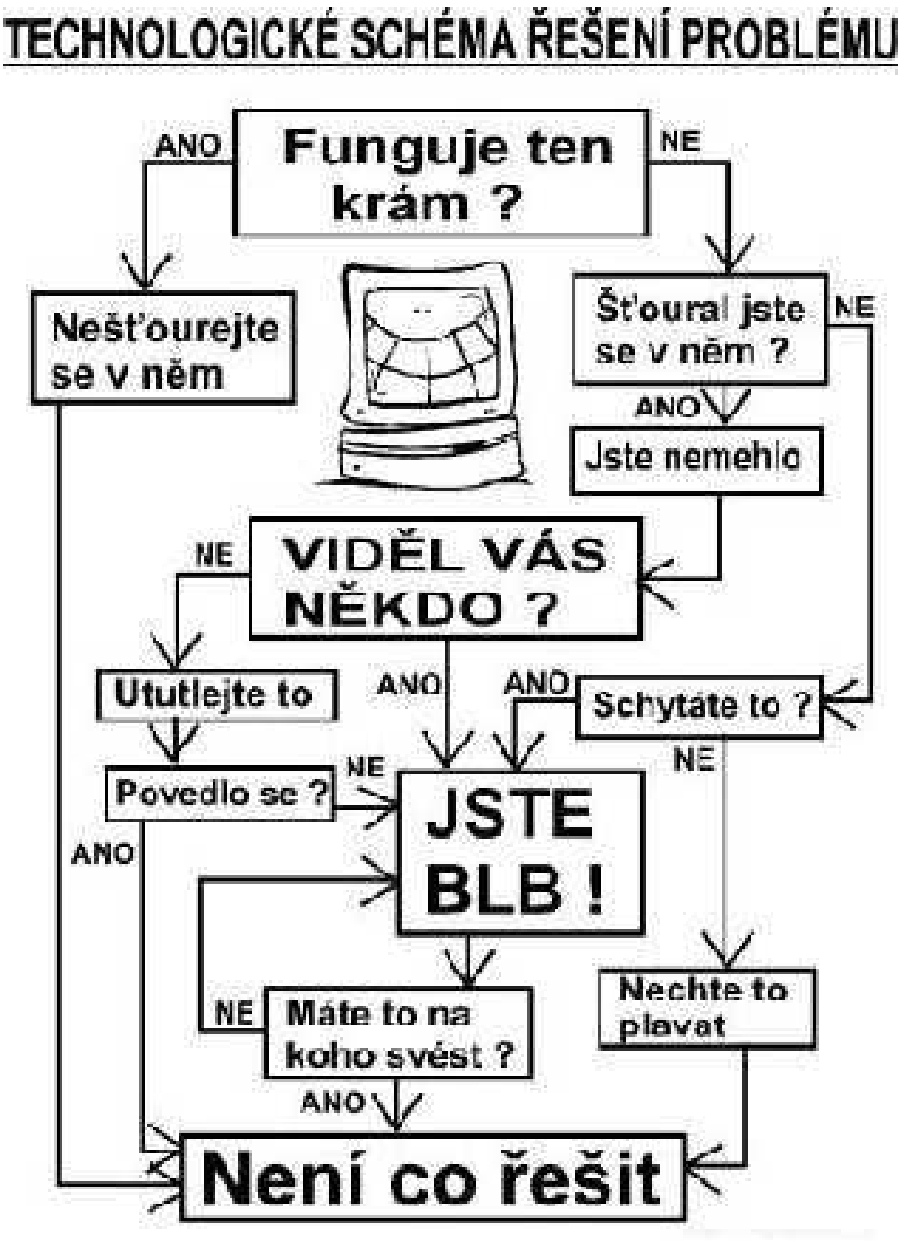
- **Algoritmus** (nebo dřívějším pravopisem algorithmus)
  - přesný návod či postup, kterým lze vyřešit daný typ úlohy. Pojem algoritmu se nejčastěji objevuje při programování, kdy se jím myslí teoretický princip řešení problému (oproti přesnému zápisu v konkrétním programovacím jazyce). Obecně se ale algoritmus může objevit v jakémkoli jiném vědeckém odvětví. Jako jistý druh algoritmu se může chápat i např. kuchyňský recept. V užším smyslu se slovem algoritmus rozumí pouze takové postupy, které splňují některé silnější požadavky:

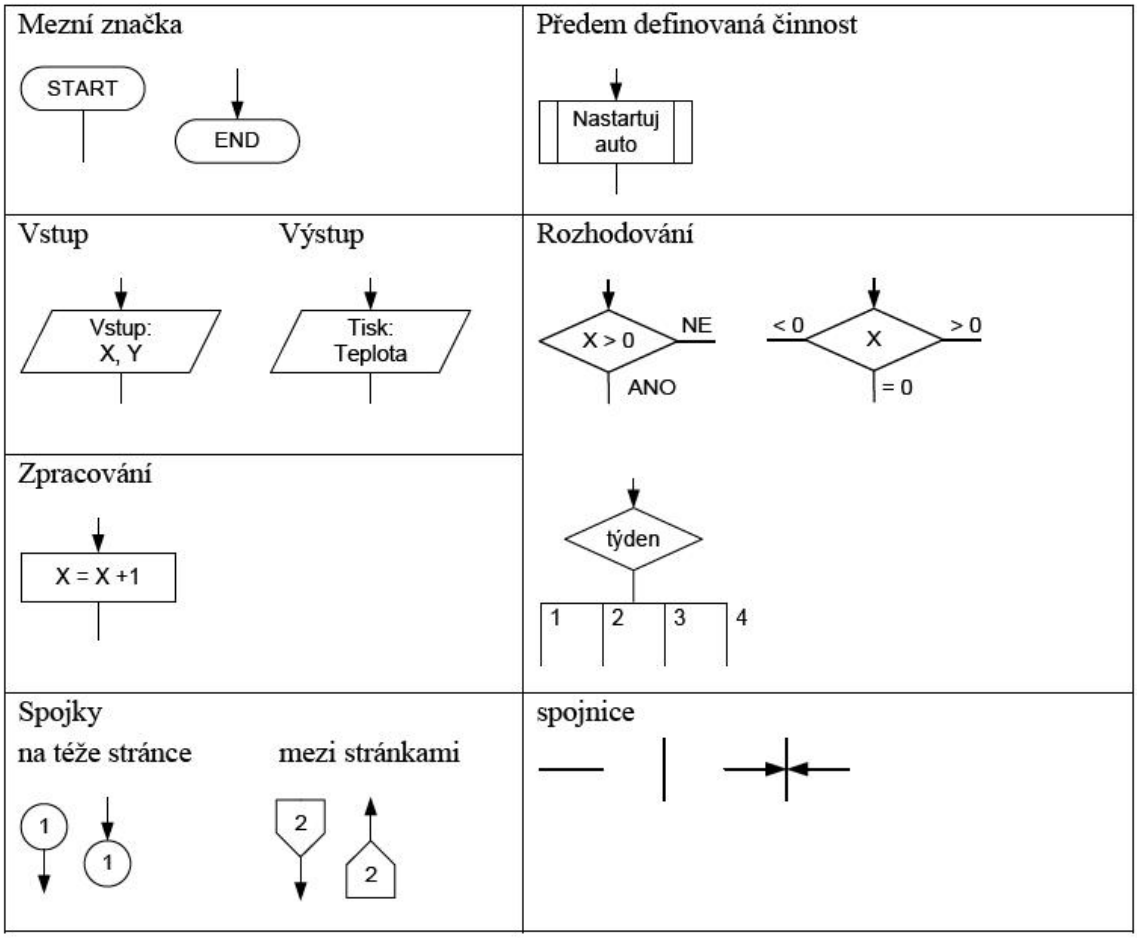
- **Konečnost**
  - Každý algoritmus musí skončit v konečném počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný.
- **Determinovanost**
  - Každý krok algoritmu musí být jednoznačně a přesně definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat. Protože běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly pro zápis algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný význam. Vyjádření výpočetní metody v programovacím jazyce se nazývá program.
- **Vstup**
  - Algoritmus obvykle pracuje s nějakými vstupy, veličinami, které jsou mu předány před započetím jeho provádění, nebo v průběhu jeho činnosti. Vstupy mají definované množiny hodnot, jichž mohou nabývat.
- **Výstup**
  - Algoritmus má alespoň jeden výstup, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší. (Algoritmus vede od zpracování hodnot k výstupu - Resultativnost)
- **Efektivita**
  - Obecně požadujeme, aby algoritmus byl efektivní, v tom smyslu, že požadujeme, aby každá operace požadovaná algoritmem, byla dostatečně jednoduchá na to, aby mohla být alespoň v principu provedena v konečném čase pouze s použitím tužky a papíru. (tj. byla elementární)
- **Obecnost (Hromadnost)**
  - Algoritmus neřeší jeden konkrétní problém (např. „jak spočítat 3×7“), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel“).



Algoritmus má 1 důležitou funkční chybu?

Úvod do vývoje programů (Teď už vážně)

- vývojový diagram – jedna z metod zápisu postupu řešení nějakého problému (algoritmus)
- pro srozumitelnost se používají zavedené symboly, není normalizováno, takže se můžeme setkat z různými variantami, ale většinou odvozené nebo podobné.



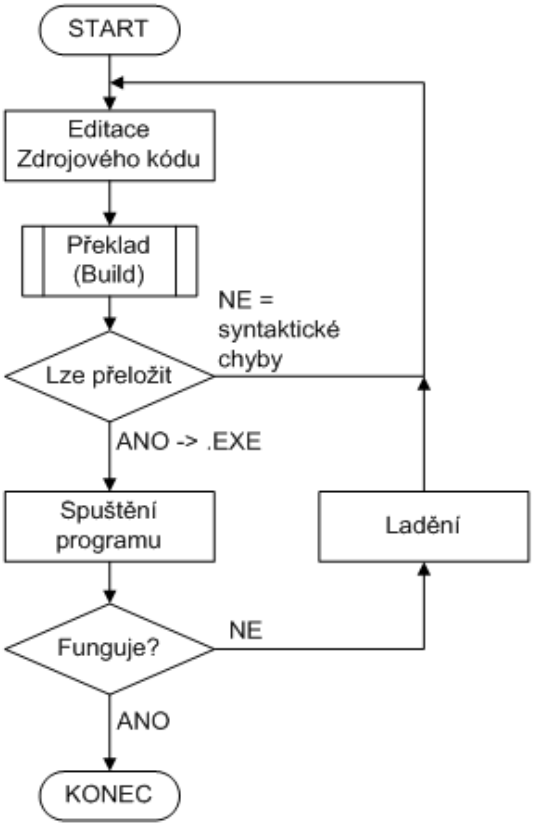
- Jednotlivé bloky se spojují spojnicemi, při zápisu se snažíme držet směr z vrchu dolů a z leva do prava.

- Vývojové diagramy dobře slouží k rozmyšlení způsobu jakým budeme pomocí programování řešit, nebo k nastínění řešení někomu jinému, aniž by jsme použily konkrétní programovací jazyk – řešení zůstává čitelné pro více lidí.

- Dobrý programátor neznamená umět dobře programovací jazyk/jazyky, ale hlavně schopnost nalézt řešení problému a to pokud možno ne jedno, ale více a umět i mezi nimi vybírat. Žádné řešení většinou není ideální, jen je více / méně efektivní.

- Nakonec stejně není až tak důležitá cesta, ale cíl.

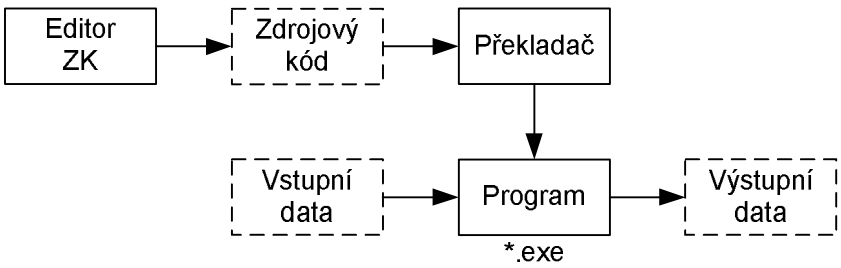
- vývojový diagram postupu při vývoji programu (od zápisu ve VPJ do jeho funkční podoby)



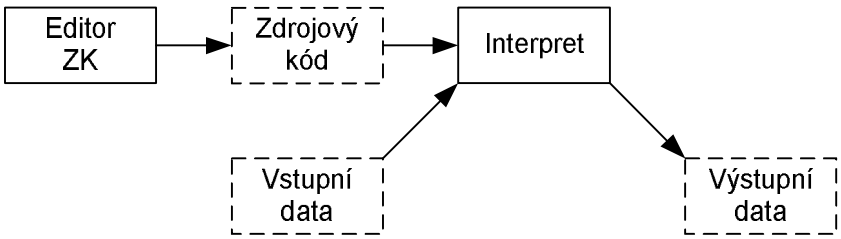
  - Zdrojový kód = textový soubor s programem v PJ
  - Překlad = převod do strojového kódu, provádí překladač (Compiler)
  - Ladění = hledání a odstraňování logických a běhových chyb, různé techniky
    - Logická chyba: Proč to počítá 2+2=6, netiskne, apod.
    - Běhová chyba: /0, zápis do paměti, která není naše apod. – program se hroutí
    - chyba v programu = bug
    - ladění = debugging
    - ladící program = Debugger
  - Toto = kompilační implementace PJ

Kompilační vs. interpretované jazyky

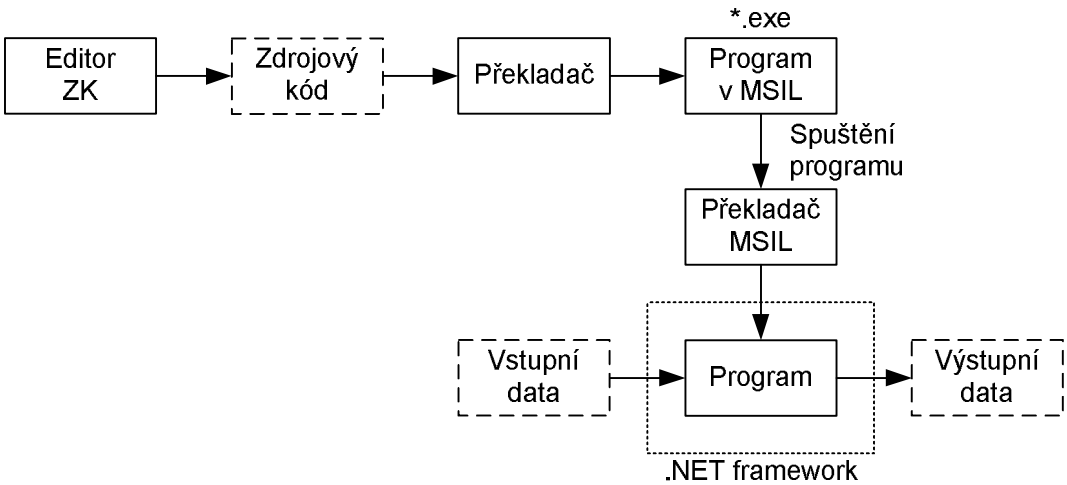
- kompilační implementace PJ



  - např. C, Pascal, C++
  - cílový program = „nativní *.exe“ → obsahuje instrukce CPU
  - nejrychlejší, nepřenositelné
- interpretační implementace PJ



  - interpret → „on-line“ překládá příkazy PJ do instrukcí
  - pomalé, přenositelné
  - např. Basic, PHP, Java Script
- kombinace
  - Např. C# (.NET), Java



  - MSIL = Microsoft Intermediate Language
  - Relativně rychlé, relativně přenositelné (lze spustit všude, kde je .NET)

Vývojové prostředky

Editory zdrojových kódů

= jednoduchý textový editor (nemá pokročilé formátovací funkce jako např. MS Word), ale navíc:

- Zvýrazňování syntaxe (syntax highlight)

- Formátování kódu (code formating)
- Automatické doplňování jmen apod. (code completion)

Překladače

- obvykle ovládány přes příkazový řádek velkou sadou různých přepínačů

debuggery

- umožňují:
  - Zastavení programu
  - Krokování
  - Prohlížení vnitřních stavů („načetla se ta 3 z klávesnice?“)

Integrované vývojové prostředí

- = integrace editoru zdrojových kódů, ovládání překladače, debuggeru a dalších funkcí do jednoho jednoho programu
- IDE – Integrated Development Envrironment
- Např.:
  - Borland C (C++) – jen pro DOS
  - Borland C++ Builder
  - Microsoft Visual Studio
  - KDevelop (Linux)

VS 2005 IDE

Orientace a manipulace s IDE

- Založení projektu
- Okna, Připínáčky
- Solution explorer SE → důležité přesvědčit se, že jste na program.cs ze SE
- Uložení, co je kde na disku

Zápis ZK

- Doplňování kódu
- Code snippets: cw = `Console.WriteLine()`

Překlad

- Build Solution, project – rozdíl
- Hledání a oprava syntaktických chyb
- Co je výsledkem, kde to je, kdy to jde spustit (nutný .NET)

Nápověda – MSDN

- Microsoft Developer Network
- = Nápověda
- Dostupná on-line na http://msdn2.microsoft.com/en-us/default.aspx

- Zejména http://msdn2.microsoft.com/en-us/library/kx37x362(VS.80).aspx

Samostatné příklady

Příklad č. 1

Napište program, který na obrazovku vytiskne adresu pana Nováka:

```
Josef Novak
Jindrisska 16
111 50, Praha 1
```

Adresa bude vytištěna dvakrát, tisky budou odděleny prázdným řádkem:

- Jednou pomocí několika příkazů (tisk každého řádku zvlášť)
- Podruhé pomocí jediného příkazu – vyzkoušejte, co se stane, pokud do tisknutého textu vložíte řetězec `\n`

Výstup aplikace bude vypadat jako na obrázku: