Program a životní cyklus programu

- **Program**
  - algoritmus zapsaný formálně, srozumitelně pro počítač
  - program se skládá z elementárních kroků

- **Elementární kroky** mohou být:
  - instrukce operačního kódu počítače
  - příkazy nějakého **programovacího jazyka**
  - již hotové úseky kódu nebo podprogramy

Životní cyklus programu

- **Specifikace požadavků na řešený úkol**
  - *„Uživatel neví, co chce, ale běda, když to nedostane!“*
  - vyjasnit rozpory, neúplnosti a nejasnosti zadání
  - vyjasnit reakci na nepřípustná data
  - umění porozumět uživateli → **systémový integrátor**

- **Dekompozice úkolu na části**
  - hierarchický rozklad úlohy na části řešitelné jedním pracovníkem nebo týmem

- **Návrh algoritmu(ů)**
  - návrh strukturovaných algoritmů nezávislých na konečné implementaci
  - určení vazeb mezi jednotlivými částmi

- **Implementace algoritmu na program**
  - zjemňování návrhu – **CASE nástroje** (*Computer Aided Software Engineering*)
  - zdrojový text v **programovacím jazyce**
  - překlad zdrojového textu do kódu počítače → **spustitelný program**

- **Testování programu**
  - zkoušení jednotlivých komponent a celku
  - odhalování chyb prováděním programu na vhodně zvolených testovacích datech
  - ***„Zkoušej a prověřuj okamžitě, jakmile je co zkoušet !“***
  - Prověření v rámci celého systému u uživatele (**validace**)

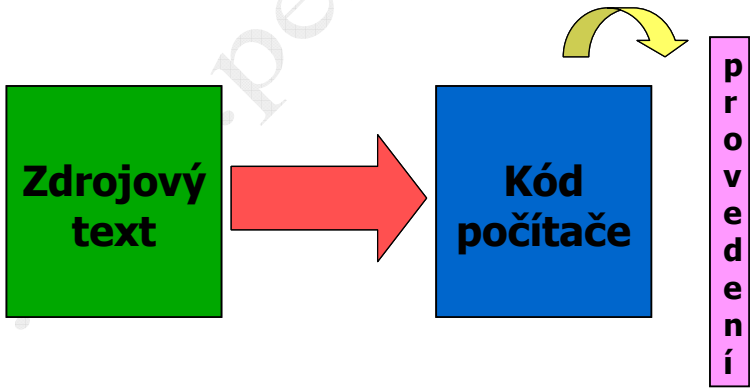
- **Používání a údržba programu**
  - odstranění skrytých vad
  - potřeba reakce na změny okolí (jiný hardware, jiný OS, změny v legislativě, organizační struktuře, …)
  - **Nutnost pečlivé dokumentace ve všech fázích životního cyklu!**

Programovací jazyky

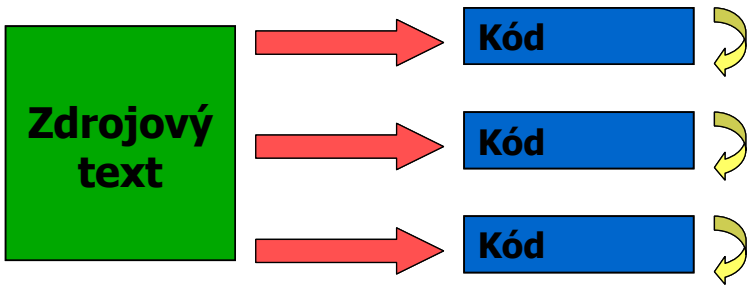
- **Programovací jazyk**
  - speciální jazyk, ve kterém programátor dokáže snadno vyjadřovat algoritmy a který počítač bez větších problémů pochopí – pomocí **překladače**
- **Syntaxe**
  - soubor pravidel, které určují, jaké zápisy (konstrukce) jsou v daném jazyce povolené
- **Sémantika**
  - soubor pravidel popisujících význam povolených konstrukcí
- **Lexika**
  - definice přípustných symbolů
- **Pragmatika**
  - určuje správný význam

Překlad zdrojového programu do kódu počítače

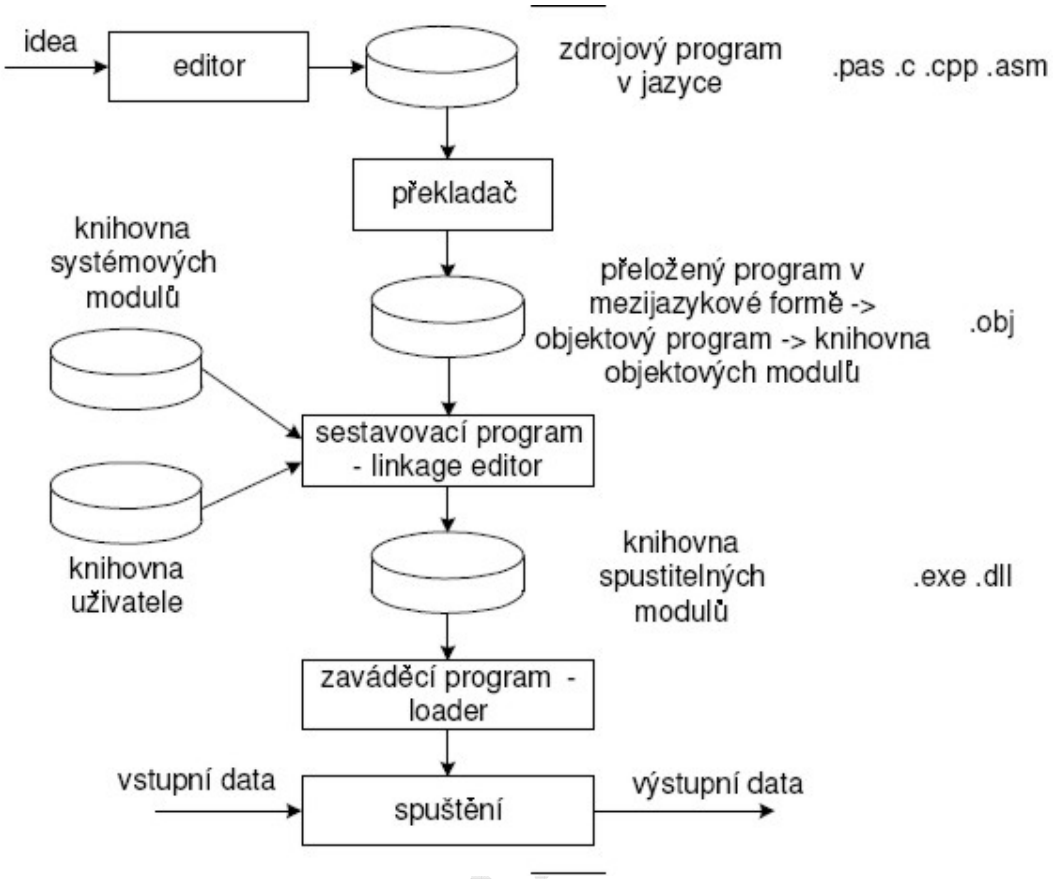
- **Kompilátor**
  - překlad celého zdrojového textu najednou
  - většinou překlad v několika fázích (průchodech)
  - výsledkem je program v kódu počítače
  - překlad se provádí jen jednou – znovu pouze pokud byl program měněn



- **Interpret**
  - překlad po částech (příkazech, skupinách)
  - přeložená část se ihned provede
  - snadnější kontrola sémantiky, avšak mnohem vyšší nároky na čas



- **Struktura překladu:**



Programovací jazyky

- **Neprocedurální**
  - popisují **co** se má řešit, ale **ne jak** dojít k výsledku
  - algoritmus řešení nepřipravuje autor

- **Procedurální - imperativní**
  - popisují algoritmus, tj. **jak** řešit úlohu – rozklad na dílčí kroky (příkazy)
  - jednotlivé kroky více či méně vzdálené kódu počítače

Neprocedurální programovací jazyky

- **Obecně orientované**
  - široký okruh řešených úloh **x** omezené možnosti řešení
  - funkcionální a logické jazyky – **LISP**, **PROLOG**

- **Specializované**
  - úzký okruh řešených úloh **x** účinnější a rychlejší možnosti řešení
  - dotazovací jazyky databází – **SQL**

Procedurální programovací jazyky

- **Strojově orientované jazyky**
  - mnemotechnické názvy instrukcí kódu počítače
  - řada makroinstrukcí (maker)
  - velká závislost na konkrétním počítači
  - **assembler**

- **Vyšší programovací jazyky**
  - pohodlnější a bezpečnější programování
  - větší nezávislost na počítači – bližší člověku
  - nutnost překladu do kódu počítače
  - součástí je deklarace struktury dat
  - **FORTRAN**, **COBOL**, **PASCAL**, **C**, …

- **Objektově orientované jazyky**
  - popisují model světa pomocí tříd objektů, které mezi sebou komunikují posíláním zpráv
  - **třída** = popis datového typu včetně metod
  - **objekt** = konkrétní instance třídy
  - změna stavu objektu provedením metody
  - **provedení programu** = změna stavu objektů a jejich vzájemná komunikace
  - **SMALLTALK**, **OBJECT PASCAL**, **C++**, …

Implementace algoritmu

Fáze implementace algoritmu

- **Úprava (doplnění) algoritmu**
  - pro konkrétní programovací jazyk

- **Realizace základních struktur**
  - sekvence, selekce, iterace ve zvoleném jazyce

- **Překlad do kódu počítače**
  - interpret **x** kompilátor

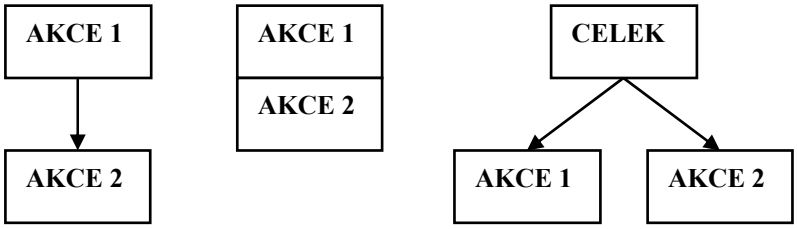
Úprava algoritmu

- **Čím nižší programovací jazyk, tím větší úpravy algoritmu**

- **Úprava podmínek** (assembler)
  - jsou k dispozici pouze rozhodování na základě porovnání střadače (S) s nulou
    - S = 0, S < 0, S > 0
  - **výraz > (<, =) hodnota** I < 500
    - S := výraz – hodnota S := I - 500
    - S > (<, =) 0 S < 0

Realizace základních struktur

- Realizace základních struktur v **cvičném strojovém kódu** (assembleru) a **Turbo Pascalu**
- **Sekvence** – v pořadí: *vývojový diagram*, *plošný strukturogram*, *strukturogram*



  - **Assembler**

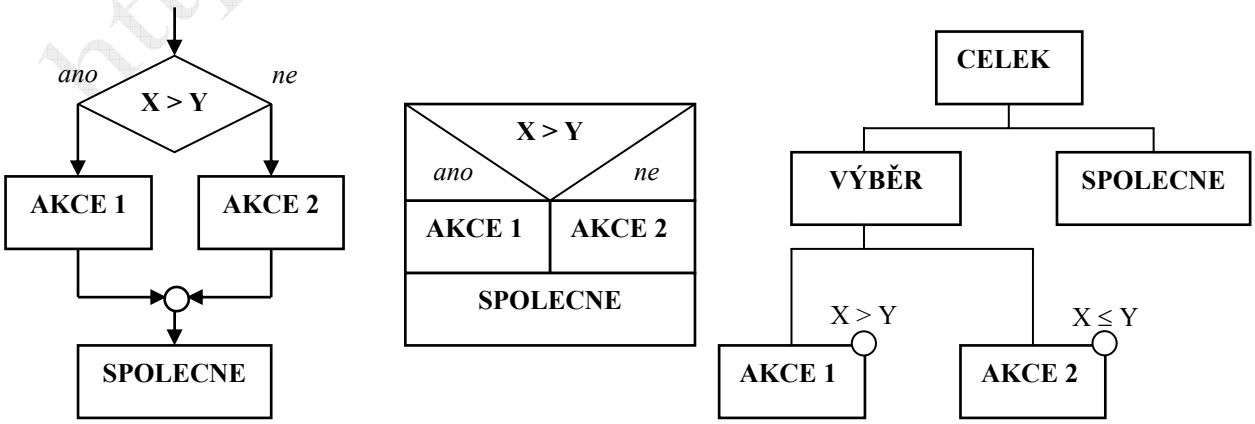
```
100: AKCE_1
101: AKCE_2
102: HLT
```

    - **Formát instrukce:** **adresa:** **návěští:** **instrukce**
      `200:` `START:` `LDA 100`
    - Každý program v assembleru musí končit instrukcí **`HLT`**, jinak by procesor údaje (většinou náhodné údaje) uložené na následujících adresách chápal jako instrukce programu a snažil by se je vykonávat → může vést ke **kolapsu**.

  - **Turbo Pascal**

```
begin
      AKCE_1;
      AKCE_2;
      ...
end
```

- **Selekce** – v pořadí: *vývojový diagram*, *plošný strukturogram*, *strukturogram*



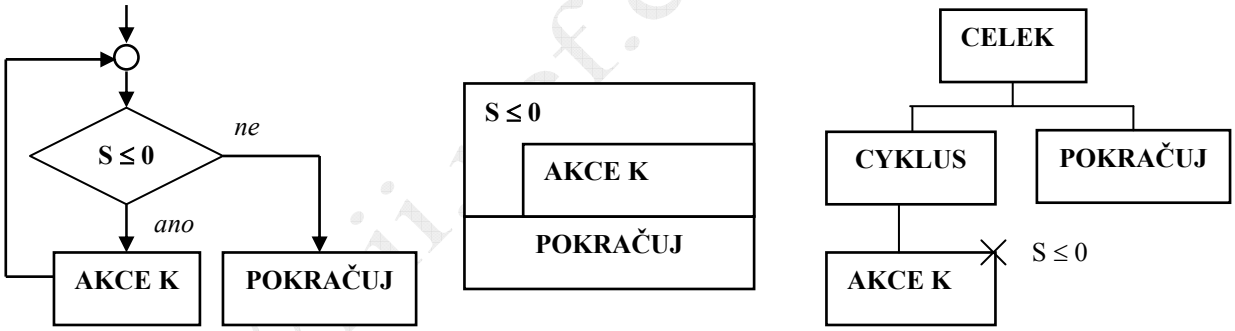
Assembler

```
100: VYBER:     JP AKCE_A
101:            AKCE_B
102:            JMP SPOL
103: AKCE_A:    AKCE_A
104: SPOL:      SPOLECNE
105:            HLT
```

Turbo Pascal

```
if  X > Y then
     begin
          AKCE_A
     end
else
     begin
          AKCE_B
     end;
Spolecne
```

Iterace s podmínkou na začátku – v pořadí: *vývojový diagram*, *plošný strukturogram*, *strukturogram*



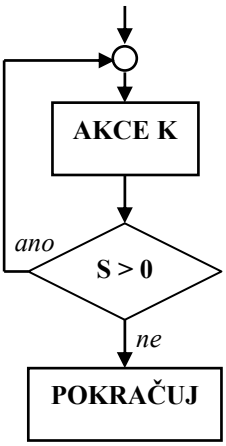
Assembler

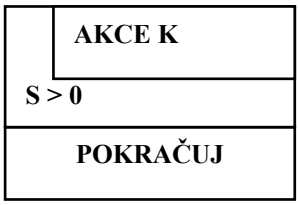
```
100: CYKL:      JP POKRAC
101:            AKCE_K
102:            JMP CYKL
103: POKRAC:    POKRAČUJ
104:            HLT
```

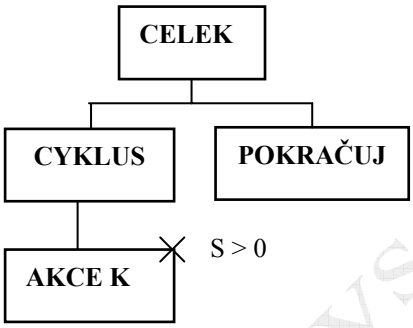
Turbo Pascal

```
while S ≤ 0 do
     begin
          AKCE_K
     end;
Pokracuj
```

- **Iterace s podmínkou na konci** – v pořadí: *vývojový diagram*, *plošný strukturogram*, *strukturogram*







Assembler

```
100: CYKL:      AKCE_K
101:            JN POKRAC
102:            JZ POKRAC
103:            JMP CYKL
104: POKRAC:    POKRAČUJ
105:            HLT
```

Turbo Pascal

```
repeat
     begin
          AKCE_K
     end
until S <= 0;
Pokracuj
```

- Za klíčovým slovem **until** („dokud ne“) je třeba uvést **negaci podmínky** uvedené ve vývojovém diagramu.

Příklad

- Algoritmicky realizujte příkaz **MOD A,B** a **DIV A,B**, kde MOD bude obsahovat zbytek po celočíselném dělení čísla A číslem B a kde DIV bude obsahovat výsledek celočíselného dělení čísla A číslem B. A a B jsou celá kladná čísla.

  - *Vstup:* A, B – dvě celá kladná čísla
  - *Výstup:* DIV – celočíselný podíl A a B
    MOD – zbytek po celočíselném dělení A a B

- **Postup:**

  - Od čísla A odečítáme číslo B tak dlouho, dokud není zbytek menší než číslo B. Tento zbytek je pak hodnotou MOD.
  - Je možné využít předchozí příklad, kde výsledek (hodnotu DIV) získáme jako počet průchodů cyklem.

- **Plošný strukturogram algoritmu:**

Čti A, B	
MOD := A	
DIV := 0	
dokud MOD ≥ B (α)	
	MOD := MOD – B **DIV := DIV + 1**
piš DIV, MOD	

- **Testovací tabulka pro hodnoty A=25 a B=6:**

krok	A	B	MOD	DIV	α
1	25	6	25	0	Ano
2	25	6	19	1	Ano
3	25	6	13	2	Ano
4	25	6	7	3	Ano
5	25	6	1	4	Ne ! konec

Implementace algoritmu v programovacím jazyku:

cvičný strojový kód (assembler)

```
200:   START:    LNA 0
201:             STA DIV
202:             LDA A
203:             STA MOD
204:   CYKL:     LDA MOD
205:             SUB B
206:             JN KONEC
207:   TELO:     STA MOD
208:             LNA 1
209:             ADD DIV
210:             STA DIV
211:             JMP CYKL
212:   KONEC:    HLT

100: A:
101: B:
102: MOD:
103: DIV:
```

jazyk Turbo Pascal

```
program DIVMOD;

var A,B,MOD_AB,DIV_AB: integer;

begin
  DIV_AB:=0;
  write('A=');readln(A);
  write('B=');readln(B);
  MOD_AB:=A;
  while (MOD_AB>=B) do begin
    MOD_AB:=MOD_AB-B;
    DIV_AB:=DIV_AB+1;
  end;
  writeln('DIV_AB=',DIV_AB);
end.
```