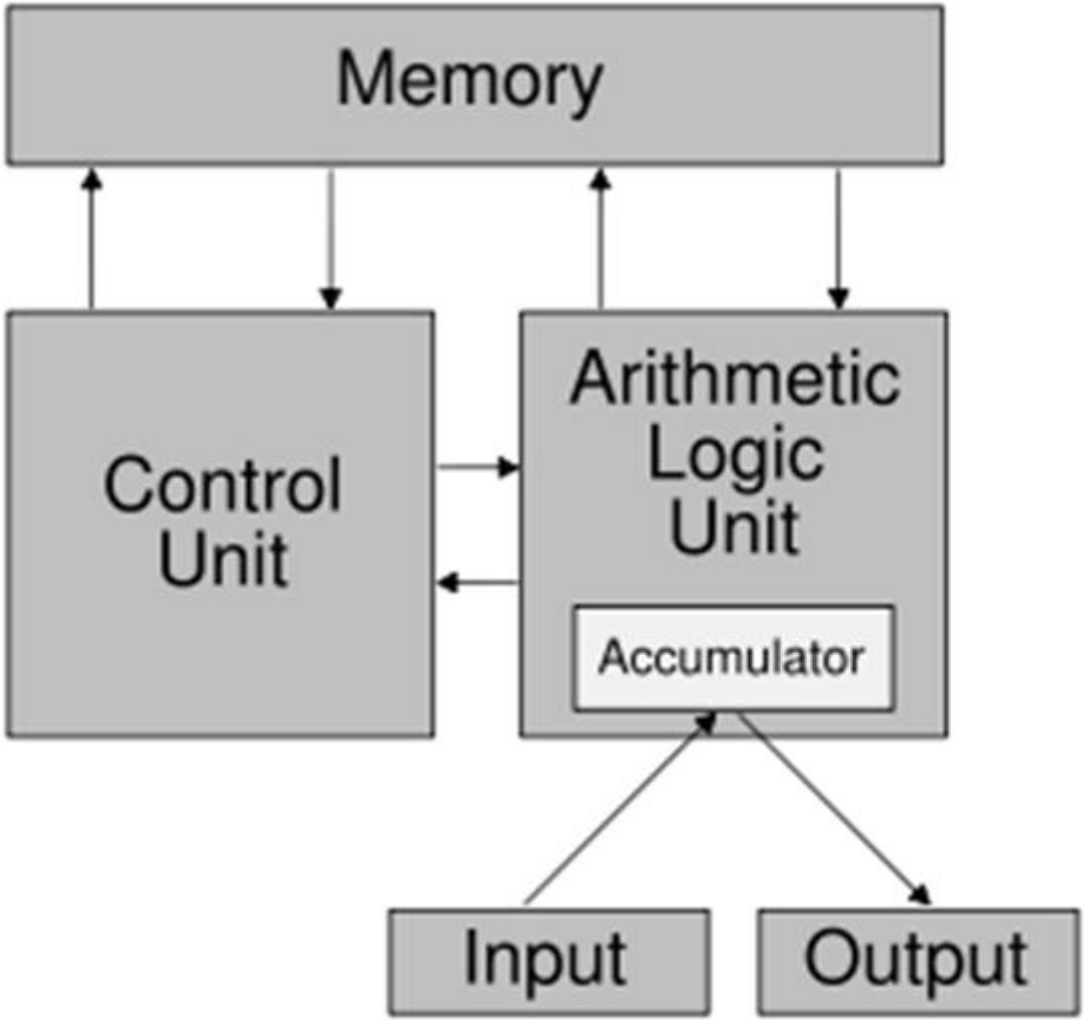
Ada Lovelace (1815-1852)

pierwszy program komputerowy - obliczanie liczb Bernoulliego w maszynie Babbage’a

Model komputera von Neumanna

– Komunikacja pomiędzy pamiecią danych a wejściem i wyjściem komputera do poprzez ALU (wąskie gardło)

John von Neumann (1903-1957)



ENIAC - elektroniczna maszyna licząca

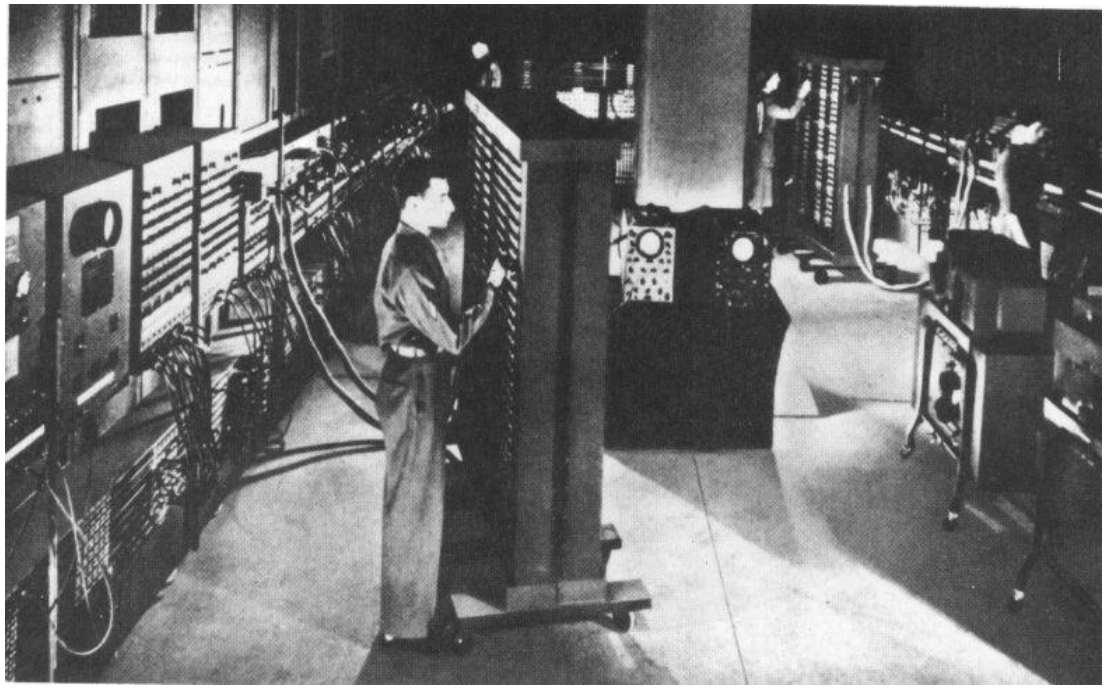
1945, USA, Uniwersytet Pensylwania

konstruktorzy: J.P. Eckert i J.W. Maulchy

18 000 lamp, 1 500 przekaźników, zużycie mocy 150 kW

30 ton, 140 m kw.

Realizacja programów częściowo sprzętowa (kable)



Dualizm programowo-sprzętowy

Systemy komputerowe składają się ze **sprzętu** (*hardware*) i **oprogramowania** (*software*).

Komputery ogólnego przeznaczenia posiadają procesory z ustaloną listą rozkazów.

W procesorach **RISC** (*Reduced Instruction Set*) lista rozkazów jest uproszczona do minimum. Powoduje to w konsekwencji konieczność budowy bardziej złożonych programów.

W procesorach **CISC** (*Complex Instruction Set*) lista rozkazów jest rozbudowana. Powoduje to w konsekwencji skrócenie i uproszczenie programów.

Generalnie wszystko to, co w komputerach ogólnego przeznaczenia może być zrealizowane programowo, może też być zrealizowane sprzętowo (np. przez szybkie procesory grafiki, procesory specjalizowane itp.).

Języki proceduralne

Język proceduralny pozwala na tworzenie oprogramowania w postaci programu głównego oraz wielu procedur (lub podprogramów), z których każda realizuje określoną funkcję i może być wywoływana wielokrotnie przez program główny.

Każda z procedur pobiera określoną liczbę parametrów oraz opcjonalnie zwraca wartość wynikową w jednej lub wielu zmiennych (procedura), albo jako wartość zwracaną (funkcja).

Istnienie mechanizmu zmiennych lokalnych oraz parametrów wywołania umożliwia zagnieżdżanie odwołań do procedury, co w efekcie pozwala na stosowanie technik rekurencyjnych.

Języki proceduralne wspomagają proces przygotowywania uniwersalnych bibliotek procedur i funkcji, dołączanych do nowo tworzonych programów, co przyspiesza proces tworzenia nowego programu oraz poprawia jakość tworzonych aplikacji.

Języki nieproceduralne

Język nieproceduralny umożliwia pisanie programów jedynie w postaci ciągłej sekwencji instrukcji (niepodzielonej na procedury).

Możliwe jest wykonywanie skoków warunkowych i bezwarunkowych oraz wykonywanie podprogramów, nie istnieje jednak pojęcie parametrów wywołania procedury, wartości zwracanej przez funkcję oraz zmiennych lokalnych i globalnych.

Tworzenie współdzielonych przez wiele programów bibliotek funkcji (do powtórnego wykorzystania) oraz stosowanie takich technik programistycznych jak rekurencja jest znacznie utrudnione lub niemożliwe.

Przykładem języka nieproceduralnego jest **BASIC**.

Języki i środowiska programowania

Języki niskiego poziomu – asemblery procesorów

Języki wysokiego poziomu – **Fortran**, **Prolog**, **LISP**, **Pascal**, **Delphi**, **C**, **C++**, **C#**, **Java**, **UML** itp.

Języki opisu sprzętu – **Abel**, **VHDL**, **Verilog**

Środowiska obliczeń matematycznych i inżynierskich –

Mathematica, **Statistica** , **Maple**, **Matlab**

Środowiska projektowania cyfrowych układów programowalnych –

Foundation ISE, **Webpack ISE** (Xilinx), **Max+Plus II** (Altera)

Hipoteza Sapira-Whorfa 1

Edward Sapir i Benjamin Lee Whorf badali języki Indian. W swoich badaniach zaobserwowali, że różnice występujące pomiędzy prymitywnymi językami naturalnymi a jezykami europejskimi, dotyczące **słownictwa** oraz takich **kategorii gramatycznych** jak *czas* czy *liczba* powodują w konsekwencji różnice w sposobie myślenia.

Przykład: w jezyku *Hopi* zorientowanym na stawanie się (proces) brak jest czasowników odnoszących się do statycznych **relacji przestrzenno-czasowych** (przeszłości, teraźniejszości i przyszłości) – są natomiast czasowniki odnoszące się do **zdarzeń**, **przewidywania** i **uogólniania**.

Wg teorii lingwistycznej Sapira-Whorfa, **używany jezyk (wzory językowe) wpływa w mniejszym lub większym stopniu na sposób (wzory) myślenia**.

Za hipotezą o znacznym udziale języka w myśleniu oraz procesach społecznych przemawiają zmiany językowe i znaczeniowe w praktycznie wszystkich grupach związanych wspólną ideologią. Zmiany te – spontaniczne czy zaprogramowane – wywierają wpływ na sposób myślenia członków grupy.

Hipoteza Sapira-Whorfa 2

W kontekście programowania komputerów ciekawym zagadnieniem są znaczne **różnice w sposobie widzenia problemów** przez programistów używających różnych języków programowania.

Może to mieć pewien związek z hipotezą Sapira–Whorfa i relatywizmem językowym jej autorów.

Używanie jednego **paradygmatu** czy **języka programowania** może pociągać za sobą pewną **jednostronność myślenia** i tendencję do posługiwania się tymi samymi schematami bez względu na to jakie zadanie należy rozwiązać. Pewnym remedium na to zjawisko jest nauka – przynajmniej podstaw – kilku różnych języków programowania, dzięki czemu można patrzeć na to samo zadanie z różnych stron.

Wśród języków szczególnie zalecanych programistom do nauki nieschematycznego myślenia są m.in. **LISP**, **Haskell**, **Prolog**, czy **Oz**.

Programowanie – klasyfikacje 1

Ze względu na rozmiar projektu i charakter pracy programowanie dzielimy na:

indywidualne, zespołowe, interaktywne

W systemach równoległych i rozproszonych, w których występuje wiele komputerów/procesorów/wątków biorących udział w obliczeniach, wyróżniamy rodzaje programowania:

równoległe, **współbieżne, wielowątkowe, rozproszone**

Ze względu sposób podejścia do hierarchii projektu programistycznego wyróżniamy programowanie :

modularne,
zstępujące = analityczne (top-down) i
wstępujące = syntetyczne (bottom-up)

Programowanie – klasyfikacje 2

Ze względu na dominujący paradygmat programowania wyróżniamy programowanie:

proceduralne – kod programu dzielony jest na **procedury**, czyli fragmenty wykonujące ściśle określone operacje; procedury nie powinny korzystać ze zmiennych globalnych, lecz lokalnych oraz pobierać i przekazywać wszystkie dane (czy też wskaźniki do nich) jako parametry wywołania.

strukturalne – programowanie oparte o tzw. instrukcje strukturalne, takie jak sekwencje, instrukcje warunkowe (*if*, *if ... else*), instrukcje wyboru (*case*), pętle (*while*, *repeat*); ograniczone lub wyeliminowane jest używanie instrukcji skoku (*go to*); strukturalność zakłócają instrukcje typu *break*, *continue*, *switch* – stosowane czasem dla zwiększenia czytelności kodu.

Programowanie – klasyfikacje 3

obiektowe – programy definiuje się za pomocą **obiektów** – elementów **wiążących** ze sobą stan (dane, nazywane *polami*) i zachowanie (procedury, czyli *metody*); program obiektowy stanowi zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

(programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów)

uogólnione – pozwala na pisanie kodu programu bez wcześniejszej znajomości typów danych, na których ten kod będzie pracował; przykładowe języki: **C++**, **Java**, **Haskell**

metaprogramowanie – technika pozwalająca programom na tworzenie lub modyfikację własnego kodu (lub kodu innych programów) w trakcie jego kompilacji lub wykonywania; przykładowe języki: **Lisp**, **Smalltalk**, **PHP**, **Python**, **Tcl**, **Perl**.

Programowanie – klasyfikacje 4

W przypadku języków opisu sprzętu (HDL) mamy dwa paradygmaty programowania :

strukturalne – opisywane są komponenty składowe i sposób ich połączenia, a nie zachowanie układu.

behawioralne – w przypadku języków opisu sprzętu opisywane jest zachowanie układu, a nie jego struktura.

Jeżeli występuje programowanie w wielu językach lub paradygmatach równocześnie mamy do czynienia z **programowaniem mieszanym**, np. :

język wysokiego poziomu + asembler

paradygmat strukturalny + behawioralny

Programowanie – klasyfikacje 5

Programowanie **wizualne** (*visual programming*)– technika tworzenia programów przy pomocy narzędzi umożliwiających wybór standardowych elementów z menu i automatyczna generacja odpowiadającego im kodu źródłowego.

Zastosowanie – tworzenie interfejsu użytkownika aplikacji, wspomaganie tworzenia klas obiektów, podział na funkcje i procedury, generowanie dokumentacji.

Przykłady jezyków: **Borland Delphi, Symantec C++, Visual C++**

Programowanie **szablonowe** (*template programming*) – szablony umożliwiają tworzenie kodu bez uwzględniania typów. Szablonów używa się tworząc instancje (konkretyzacja).

W **C++** popularne są szablony funkcji i klasy.

Java, **C#**, **Haskell** mają podobne rozwiązanie – typy generyczne.

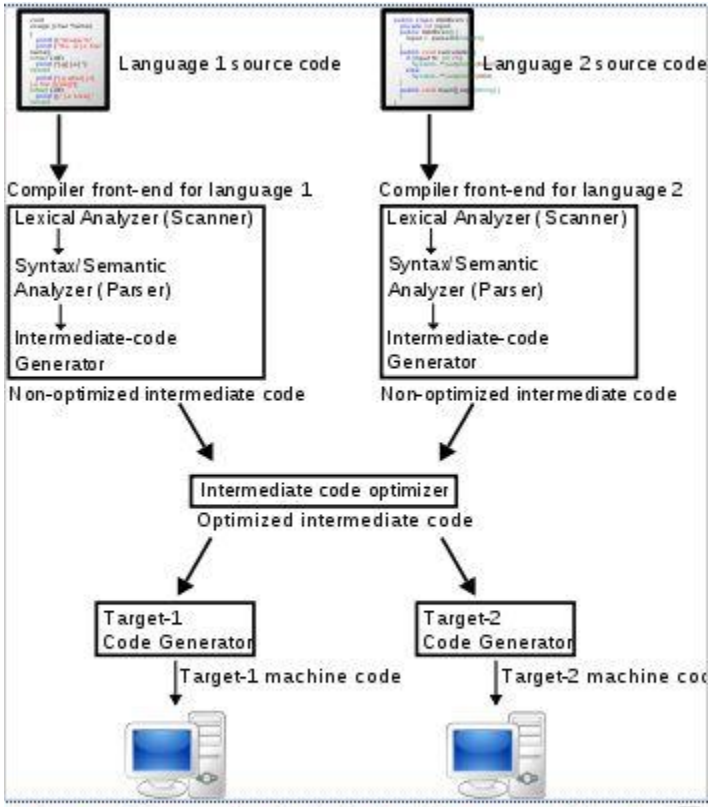
W **VHDL** istnieje biblioteka szablonów typowych układów cyfrowych – potrzebna jest konkretyzacja (uwzględnienie rozmiaru układu, nazw).

Kompilacja kodu źródłowego

Kompilacja : kod źródłowy jest zwykle kompilowany do kodu maszynowego, wykonywalnego na danej maszynie. Kompilacja składa się z kilku faz: analizy leksykalnej, analizy syntaktycznej i semantycznej, generacji kodu pośredniego, optymalizacji tego kodu oraz generacji kodu wynikowego.

Procesem odwrotnym do kompilacji jest **dekompilacja.** W wyniku dekompilacji z kodu wynikowego odtwarzany jest kod źródłowy. Praktykę taką nazywa się **reverse engineering.**

Oprócz kompilacji programowej wyróżnia się ponadto **kompilację sprzętową**, np. w celu skonfigurowania układu programowalnego FPGA.



A diagram of the operation of a typical multi-language, multi-target compiler.

Wymagania jakościowe 1

Programy, jako wynik działalności programistycznej, powinny spełniać następujące wymagania o charakterze jakościowym:

Efektywność – wydajność : ilość zasobów systemu komputerowego wymagana przez program (czas pracy procesora, zajmowany obszar pamięci, przepustowość sieci oraz w pewnym zakresie również intensywność komunikacji z użytkownikiem) – powinna być jak najmniejsza.

Niezawodność : jak często wyniki generowane przez program są poprawne; zależy od zapobiegania propagacji błędów pochodzących z konwersji danych, oraz zapobiegania błędom pochodzącym z przepełnionych buforów, dzielenia przez zero itp.

Odporność : jak dobrze program jest w stanie przewidzieć sytuacje konfliktu typu danych i inne rodzaje niekompatybilności, które skutkują błędami wykonania programu lub jego zawieszeniem. Program powinien zapewniać w takich sytuacjach właściwy kontakt z użytkownikiem i obsługę wyjątków.

Wymagania jakościowe 2

Użyteczność : czytelność i intuicyjność programów decyduje o ich akceptacji przez docelowego użytkownika i sukcesie rynkowym. Interfejs użytkownika obejmuje zarówno ogólną koncepcję jak i szeroką paletę elementów tekstowych i graficznych, które powodują, że program jest łatwy i wygodny w użytkowaniu.

Przenośność : zakres platform sprzętowych i systemów operacyjnych, na których może być kompilowany, interpretowany i wykonywany kod źródłowy programu. Przenośność zależy przede wszystkim od dostępnych na danej platformie kompilatorów, a nie od samego programu.

Inne związane z tymi wymaganiami cechy programów są następujące: **paradygmat programowania**, **styl programowania**, niska **złożoność obliczeniowa** implementowanego algorytmu, **testowalność programu**, potencjalna **wielokrotna używalność** (*re-usability*), **zwięzłość** i **dokumentacja**.

Metody programowania

Rekursja

Metoda dziel i zwyciężaj (*divide and conquer)*

Równoważenie (*balancing*)

Programowanie z nawrotami (*backtracking*)

Wyszukiwanie wyczerpujące (*exhaustive search*)

Wyszukiwanie lokalne (*local search*)

Programowanie dynamiczne

Programowanie liniowe (optymalizacja)

Programowanie zachłanne (*greedy*)

Metaheurystyki (algorytmy iteracyjne)

Programowanie równoległe *(parallel programming)*

Literatura

Literatura podstawowa

Cormen T.H., Leiserson C.E., Rievest R.L., : Wprowadzenie do algorytmów, WNT, Warszawa 2002
Kernighan B.W., Pike R. : Lekcja programowania, WNT, Warszawa 2002
Lipski W. : Kombinatoryka dla programistów, wyd.3. rozsz., WNT 2004
Sedgewick R. : Algorytmy w C++, Wyd. RM/Addison-Wesley, 1999

Literatura dodatkowa

Bentley J. : Perełki oprogramowania, WNT, Warszawa 1992
Hunt A., Thomas D. : Pragmatyczny programista, WNT 2002
Knuth D. : Sztuka programowania, t. 1–3, WNT, Warszawa 2002
MacVittie D.W., MacVittie L.A. : Programowanie zorientowane obiektowo. Nowy sposób myślenia, MIKOM, Warszawa 1996
Malina W., Szwoch M. : Techniki programowania, Wyd. PG, Gdańsk 2001
Van Roy P., Haridi S. : Programowanie, koncepcje techniki modele, Helion/MIT Press 2005