ROBOTI

VE ŠKOLE PRO PRAKTICKOU VÝUKU, MOTIVACI I ZÁBAVU

CZ.1.07/1.1.24/01.0066

ZÁKLADY PROGRAMOVÁNÍ

Co je vhodné vědět, než si vybereme programovací jazyk a začneme programovat roboty.

Mgr. Vladislav BEDNÁŘ

Střední škola elektrotechnická, Ostrava, Na Jízdárně 30, příspěvková organizace

2014

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
střední škola elektrotechnická SŠE
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

OBSAH:

ÚVOD

Tyto učební texty vznikly jako podpora pro programování robotů. Nekladou si za cíl orientaci na konkrétní programovací jazyk. Na konkrétní programovací jazyk je již napsáno množství kvalitních výukových skript a na internetu lze nalézt dostatek návodů a diskuzních fór. Tento text ukazuje problematiku programování. Měl by pomoci uživatelům v orientaci při rozhodování, v jakém jazyku začít programovat a ukázat souvislosti, s kterými se setkáváme při programování.

Uživatel (budoucí programátor) musí zvládnout definovat své požadavky a pokud možno formulovat je v podobě vývojového diagramu. Potom by měl být schopen toto ve vybraném programovacím jazyce převést na algoritmus, který bude provádět požadovanou činnost.

Začínající uživatelé se seznámí s možností, že lze vytvářet program pro programování robota, aniž by museli znát konkrétní programovací jazyk. Existuje možnost pomocí reverzní kinematiky vytvářet programy, kde se při mechanických pohybech definují pouze začáteční a koncové pozice. Uživatel pracuje s hotovým mechanickým modelem. Animátor ručně pohybuje s pohyblivými segmenty (klouby, údy) robota. Počítač zaznamenává požadované kloubové pozice a sám provede potřebné výpočty tak, aby robot mohl vytvořit plynulý 3D pohyb, který je vyjádřen buď počítačovou 3D animací nebo již skutečným pohybem mechanického robota.

Obrázek 1 Ukázka robotu ze stavebnice Robotis Bioloid [25]

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

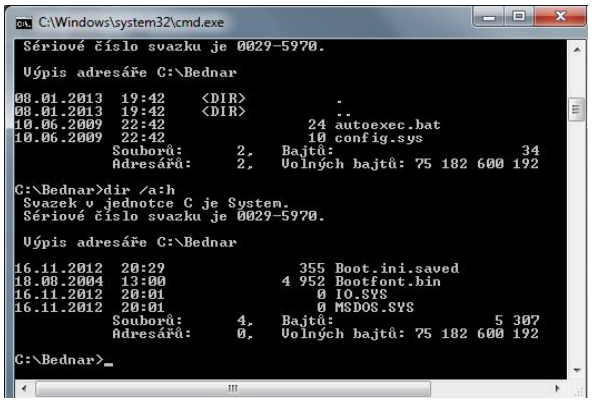
http://www.robotis.com/xe/
http://coptel.coptkm.cz/index.php?action=2&doc=7876&docGroup=4873&cmd=0&instance=2
http://www.megarobot.cz/cj/manualy/robotis/bioloid/Stavebnice%20BIOLOID.pdf
http://www.megarobot.cz/cj/manualy/robotis/bioloid/ExpertKit_aj.pdf
http://wikipedia.infostar.cz/i/in/inverse_kinematic_animation.html

ČAS POTŘEBNÝ KE STUDIU 200 minut

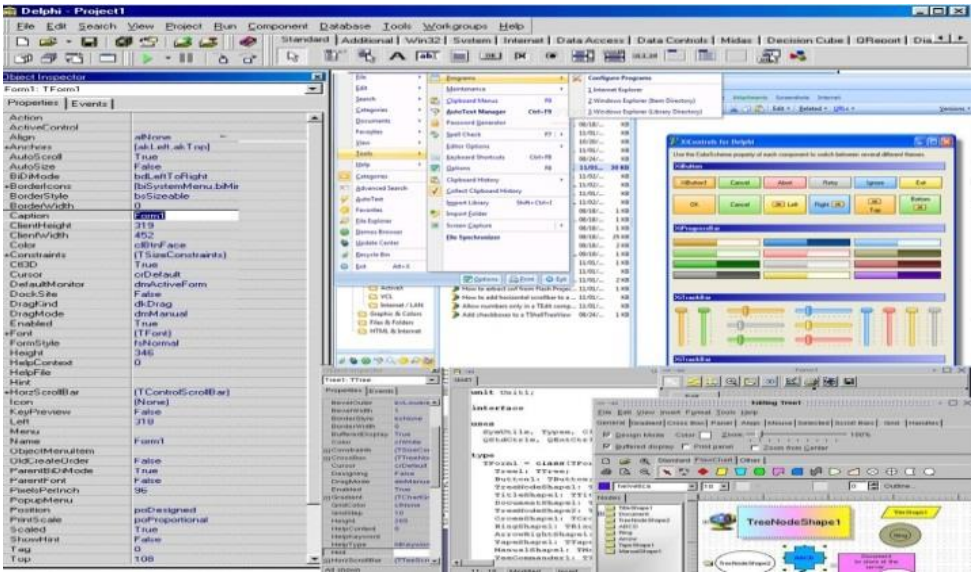
CÍL: ujasnění co je programátor a co programování.

1. PROGRAMOVÁNÍ A JEHO ÚČEL

Programováním rozumíme posloupnost kroků, které vedou k vyřešení určitého zadaného úkolu pomocí programovacího jazyku. Jde tedy o posloupnost příkazů přesně definovaných v určitém programovacím jazyku. Programovací jazyky i vlastní programování se vyvíjelo a stále vyvíjí. Základní rozdělení můžeme definovat jako procedurální či neprocedurální programování. Jiný způsob dělení spočívá v rozdělení na programování objektové či příkazové. Programátory můžeme taktéž pomyslně rozdělit na programátory systémové a aplikační. Systémoví programátoři pracují ve větších kolektivech (individualita zde již v dnešní době nemá místo) a jejich výstupem jsou hlavně operační systémy (Windovs, UNIX), popřípadě databázové systémy, či jiné specializované platformy (může jít třeba i o nástavbu jako je JAVA). Výraz aplikační programátor vznikl tak, že tito programátoři vyvíjeli (programovali) programy, které fungovaly nad konkrétním operačním systémem. Dřívější programátoři pracovali pouze v textovém režimu (tento režim je takový, jako když si v prostředí Windows zapneme příkazový řádek). Toto programování bylo značně zdlouhavé. V dnešní době kombinujeme jak prvky textové, tak prvky grafických nástaveb.

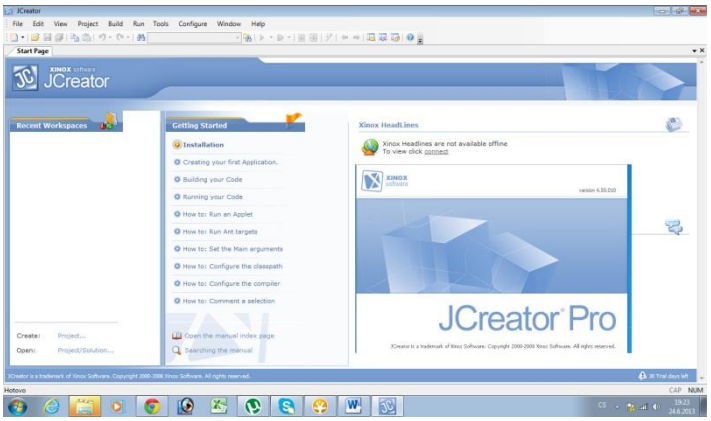


Obrázek 3 Klasický dosovský příkazový řádek [1]

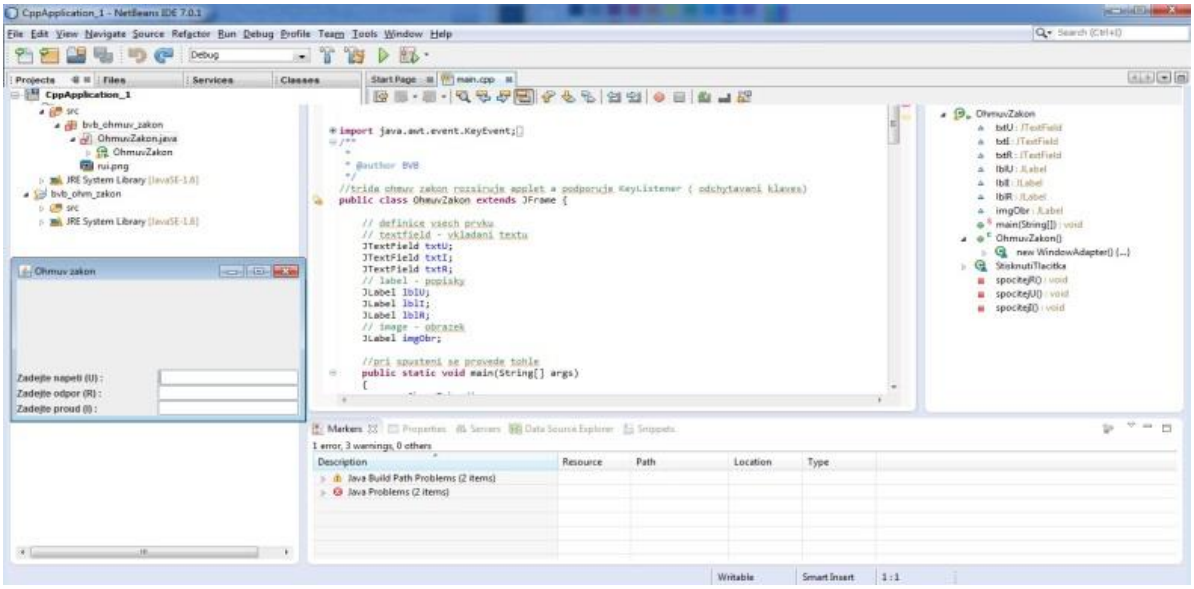


Obrázek 2 Grafické rozhraní DELPHI [1]

Pro programování v Javě se používá velmi rozšířené programování ve volně šířitelném rozhraní Eclipse nebo NetBeans IDE, které podporují vývoj aplikací nejen v Java ale i v C/C++ , Groovy, JavaScript, PHP, Python, Ruby, UML, HTML, XML.



Obrázek 4 Vývojové prostředí JCreator Pro



Obrázek 5 Programovací rozhraní NetBeans IDE 7.0.1 [1]

1.1. Posloupnost činností a program

Samozřejmě člověk vykonává posloupnost činností, které má již naučené (navyklé) a většinou si ani neuvědomuje, že vlastně vykonává předem připravený (osvojený) program.

Tyto postupy byly přejímány (učeny) automaticky od dřívějších generací. Předávání těchto sekvenčních (vlastně programových) kroků postupovalo nejdříve ústním podáním a později zápisovou formou. Nemusí se jednat jen o řeč, ale můžeme používat i dopředu dohodnuté symboly, které skupina lidí daného okruhu používá (každý obor má svá specifika).

Samozřejmě i prosté manuální kopírování výrobního (pracovního) postupu je ve své podstatě program. Již dřívější lovec využíval osvědčené postupy při lovu či zpracovávání ulovených zvířat, které se v této společnosti lovců předávaly spíše vizuální ukázkou. Již tady můžeme hovořit o tvorbě základního programu. Některé kroky (činnosti) se daly přehodit

nebo vykonávat současně. Některé byly dány, jelikož byly a jsou v logických návazných souvislostech.

Typickou ukázkou je třeba kuchařská kniha. V každém receptu máme dané vstupy (suroviny) a je dán taktéž postup, co s těmito surovinami mámě dělat. Vypuštění, či změna některých operací většinou směřuje k nezdaru. V kuchařské knize máme popisy slovní. Setkáváme se při vaření taktéž se symbolickým vyjádřením pracovního postupu (výrobce nás vlastně programuje k určitým činnostem) pomocí symbolů, kterým většinou rozumíme, i když dotyčný návod není v naší rodné řeči. Využívají se zde symboly pro množství vody, míchání, vaření, a to většinou i s doprovodnou informací a době, kterou potřebný úkon máme provádět (jak dlouho máme protřepávat, či doba varu). V oborech ovlivněných výpočetní technikou se ustálilo používání určitých symbolů pro grafické znázorňování prováděných operací (či možných operací). Tuto symboliku převzali i programátoři a včlenili tyto symboly do textových editorů, takže s nimi může pracovat i neprogramátor, který se snaží vyjádřit graficky určitý problém či závislost vztahů a návazností.

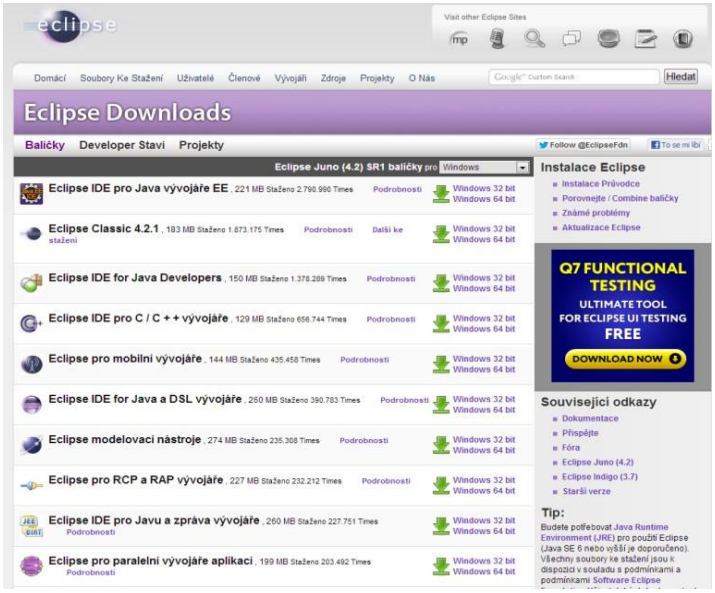
Podle směrů, v kterých potřebujeme dotyčnou aplikaci naprogramovat, vzniklo množství programovacích jazyků, které byly podřízeny odvětví, v kterém byly nasazeny, či uživatelům, kteří se s programováním setkávají. Určitě jiné nároky na programování bude mít doktor medicíny při tvorbě programu (pracovního postupu, který bude třeba testovat krev), než letecký konstruktér. Taktéž ve strojní výrobně se budou při programování obráběcích strojů klást jiné nároky na programátora, který bude připravovat program pro sériovou linku osazenou CNC stroji, a na frézaře, který bude obsluhovat programovatelnou frézku. Aniž si to uvědomujeme, tak programování využíváme i v běžném životě při obsluze domácích spotřebičů (pračky, DVD rekordéry, chytré dálkové ovladače, vytápění a podobně).

1.2. Kdo je to programátor a jak programuje

Programátor je osoba, která vytváří posloupnost příkazů a instrukcí, které mají svoji specifickou logiku. Samozřejmě jsou různé úrovně programování a různé programovací jazyky. Jakýkoliv programátor musí mít znalosti konkrétního programovacího jazyka a ovládat vývojové a aplikační prostředí pro tento jazyk. Některé programovací jazyky jsou si velmi podobné (protože vycházely ze stejných základů), ale mají své specifické odlišnosti. Taktéž vývojové prostředí se může pro dotyčný programovací jazyk hodně lišit. Některá vývojová prostředí podporují práci s několika programovacími jazyky. Jedním z mnoha programovacích prostředí, které je zdarma, je vývojová platforma Eclipse. Jde o open source aplikaci (otevřený software). Z názvu je patrno, že tento software má otevřený zdrojový kód (volně stažitelný pro kohokoliv, a zároveň taktéž legálně dostupný, který můžeme za předem daných známých podmínek využívat, měnit či upravovat).

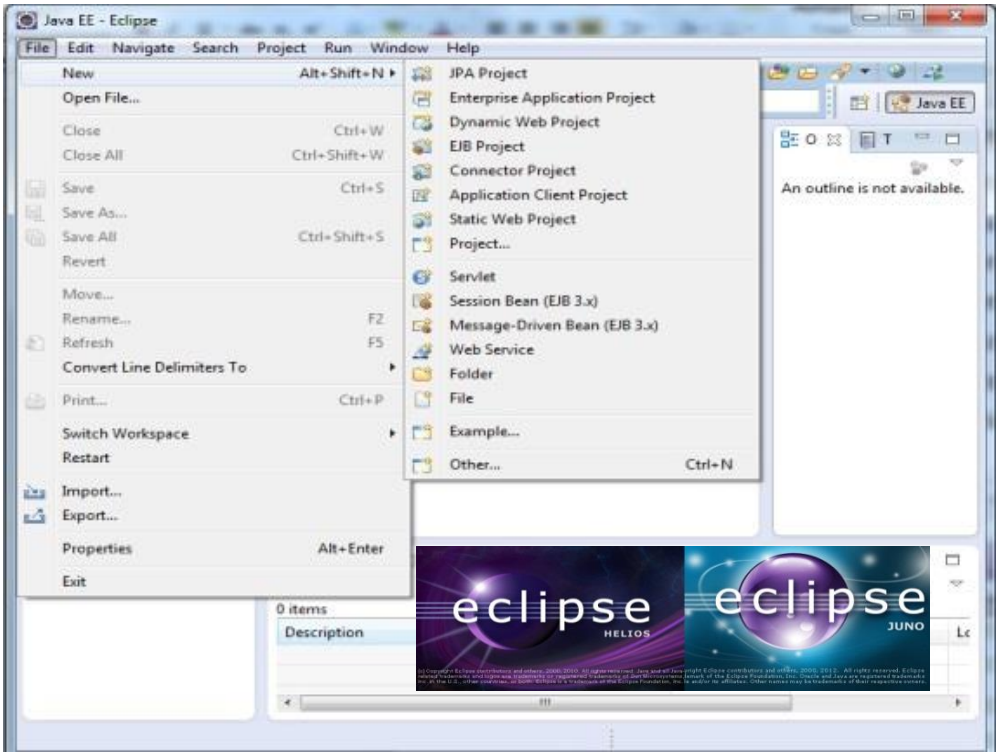
Eclipse je vývojové prostředí (IDE) primárně určené pro programování v Javě. Vsuvky (pluginy) do tohoto prostředí nám umožňují pracovat nejen na vývoji Javových aplikací, ale třeba i v PHP, C++ a podobně. Pomocí rozšiřitelných pluginů můžeme do velké části programovacích prostředí vložit (doinstalovat) nástroje pro vizuální návrh grafických uživatelských rozhraní.

Každé vývojové prostředí má svá specifika. Každý programátor má své individuální postupy a oblíbená řešení. Bohužel i jednoduše řešitelná věc se dá někdy naprogramovat značně složitě a neprůhledně.



Obrázek 6 Platforma Eclipse, Java Development Tools [1]

Každý z nás je osobnost formovaná prostředím a dřívějšími nabytými znalostmi a zkušenostmi, které potom, aniž si to uvědomujeme, předáváme do hotového programátorského díla. Říkáme, že programátor má svůj rukopis. Jsou programy jednoduše čitelné (zde máme na mysli zdrojové kódy programu, jež ještě nebyly přeloženy).



Obrázek 7 Platforma Eclipse [1]

Najdeme však i programy, v kterých se i vlastní tvůrce po určité době není schopen orientovat. Nejde jen o to, že každý má jiný způsob myšlení a navrhuje i jiné algoritmy při řešení konkrétního úkolu (každý úkol lze řešit více způsoby). Program by měl být napsán tak, aby byl jednoduše čitelný (třeba i srovnané začátky). Tím se sice program zvětšuje, ale dá se v něm lehce orientovat.

K problému se dá taktéž přistupovat z pohledu efektivnosti běžícího programu (využívání systémových prostředků).

```
  * @return preda kompletni tabulku      */
            public static String[][]
NaplnData(String[] jmena,
String[] hlavicka,
int[][] data)        { String[][] pole = new
String[9][9];
pole[0] = hlavicka;    for(int i=0; i< 8; i++)
                { pole[i+1][0] = jmena[i];      }
for(int i=0; i< 8; i++  {   for(int j=1 ; j<9 ; j++)
{  pole[i+1][j] = Integer.toString(data[j-1][i]);}   }
                return pole;    }
              /**
```

Text 1 Nepřehledné členění programu (nezarovnané) [1]

Jsme schopni napsat program, jenž může vytížit procesor na 10 % výkonu a jiný programátor, stejný úkol vyřeší tak, že vytíží procesorové prostředky na 95 % výkonu.

Každý hotový program (můžeme mu říkat autorské dílo) je vhodné ve zdrojovém kódu dostatečně okomentovat (popsat). V době psaní programu nám to připadá velmi jasné. Po delší době si již nemusíme vzpomenout, jak jsme to vlastně mysleli. Pokud potom chceme v programu něco předělávat, doplňovat, máme to s popisem mnohem snazší.

```
  * vytiskne celou tabulku
public static void Tiskni(String[][] pole)
 {
    for(int i=0 ; i<9 ; i++)
    {
       for(int j=0 ; j<9 ; j++)
       {
          System.out.print(pole[i][j]+"\t");
       }
       System.out.print("\n");
    }
 }
```

Text 2 Přehledné členění programu [1]

```
// pri stisku tlacitka na  klavecnici
        @Override
        public void keyPressed(KeyEvent arg0) {  switch(arg0.getKeyCode())   {
       case KeyEvent.VK_LEFT : strojek.pohniDoleva(10); break;
                              //pri stiskku leve sipky posunem doleva
       case KeyEvent.VK_RIGHT : strojek.pohniDoprava(10); break;
                              // prave sipky doprava
       case KeyEvent.VK_K : mlynek.stop();                  //tlacitka k (zastavime mlynek) }
        repaint();                // po stisknuti tlacitka prekreslime applet  }
```

Text 3 Program s komentářem. (Okomentování programu je provedeno za //) [1]

Správnost běhu programu je podmíněna taktéž jeho bezchybnou funkcí. Programátor tedy ve svém programu nesmí zapomenout ošetřit všechny případné kolizní stavy a musí mít snahu ošetřit chyby, které může způsobit uživatel. Nejhorší pro uživatele je, když mu počítač nahlásí, že se vyskytla chyba a program se ukončí (skončí bez uložení vložených nebo již upravených veličin).

Představme si příklad, že obsluha vkládá jména osob, bydliště a datum narození. Program nebude ošetřen proti tomu, že na místo data (číselné hodnoty) se vloží text. Program jen nahlásí kód nějaké chyby (třeba Eror 256) a nestandardně skončí. Obsluha nejen že vůbec nechápe, co je ten kód 256, ale navíc její práce (třeba ta jména a data tam vkládala celé dopoledne) je pryč. Dobře napsaný program musí počítat se všemi možnými eventualitami a musí být proti těmto případným chybám ošetřen. Nejde jen o ošetření vlastního programu, ale i o komunikaci směrem k uživateli. Aby uživateli sdělil v co nejsrozumitelnější a nejjednodušší podobě, kde dělá chybu a jak má vzniklou chybu napravit.

Jedna z možností, jež může obsluhu přivést k nepříčetnosti, nastane třeba při vkládání dat do databáze. Program chce vložit datum. Datum obsluha musí vložit třeba ve formátu „30.05.2013“. Není však uvedeno, že se má vložit den, měsíc, rok a to vše oddělovat tečkami. Program sice, pokud to nebude vloženo v tomto formátu, nahlásí, že data byla špatně vložena, ale neposkytne nic jiného a chce nové vložení údajů. Uživatele po celou dobu jeho snažení nemusí napadnout, že by se údaje mohly oddělovat tečkami a ani neví, zda má dávat třeba data v pořadí rok, měsíc, den (taktéž neví, zda se mají dny a měsíce zadávat dvojciferně, či zda rok nemá být vyjádřen jen posledním dvojčíslím). Pracovat s takovými programu je pak pro uživatele utrpení.

Programátor by tedy měl vycházet z logiky předpokládané obsluhy a vždy vkládat nápovědy nebo vytvářet kontextovou možnost (nejčastěji tlačítkem F2) tuto nápovědu vyvolat. Programátorovi by se měly na začátku přesně formulovat požadavky, co daný program má dělat a hlavně, jak se k tomuto programu bude přistupovat ze strany uživatele a jaké výstupy obsluha bude očekávat.

```
public class Pozdrav{
public static void main (String[] args){
      System.out.print("Ahoj jak se mate");  // Metoda Tisk na obrazovku

       System.out.println("01 pokus");   // Tisk na obrazovku vzdy na nový řádek
       System.out.println("02  Bed"); /* pokus o komentář*/
       System.out.println("03  Bedy");
}
}
```

Text 4 Ukázka programu pro tisk na obrazovku [1]

Programové dílo (program) má usnadňovat obsluze činnost, má vycházet z logiky práce, v níž bude nasazen. Nezřídka se však stává, že firma pro své zaměstnance zakoupí program, který sice má 95 % funkcí, ale ty se ve firmě nevyužijí a funkce, kterých by bylo zapotřebí, zde aplikovány nejsou.

POJMY K ZAPAMATOVÁNÍ

Programátor.

Program

Vývojové prostředí

Komentování programu

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

https://netbeans.org/downloads/start.html?platform=windows&lang=en&option=all
http://www.eclipse.org/downloads/

1.3. Vliv hardware počítače na programování

Každý program má jiné nároky na operační paměť, odkládací diskový prostor, rychlost samostatného procesoru, rychlosti sběrnic, zpracovávání grafického zobrazování dotyčné aplikace. Nemalý vliv na programování a rychlost hotového programu má samozřejmě i operační systém. Musíme si taktéž uvědomit, že nejrychleji pracující programy jsou přímo programy ve strojovém kódu. Programy mohou využívat prostředí operačního systému, ale také mohou pracovat přímo s instrukční sadou daného procesoru. Příkladem může být program, který je umístěn přímo v paměti Pic (mikrořadič PIC).

```
:020000040000FA
:100000002500780A2A050A040D0C2400050C2B0093
:100010004A05670066070A0A68006607160AA8020A
:100020000000680C880003070D0A010868000A0632
:10003000210A6606290AAE0C880003060208A802F7
:10004000190A6607290AAE0C880003060208A802EE
```

Text 5 Ukázka programu v mikrokontroléru PIC12C508 [1]

1.3.1. Výrazy používané ve výpočetní technice

- **Bit** – nejnižší jednotka nesoucí informaci, může nabývat hodnoty buď 1, nebo 0.
- **Slabika** – byte, půlslovo, to je označení pro 8 bitů (bity číslujeme 7–0 popořadě), může nést hodnotu čísla se znaménkem (–128–127, shortint) nebo bez znaménka (0–255, byte); za počet slabik píšeme B (KB, MB).
- **Půlslabika** – půlbyte, nibl, označení pro 4 bity.
- **Slovo** – dvě slabiky, označení pro 16 bitů. Bity číslujeme 15–0 (7–0, 7–0) popořadě. Může nést hodnotu čísla se znaménkem (–32768–32767, integer) nebo bez znaménka (např. adresa, 0–65535).
- **Instrukce** – pokyn mikroprocesoru k vykonání nějaké činnosti (přesuň, sečti).
- **Program** – posloupnost instrukcí, které vedou k vykonání úlohy. Program je většinou uložen na disku ve formě souboru (typu EXE, COM). V něm je uložena řada čísel, které znamenají jednotlivé instrukce (strojový kód). Po spuštění je buď celý program, nebo jeho část, uložena do paměti počítače.
- **Překladač** – program umožňující převést algoritmus zapsaný v textovém tvaru do strojového kódu mikroprocesoru. Ve strojovém kódu jsou jednotlivé instrukce zapsány s pomocí jedné, či více slabik, které jsou pro každou instrukci odlišné. Jestliže tedy necháme počítač, aby četl instrukce z části paměti, kde jsou data (ne operační kód instrukcí), dojde většinou k "zmrznutí" počítače, protože data mohou obsahovat kódy znamenající instrukce nesmyslného programu.

- **Paměť** – část počítače, kde je uložen program a data.
- **Zásobník** – část paměti sloužící k odkládání dat, případně k předávání hodnot mezi podprogramy.
- **Mikroprocesor** – v každém počítači nalezneme jeden či více mikroprocesorů. Jedná se o část zajišťující přesuny dat v počítači a jejich zpracování. Uvnitř mikroprocesoru jsou vždy tyto části:

  Aritmeticko-logická jednotka (ALU) – je to sčítačka doplněná o posuvné registry a logické obvody. Vykonává operace spojené se zpracováním dat: matematické, logické a posuvy (rotace). Počet bitů, se kterými je schopna ALU pracovat, udává, kolikabitový je celý mikroprocesor.

  Registry – jsou rychlé paměti určené pro zaznamenávání dat a adres. Jednotlivé mikroprocesory se od sebe liší počtem registrů a jejich velikostí, která udává, jak velké číslo jsme schopni v něm uchovat. Jestliže registr slouží jako vstupní a výstupní pro hodnoty určené ALU, říkáme mu střadač.

  Dekodér instrukcí – dekóduje číslo, které pro mikroprocesor znamená instrukci.

  Obvody řízení – zajistí vykonání instrukce vytvořením posloupnosti impulsů, která ovlivní jednotlivé části procesoru tak, aby po ukončení této posloupnosti byla instrukce vykonána. Tato posloupnost je ovlivněna mikroprogramem popisujícím jednotlivé instrukce.
- **Vstupní-výstupní porty** – za ně považujeme obvody, které jsou určené k předání dat do nebo z počítače.
- **Adresa** – číslo označující místo slabiky v paměti nebo vstupního/výstupního portu, se kterým chceme pracovat (tzn. kam požadujeme zapsat, odkud chceme číst). Maximální velikost adresy určuje velikost adresového prostoru, tedy počet slabik v paměti nebo počet vstupně/výstupních portů.
- **Systémová sběrnice** – je soustava vodičů určená k transportu dat, řídících signálů a adres mezi mikroprocesorem, pamětí a vstupně výstupními obvody. Má tyto části:

  Datová sběrnice – určená k přesunům dat a kódů instrukcí.

  Adresová sběrnice – určená k přesunům adres slabik v paměti a adres vstupně-výstupních portů.

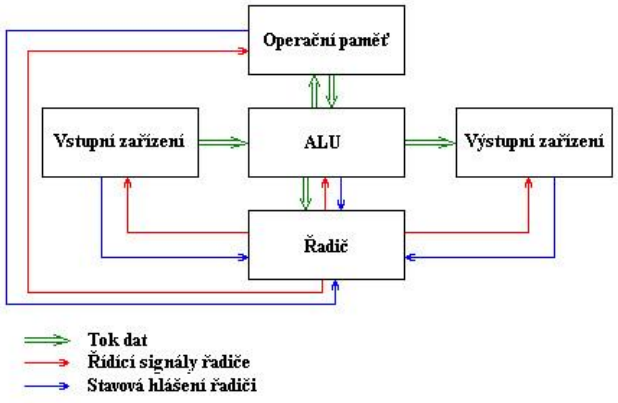
  Řídící sběrnice – určená k synchronizaci všech částí počítače. [30]

1.3.1. Architektura počítače

Představ o tom, jak by počítače měly vypadat a jak by měly fungovat, existovalo a existuje více.

Von Neumannova architektura (von Neumann architecture), někdy označovaná také jako von Neumannova koncepce či von Neumannovo schéma, je ucelenou soustavou názorů a představ o tom, jak by měl počítač fungovat, z jakých hlavních částí by se měl skládat, co a jak by tyto části měly dělat, jak by měly vzájemně spolupracovat atd. Význačným rysem von Neumannovy architektury je samotný způsob provádění programu. Dnes nám asi připadá zcela samozřejmé, že strojové instrukce, ze kterých se každý přímo spustitelný program skládá, se provádějí postupně, jedna za druhou. Řečeno jinými slovy: každá instrukce se provede tehdy, až na ni dojde řada, a nad takovými daty, jaká jsou právě k dispozici.

Von Neumannova architektura je tedy ryze sekvenční a ve své čisté podobě nepředpokládá žádný paralelismus. To je na jedné straně velká výhoda, neboť sekvenční postup je přirozenější a intuitivnější než postup paralelní. Psát programy, které předpokládají čistě sekvenční zpracování, je pak snazší než psaní paralelních programů. [22]



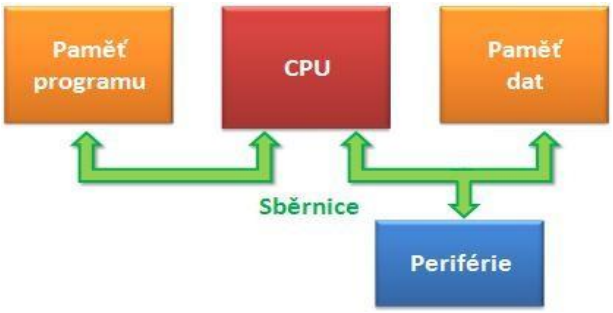
Obrázek 8 Von Neumannova architektura PC [23]

Podle schématu von Neumannovy architektury se počítač skládá z pěti hlavních modulů:

1. Operační paměť (memory): slouží k uchování zpracovávaného programu, zpracovávaných dat a výsledků výpočtu.
2. ALU – Arithmetic-Logic Unit (aritmetickologická jednotka): jednotka provádějící veškeré aritmetické výpočty a logické operace. Obsahuje sčítačky, násobičky (pro aritmetické výpočty) a komparátory (pro porovnávání).
3. Řadič (control unit): řídící jednotka, která řídí činnost všech částí počítače. Toto řízení je prováděno pomocí řídících signálů, které jsou zasílány jednotlivým modulům. Reakce na řídící signály, stavy jednotlivých modulů jsou naopak zasílány zpět řadiči pomocí stavových hlášení.
4. Vstupní zařízení (input): zařízení určená pro vstup programu a dat.
5. Výstupní zařízení (output): zařízení určená pro výstup výsledků, které program zpracoval. [23]

Harvardské schéma

Harvardská architektura má na rozdíl od von Neumanovy architektury oddělený paměťový prostor pro data a pro program. Harvardská koncepce dovoluje používat pro paměť programu například paměti typu ROM (Read Only Memory) a umožňuje v podstatě zdvojnásobení velikosti paměti oproti von Neumanově architektuře při stejně veliké adresové sběrnici. Bit nutný pro realizaci této možnosti je obsažen v instrukčním souboru, který obsahuje instrukce pro komunikaci s datovou pamětí. U von Neumanovy koncepce je paměť pro data a pro program společná. Tímto způsobem se můžeme na instrukce dívat jako na data a během programu je můžeme měnit a ovlivňovat chování programu. [23]

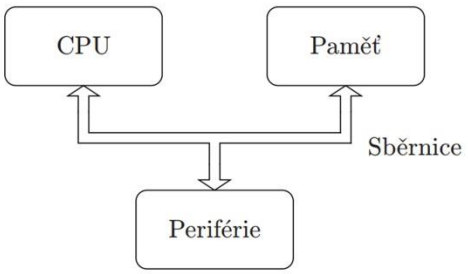


Obrázek 9 Harvardská architektura [23]

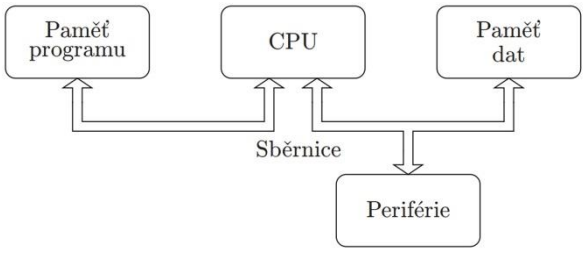
Výhody:

- program nemůže přepsat sám sebe,
- paměti mohou být vyrobeny odlišnými technologiemi,
- každá paměť může mít jinou velikost nejmenší adresovací jednotky,
- dvě sběrnice umožňují jednoduchý paralelizmus, kdy lze přistupovat pro instrukce i data současně.

Porovnání von Neumanovy architektury a Harvardské architektury počítače.



Obrázek 10 Von Neumannova architektura



Obrázek 11 Harvardská architektura počítače

1.3.2. Mikrokontrolér

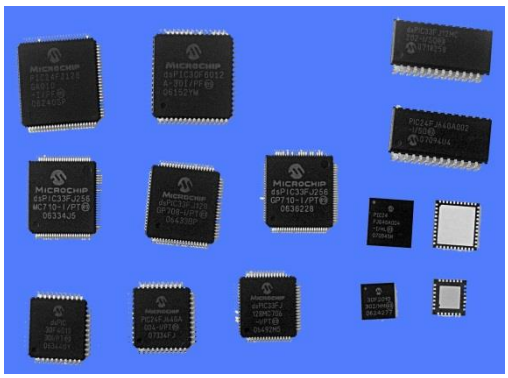
Pro Mikrokontrolér (angl. Microcontroller) se používají zkratky: µC, uC nebo MCU. Mikrokontrolér je programovatelná elektronická součástka, která má nejčastěji podobu integrovaného obvodu.

Obrázek 12 Mikrokontrolér PIC [1]

Mikrokontrolér, někdy rovněž označovaný jako mikropočítač nebo jednočipový mikropočítač, je miniaturní počítač, který je integrován na jediném čipu a který typicky obsahuje procesor (rovněž označovaný jako CPU), paměť, programovatelné vstupně-výstupní rozhraní a další periferní obvody. Mikrokontrolér je vhodný pro použití v řízení a je navržen a určen pro tzv. vestavné (angl. embedded) aplikace, tj. mikrokontrolér je buď řídící jednotkou („mozkem“) nějakého přístroje, nebo je součástí nějakého dalšího zařízení, kde plní určitou specifickou funkci (na rozdíl od běžných počítačů, které jsou určeny k univerzálnímu použití). Mikrokontrolér je proto navržen jako samostatná jednotka schopná komunikace a interakce s okolím. V mikrokontroléru jsou obvykle kromě vstupně-výstupních obvodů integrovány i mnohé další periferní obvody, např. čítač, časovač, komparátor, sériové porty, analogově-digitální, příp. digitálně-analogový převodník, USB, PWM (pulsně-šířkový modulátor), paměť EEPROM a další. Protože se mikrokontroléry často používají v přístrojích napájených z baterií, je u nich rovněž kladen velký důraz na malou spotřebu. Mikrokontrolér tak obvykle disponuje různými úspornými režimy, umožňuje řídit kmitočet oscilátoru nebo vypínat jednotlivé moduly. Některé typy mikrokontrolérů mohou být dále speciálně navrženy, aby splňovaly určité specifické požadavky. Příkladem může být zvětšený rozsah pracovních teplot (například –40 ºC až 150 ºC) u mikrokontrolérů určených k řízení motorů automobilů apod. [19]

1.3.3. Mikrokontrolér PIC

Mikrokontroléry PIC představují rodinu mikrokontrolérů založených na Harvardské architektuře (počítačová architektura s fyzicky oddělenou pamětí programu, dat a oddělenými sběrnicemi pro přístup k instrukcím a datům, díky čemuž nemusí mít programová paměť spolu s datovou paměťí stejně dlouhé datové slovo). Mikrokontroléry PIC vyrábí firma Microchip Technology a jsou populární mezi amatéry i profesionály zejména díky široké nabídce typů, přijatelné ceně a množství dostupné literatury.



Obrázek 13 Mikrokontroléry PIC [20]

Tyto mikrokontroléry se snadno programují a současně lehce přeprogramovávají. Paměti FLASH, které jsou v mikrokontrolérech, mají sériové programovací rozhraní. Softwarové vývojové prostředí pro jejich programování a ladění je ve velké míře zdarma dostupné.[19]

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

http://belza.cz/control/dopic.htm
http://www.flajzar.cz/odborna-literatura-a-cd/mikrokontrolery-pic16f630-a-pic16f676.htm

1.3.4. Mikrokontroléry u stavebnic Robotis

U stavebnic Robotis BIOLOID jsou použity Mikrokontroléry s procesorem ATmega . Výrobce dodává tři typy a označuje je CM.

Modul CM-5 (hlavní řídící prvek) obsahuje desku s mikrokontrolérem ATmega128 a konektory pro komunikaci s protokolem TTL. Je určen pro ovládání AX (Bioloid sady) a MX-XXT (DarwinOp humanoidní robot).

Modul CM-510 obsahuje desku s ATmega 2561 mikrokontroler.

Obrázek 14 Mikrokontroléry ze stavebnic Robotis CM-530, CM-510, CM-5 [1]

Modul CM-530 obsahuje ARM Cortex STM32F103RE mikrokontrolér. Řídící mikrokontroléry mají v sobě flash paměť, která po naprogramování řídí pohybové servomotory sestaveného robota. Po sériové sběrnici do něj proudí data ze snímačů.

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

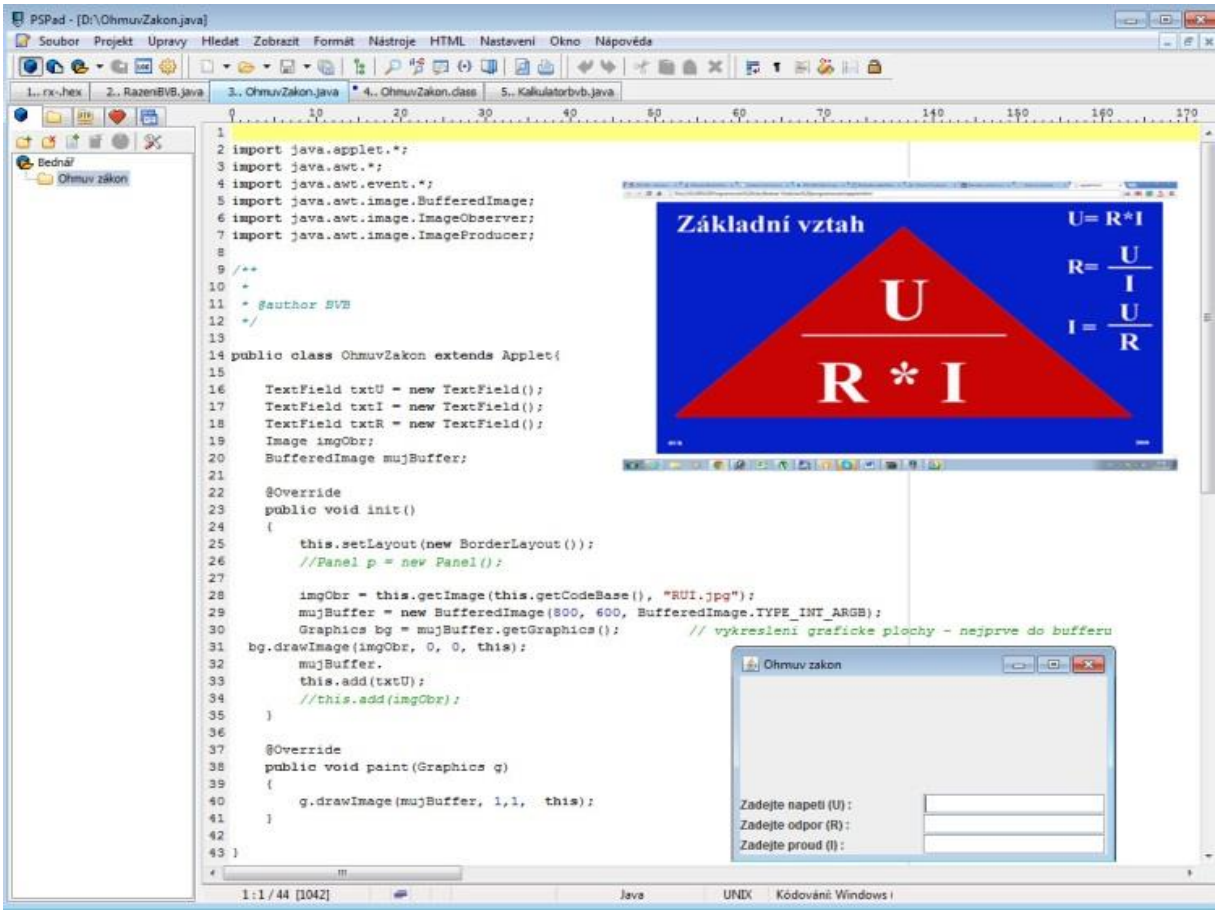
http://www.robotis.com/xe/download_en
http://www.generationrobots.com/cm-5-main-controller-robotis,us,4,CM-5-Main-Controller.cfm
http://www.generationrobots.com/programmable-robots-with-ros,us,2,241.cfm
http://www.conrad.cz/roboti.c37371

ČAS POTŘEBNÝ KE STUDIU 180 minut

CÍL: Představit programovací nástroje

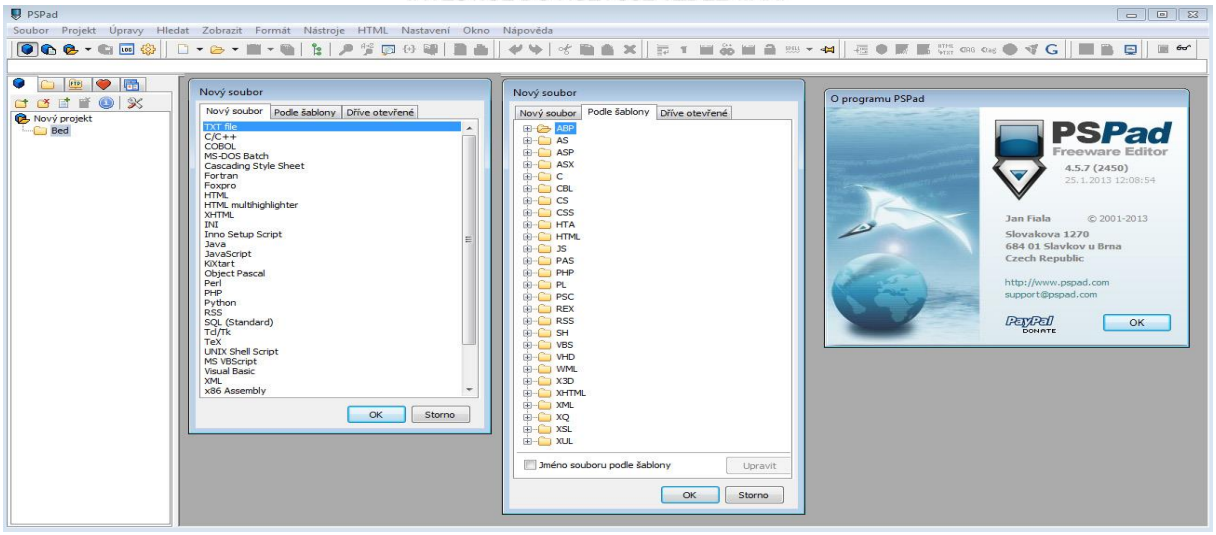
2. PROGRAMOVÁNÍ NA PC

Při programování na PC musíme nejdříve řešit, pod jakým operačním systémem budeme programovat a na jakém systému dotyčná aplikace bude provozována. Můžeme vytvářet programy, které budeme jen pomocí překladače kompilovat do spustitelného tvaru pro rozličné platformy. Příkladem může být programování v Javě, kdy dotyčnou naprogramovanou aplikaci můžeme přenést třeba z klasického počítače běžícího pod operačním systémem Windows na mobilní telefon pracující s operačním Systémem Android.



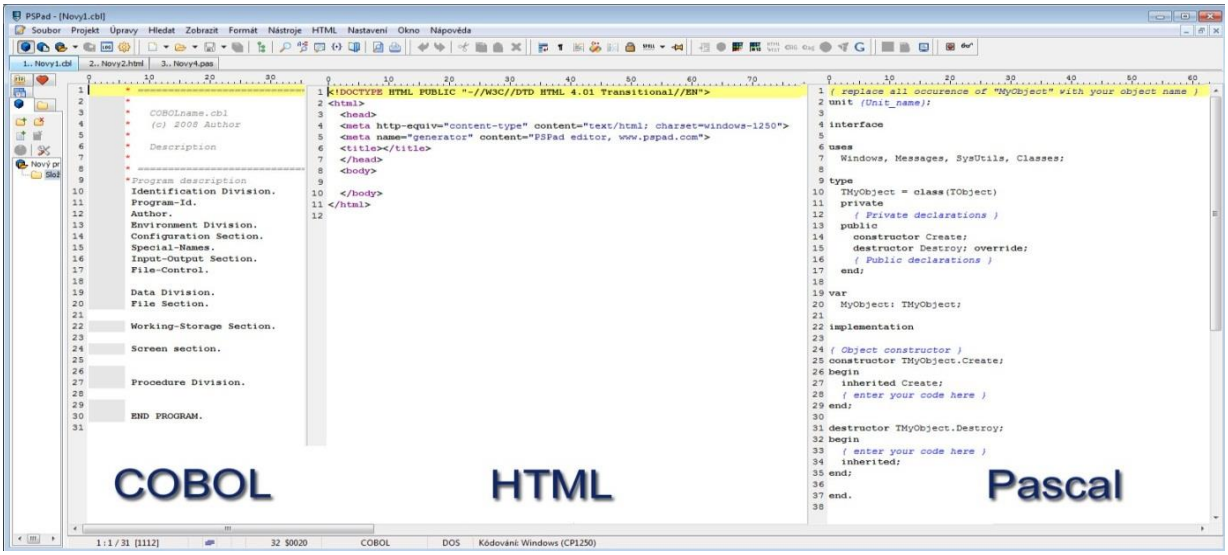
Obrázek 15 Ukázka java appletu (programu spustitelného přes webový prohlížeč)

Pokud známe dokonale syntaxi (skladbu a zákonitosti) vybraného programovacího jazyku, můžeme programovat přímo v Poznámkovém bloku. Toto je ale vhodné pouze pro malý kousek zdrojového kódu. Je jistě lepší použít třeba univerzální textový editor PSPad (který je volně šiřitelný) pro platformu operačního systému MS Windows.



Obrázek 16 Ukázka šablon a předefinovaných typu souborů v programu PSPad

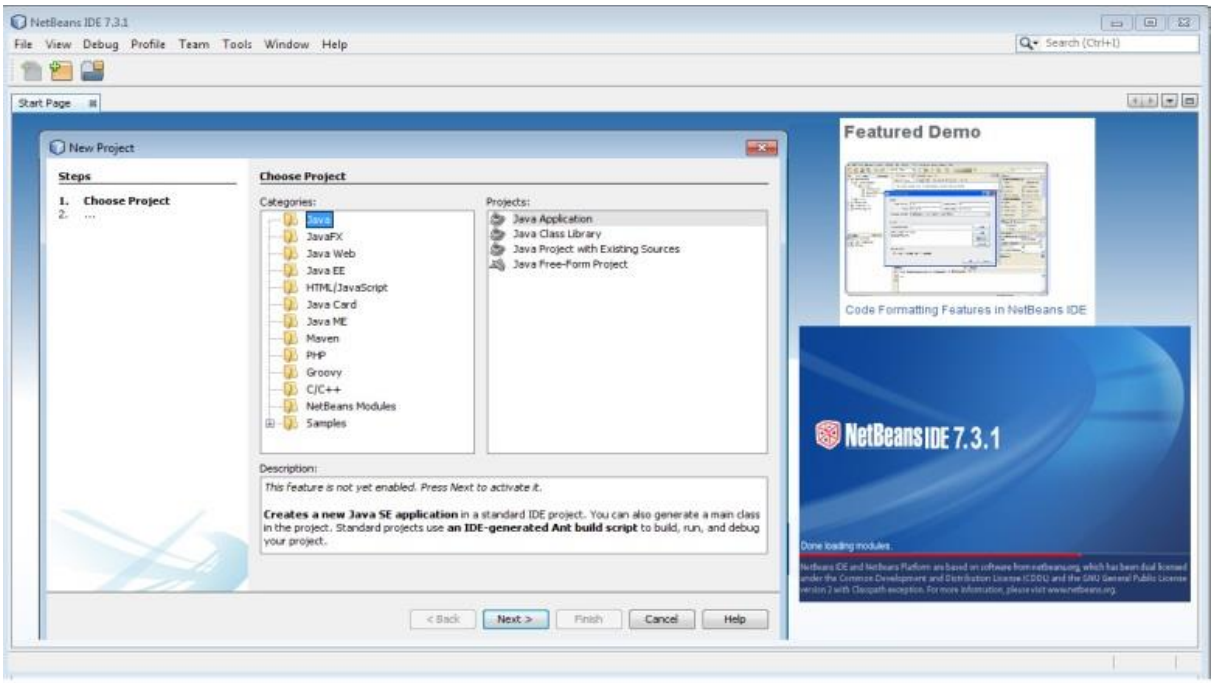
Editory na rozdíl od Poznámkového bloku nám mohou ukazovat číslování řádků, upozorňovat na programátorské chyby změnou barev řádku a taktéž pro přehlednost zobrazovat komentáře v programu odlišnou barvou než samostatný zdrojový kód. Pokud jako v případě PSPadu mají již v sobě šablony, tak nám hned automaticky vytvoří úvodní zápis.



Obrázek 17 Předdefinované hlavičky v programu PSPad

Pro psaní programu je vždy výhodnější pracovat v nějakém specializovanějším vývojovým prostředí. Příkladem může být NetBeans. Jde o IDE (vývojové prostředí) nástroj, pomocí kterého programátoři mohou psát, překládat, ladit a distribuovat aplikace. Toto vývojové prostředí je v jazyce Java, ale podporuje skoro jakýkoliv programovací jazyk. NetBeans je bezplatně šířený produkt (jde o Open Source projekt) a jeho užívání není nijak omezeno.

Mezi další vlastnosti NetBeansu patří např. UML modelování, XML editor, nástroje pro dynamickou tvorbu webu (AJAX, CSS, JSF) a mnoho dalších profesionálních a moderních technologií, které oceníte jak u vývoje desktopových aplikací, tak i webových stránek. Existuje rovněž velké množství modulů, které toto vývojové prostředí rozšiřují.



Obrázek 18 Vývojové prostředí NetBeans [1]

V dnešní době získávají na popularitě služby vývojových prostředí, IDE v cloudu. Do cloudu se pomalu stěhuje také vývoj softwaru, především mobilních aplikací. Služby jako například Cloud9 IDE, Codenvy, Exadel Tiggzi lákají vývojáře do cloudu a slibují jednoduché použití, možnosti spolupráce, neomezený přístup odkudkoliv, a to i z několika počítačů. Programátor musí mít jen připojení k internetu a vytváří aplikace na vzdáleném serveru. Nemusí mít nainstalované ve svém počítači žádné vývojové prostředí.

POJMY K ZAPAMATOVÁNÍ

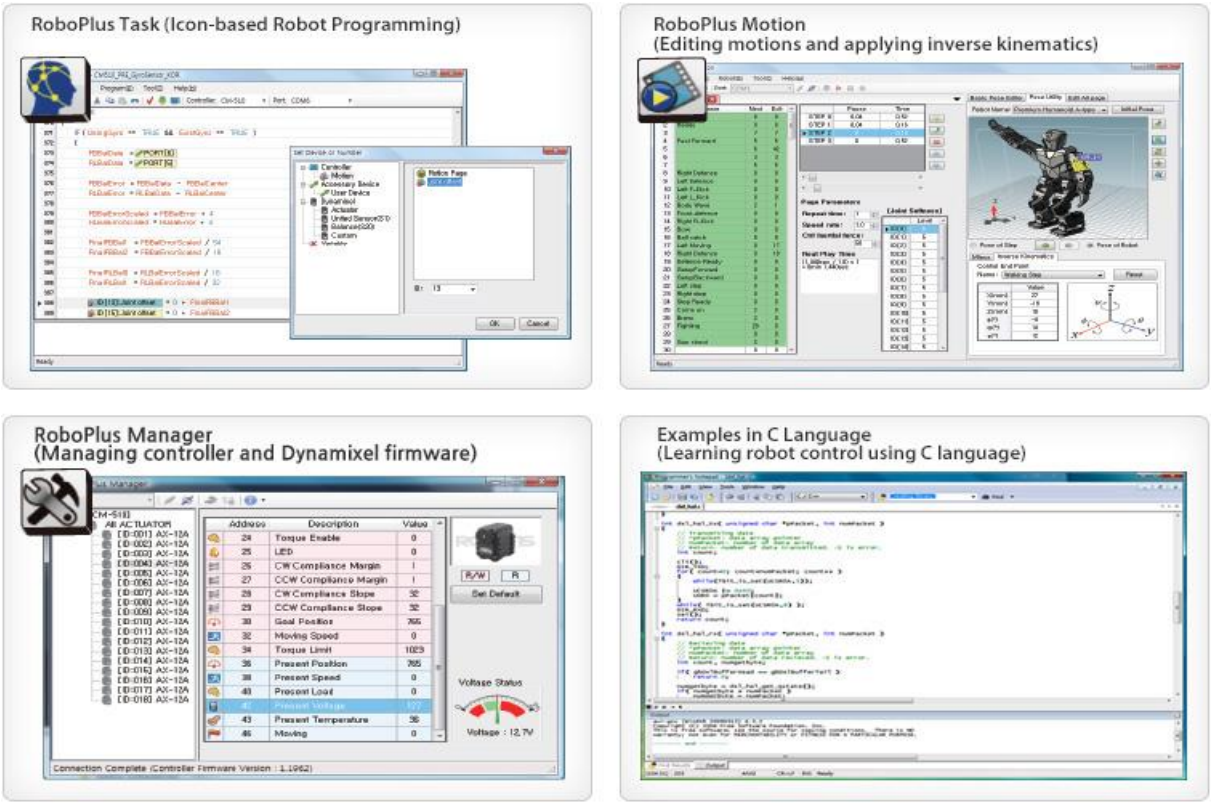
Vývojové prostředí (IDE)

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

http://www.pspad.com/cz/
https://netbeans.org/index_cs.html
http://vyuka.pecinovsky.cz/vse/115/IDE_NetBeans_S.htm
http://www.panrepa.org/CASE/jaro2007/ide_case_jaro2007.pdf

2.1. Ukázka programování robotů Robotis BIOLOID Premium kit

Pro práci s výukovou stavebnicí a programování RoboPlus dodává výrobce software BIOLOID Premium kit. Tento programovací prostředí spustíme na operačním systému Windows a má několik možností.



Obrázek 19 Ukázky programovacího prostředí pro rotota BIOLOID RoboPlus [21]

- **RoboPlus Task**

Jde o softwarový nástroj, který umožňuje přímou komunikaci s robotem v textovém režimu. Uživatelé zde mohou vytvářet své vlastní příkazy a definovat instrukce.

- **RoboPlus Motion**

Pomocí tohoto nástroje programujeme roboty nepřímo. Vykonáváme mechanické pohyby robotem, které zaznamenáváme programem. Program si ukládá výchozí pozice, které potom může vykonávat. Jde o editor pohybu, který můžeme nazývat inverzní kinematikou. Tento software je schopen komunikovat s programem Microsoft Excel (pro upřesnění polohy pohonů na základě svých výpočtů). Při tomto programování nemusíme znát žádný programovací jazyk. Musíme se jen naučit jak se dotyčný pohyb servomotorů a jejich pozice (jak výchozí tak konečná) zapíše do programu. Příkazy pohybů pak můžeme v programu ovlivňovat pomocí doplňkových zápisů (podprogramů) a spouštět jednotlivé kroky pohybů, které se mají vykonávat podle časových údajů a podnětů ze snímačů.

- **RoboPlus Manager**

Je určen pro správu všech funkcí robota. Tento software sleduje a nastavuje správu firmware. RoboPlus Manager poskytuje jednotný pohled na všechny součásti (komponenty a snímače) připojené na komunikační sběrnici. Můžeme zde upravovat parametry každého prvku v reálném čase.

- **Examples in C Language**

Umožňuje vývoj autonomního chování robotů. Příkazy se zadávají velmi podobně jako v programovacím jazyku C. Je určen jak pro programátory v jazyku C, tak pro uživatele, kteří se teprve setkávají, poprvé z objektovým programováním.

2.2. Vliv typu úlohy na programování

Pokud budeme uvažovat o programech, které běží (fungují) pod určitým operačním systémem, tak podle typu prováděné úlohy budeme vybírat programovací jazyk vhodný pro zadaný typ úlohy. Některé programovací jazyky jsou vhodné pro úkoly čistě výpočetní, bez grafického ztvárnění. Najdeme i specializované pro programování her, či dokonce pro vytváření virtuálních videofilmů.

Najdeme i programy či multiplatformní nástroje pro tvorbu her s dokonalou 3D grafikou, bez jakýchkoliv zkušeností s programováním. Příkladem může být třeba Maker3D.

Každý program sice využívá operační systém, ale třeba účetnický program bude mít jistě jiné nároky na grafické zpracování než nějaká automobilová hra. V následujícím budou uvedeny některé typy úloh:

2.2.1. Matematické úlohy

Matematická úloha a vědeckotechnické výpočty vyžadují provádění velkého počtu operací se všemi druhy čísel. Reprezentace iracionálních čísel vyžaduje dlouhá slova. Výpočty složitějších matematických funkcí vyžadují značný počet opakování výpočtů, přesnost vyžaduje použití čísel na mnoho desetinných míst. Algoritmy jsou náročné, je potřeba velké výpočetní kapacity. Program se nemusí využívat pravidelně.[14]

2.2.2. Ekonomické úlohy

Ekonomická úloha vyžaduje poměrně jednoduché operace (porovnávání, sčítání a podobně), ale s velkým počtem položek. Algoritmy jsou jednoduché, data nevyžadují zpravidla příliš dlouhá slova. Důležitá je možnost práce se složitou strukturou dat, a vyhledávání položek. Program se provádí pravidelně. Příkladem jsou práce s databázemi. Jinak dávkové programy. [14]

2.2.3. Řízení procesů (i výrobních a robotických systémů)

Řízení procesů vyžaduje na rozdíl od obou předchozích trvalou kontrolu nad probíhajícím časem, možnost reakce na vnější okolnosti a spojení s řízeným procesem. Pro spojení obsluhy s procesem slouží takzvané vizualizační systémy. Jinak událostmi řízené programy. [14]

2.2.4. Multimediální aplikace

Multimediální aplikace vyžaduje rychlé sejmutí dat ze vstupního zařízení (kamery, fotoaparáty) a po zpracování (různé formáty) předat do výstupního zařízení (displeje, projektory). Zde se uplatní jak požadavek na složitost, tak na rychlost algoritmů. [14]

2.2.5. Programování síťových her

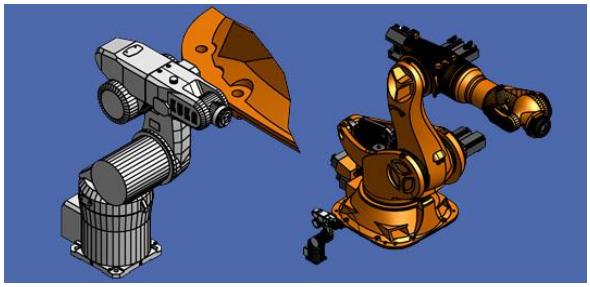
Nejdůležitějšími parametry pro programování hry bude vykreslování hry na obrazovku (FPS), rychlost, s jakou lze posílat data mezi klientem a serverem a rychlost výpočtu stavu hry.

2.3. Jaké PC pro programování

V době psaní tohoto textu je běžná sestava tvořena procesorem Intel Core i5 pracujícím na taktu 3 GHz. Má 8 GB RAM a pevný disk 1 TB. Jako operační systém se používá Windows 8 nebo Linux. Vývojové prostředí jsou schopny pracovat i na počítačích mnohem méně výkonných. Můžeme pracovat i na počítačích s procesorem Pentium a operačním systémem Windows XP. Na těchto počítačích ale již nemá cenu pracovat s 3D prostorovým zobrazováním a jejich animacemi.

Můžeme říci, že jakákoliv dnešní sestava či dokonce i tablet se mohou používat pro psaní programů a spolupracovat s robotky, pokud mají k tomu odpovídající rozhraní.

Záleží hlavně na operačním systému, a jaké vývojové prostředí si vybereme. Vývojová textová rozhraní mají minimální nároky na výkon počítače. Pokud budeme potřebovat 3D zobrazování, budeme zajisté potřebovat grafickou kartu s větším výkonem, než je třeba pro textové aplikace.



Obrázek 20 Průmyslová animace [26]

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

http://www.kuka-robotics.com/czech_republic/cs/products/systems/occubot/start.htm
http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Featured-Articles/What-Material-Removal-Process-is-Right-for-You/content_id/1081

2.4. Jak vzniká program

Vzniku velkého programu předchází spousta procesů. Nejdříve je třeba provést analýzu, návrh funkčnosti programu, uživatelského prostředí. Je třeba si rozmyslet, jak bude program vypadat, co bude jeho funkcí, pod jakými operačním systémem bude pracovat, budou-li v budoucnu potřeba nějaké změny v programu (a jaké), kolik lidí bude na programu pracovat a jak dlouho, jak se bude program následně udržovat aktuální a funkční atd. Z toho pak vyplyne, jakými postupy program vznikne. Dobrou pomůckou vám při tom může být například znalost jazyka UML. [27]

2.4.1. Přenositelnost

Přenositelnost programu je pojem, který nám říká, kde všude náš program poběží. Například program pro Windows 3.11 poběží i pod Windows 95, ale nepoběží pod Linuxem. Program pro 32-bitová Windows poběží při troše štěstí i pod 64-bitovými Windows, ale program pro 64-bitová Windows pod 32-bitovými Windows nepoběží.

Přenositelnost programovacího jazyka je vlastnost, která jednoduše určuje, na kterých platformách lze zdrojový kód tohoto jazyka přeložit. Jinak řečeno, pro které platformy (operační systémy a procesory) existuje překladač. Čím více existuje překladačů pro různé procesory a OS, tím je program "přenositelnější". V tomto ohledu patří jazyk C k jednomu z nejlepších. Ještě lepší je Java, která poběží nejenom na PC, ale třeba i na mobilním telefonu. Java, mimochodem, vychází z jazyka C, jako ostatně spousta dalších moderních programovacích jazyků. (Takže pokud se naučíte programovat v C/C++, bude vám syntaxe Javy povědomá.) Překladače pro různé platformy vytvářejí různí lidé, či organizace, proto vznikají standardy jazyka, které říkají, čemu všemu musí překladač rozumět. Pokud se standardů budete držet, bude váš zdrojový kód přenositelný.

Přenositelnost zdrojového kódu je dána přenositelností programovacího jazyka a dodržováním standardů. Některé překladače totiž umí „něco navíc“, co obvykle usnadňuje psaní zdrojového kódu, ale snižuje přenositelnost. Další faktor, který má vliv na přenositelnost zdrojového kódu, je používání knihoven. Knihovny jsou soubory obsahující funkce, které může váš program využívat jejich voláním. Za roky vývoje jazyka C bylo vytvořeno mnoho knihoven a jejich používáním si ušetříte velkou spoustu času. Existují standardní knihovny, které jsou dostupné na všech podporovaných platformách. Například knihovny pro práci se soubory. Některé takové knihovny budeme probírat. Jiné knihovny jsou však dostupné jen na některých platformách. Například pokud budeme chtít naprogramovat nějakou aplikaci v systému Windows a využívat přirozené grafické prostředí Windows, tj. všech těch pěkných tlačítek, rozbalovacích menu, přepínačů, okének atp., budeme k tomu využívat knihovny Windows, které nenajdeme jinde než ve Windows. Váš program se tak stane na Windows závislý. Existují knihovny pro grafické uživatelské rozhraní (angl. zkratka GUI), které jsou přenositelné například mezi Windows a Linuxem, ale zase nejsou přenositelné nikde jinde a uživatel takového programu si na první pohled všimne, že to vypadá a chová se trochu jinak. Je to dáno tím, že standard jazyka C (ani C++) nezahrnuje GUI (na rozdíl třeba od Javy, která GUI má). Díky tomu můžeme v Javě psát snadno přenositelný program s GUI, ovšem za tu cenu, že všude bude vypadat stejně). [28]

ČAS POTŘEBNÝ KE STUDIU 40 minut

CÍL: Seznámení se zásady při používání celého autorského díla (programu), či jen jeho části jak pro osobní potřebu, tak při využívání části kódu pro své vlastní programy. Členění a rozdělením softwaru podle licenčních práv.

3. LEGÁLNÍ A NELEGÁLNÍ OPERAČNÍ SYSTÉM A SOFTWARE

S rozvojem počítačů a jejich dostupností se rozvíjí i nároky na programové vybavení. Počítačové programy vznikají jako samostatná dílka známých či neznámých autorů. Na tvorbě počítačových programů je založeno mnoho prosperujících firem. Tyto firmy zaměstnávají spousty lidí, které je nutno platit. Platí se nejenom tito lidé, ale i pracoviště, energie a v neposlední řadě i distribuční cesty a prodejci. Při programování můžeme využívat již dříve hotových programů či podprogramů od jiných programátorů, ale pouze od těch, u kterých je to dovoleno a určeno typem licence. Počítačové programy (platí to i pro části zdrojových kódů) se rozdělují podle způsobu distribuce, plateb a využívání do několika skupin.

3.1. Licence registrované

Převážná část těchto licencí je uživatelům dostupná až po zaplacení poplatku (koupě produktu). Toto zaplacení může být provedeno přímo nákupem nosiče s programem (většinou i s manuálem), nebo může mít podobu uhrazení poplatku elektronickou platbou (uživateli potom nejčastěji dochází licenční číslo a smlouva pouze v elektronické podobě). Registrované licence mají dva typy vázanosti na PC či osobu.

OEM (Original Equipment Manufacture) licence zůstává po celou dobu užívání produktu vázána na počítač, na který byla prvotně nainstalována. V případě výměny hardwaru nebo poškození či likvidace počítače, dochází k zániku licence. Software OEM musí být vždy dodán včetně manuálu, licenčního ujednání a certifikátu pravosti (COA štítek). Vlastnictví samotného disku CD se softwarem není dostatečným důkazem legálnosti licence (a to i v případě, že je tento disk pravý).

Samostatná licence (nejčastěji v domácím použití) je určena pro počítač a osobu, která provedla registraci programu. Oprávněné osoby pro práci s tímto programovým vybavením jsou všechny osoby, které daný počítač používají. Komerční licenční politika je určena pro organizace a podniky s větším počtem počítačů. Smí být použita pouze na tolika počítačích, na kolik je smlouva uzavřena (kolik se koupilo licencí). Tyto licence nejsou vázané na konkrétní uživatele ani na konkrétní počítače.

3.2. Trial verze

Tato licence se vyznačuje tím, že většinou funguje ve všech částech plnohodnotně. Je ale omezena buď časem, nebo počtem spuštění. Omezení se projeví buď částečnou nefunkčností programu (nejde tisknout a nejde nic zálohovat), nebo nemožností spustit program vůbec. Po skončení zkušební doby se objeví informace, že tento program již používáte neoprávněně, nabádá vás potom k jeho odstranění nebo zakoupení, ale někdy program funguje dál jako plná (neomezená) verze.

3.3. Freeware

Freeware je software, který je distribuován bezplatně, někdy hovoříme o typu softwarové licence. Autor si u freewaru zpravidla ponechává autorská práva, například nedovoluje program upravovat nebo omezuje použití zdarma jen pro nekomerční či osobní potřebu. Jedná se tedy o volně šiřitelný program, bez placení autorského honoráře. Na světě existuje mnoho katalogů, které seskupují tyto programy většinou společně s programy, které jsou k dispozici ke stažení na zkušební dobu. Tyto katalogy jsou dobrým zdrojem alternativního software k drahým placeným licencím. [02]

3.4. Demo

Jsou to programy, které se šíří zdarma. Jsou funkčně omezeny a slouží k tomu, aby si uživatelé udělali představu, jak program pracuje, co vše nabízí, a mohli si ho částečně vyzkoušet. Funguje zároveň jako reklama sám na sebe. Většinou nabízí uživateli možnost si zakoupit tento program v plné neomezené verzi.

3.5. Shareware

Je to kategorie softwaru (programového vybavení počítače), v níž jsou programy šířeny zdarma. Zaplacení licenčních poplatků autorovi programu je většinou vyžadováno až v případě, kdy je uživatel s programem spokojen a běžně ho používá. Často je doba, po níž je možno program bezplatně zkoušet, omezena na několik dnů. [03]

3.6. Adware

Adware (advertising-supported software) je označení pro produkty znepříjemňující práci s nějakou aplikací reklamou. Ty mohou mít různou úroveň agresivity, od běžných bannerů až po neustále vyskakující pop-up okna nebo ikony v oznamovací oblasti. Další nepříjemnou věcí je např. změna domovské stránky v Internet Exploreru, aniž by o to uživatel měl zájem. Většinou ale nejsou přímo nebezpečné jako spyware a jsou spojeny s nějakým programem, který je freeware. To se dělá proto, že díky těmto reklamám mohou vývojáři financovat dál svůj program. Nebo když se jedná o placený produkt, může se díky těmto reklamám prodávat program se slevou.

Nějaký adware je taky shareware, ale není to totéž. Rozdíl mezi adware a shareware je ten, že u adware je reklama podporovaná. Některé produkty nabízejí uživateli možnost odstranění reklam po zaplacení. [04]

3.7. GPL

GNU General Public License, GNU GPL (česky „všeobecná veřejná licence GNU“) je licence pro svobodný software. Zdrojové kódy software pod GPL mohou být svobodně upravovány a používány, šířeny však musí být opět pod GPL (jestliže se je rozhodnete dále šířit), a to obvykle bezplatně (příp. za cenu distribučních nákladů). Pro nutnost dalšího šíření pod stejnou licencí je někdy také nazývána virová licence. Binární formy software používající GPL však mohou být poskytovány za libovolně vysokou úplatu. Ke GPL softwaru musí jeho autor (upravitel) na požádání zdarma poskytnout zdrojové kódy. [05]

GNU software je dostupný několika různými metodami – nákupem zařízení s již předinstalovaným systémem (Linux, Android), koupí kopií na nosiči CD-ROM, stáhnutím přes FTP, vytvořením si vlastní kopie např. od kamaráda apod.

3.7.1. Volné dílo

V oblasti software se některá díla označují anglickým termínem public domain. Tento pojem znamená, že autor díla se rozhodl, že dovolí své dílo volně užívat, bez nároku na další ochranu díla.

V českém právním systému se nikdo nemůže vzdát svých (autorských) práv, je pouze možné nabídnout veřejnosti bezúplatnou licenci na libovolné užití díla, ale lze každopádně předpokládat, že autor, který svoje dílo takto označil, se svých práv nebude domáhat (kde není žalobce, není soudce). [06]

3.7.2. Open source

Open source, nebo také open-source software (OSS) je počítačový software s otevřeným zdrojovým kódem. Otevřenost zde znamená jak technickou dostupnost kódu, tak legální dostupnost – licenci software, která umožňuje, při dodržení jistých podmínek, uživatelům zdrojový kód využívat, například prohlížet a upravovat. [07]

3.7.3. Free software

Svobodný software, někdy také nazývaný free software, je software, ke kterému je k dispozici také zdrojový kód, spolu s právem tento software používat, modifikovat a distribuovat. Naprostá většina svobodného software je zdarma, ačkoliv to není podmínkou.

Za získání kopií svobodného software můžeme platit, nebo je obdržet zdarma, ovšem bez ohledu na způsob, jak jsme je získali, máme vždy svobodu kopírovat a měnit software, dokonce prodávat nebo darovat jeho kopie nebo pozměněné verze. [08]

POJMY K ZAPAMATOVÁNÍ

Placené programy.

Volně šiřitelné programy.

Volná díla

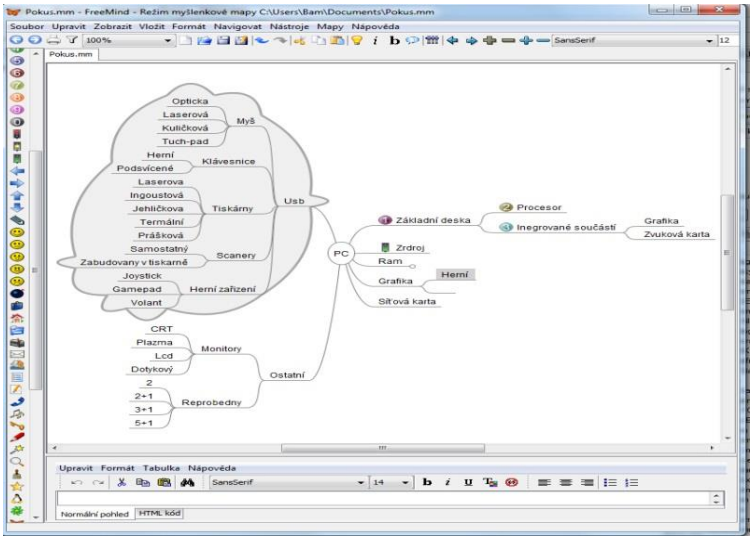
Všeobecná veřejná licence

ČAS POTŘEBNÝKE STUDIU 40 minut.

CÍL: Seznámení s grafickým zápisem vývojových diagramů a tvorbou myšlenkových map. Naučit se základní grafické symboly používané při tvorbě vývojových diagramů.

4. VÝVOJOVÉ DIAGRAMY

Každou myšlenku je vhodné nějakým způsobem reprezentovat. Říkáme tomu myšlenkové mapy. Zde si stanovujeme základní podmínky, co má dotyčný program dělat. Nejjednodušší je grafický náčrt daného problému. Hodně lidí to dělá na papíru, ale stále více uživatelů výpočetní techniky se přiklání k elektronickému vyjadřování pomocí počítačových programů. Jsou již programy, které pro specializované návrhy hlídají všechny závislosti a vazby. Najdeme však i programy, které jsou velmi intuitivní a pro základní grafické vyjádření postačují. Tyto programy bývají jak součástí větších kancelářských balíků, tak jednoúčelové. Najdeme i programy volně šiřitelné, které sice umí jen nejjednodušší závislosti, ale většinou pro první nástin problému stačí. Jedním z mnoha je open-source program FreeMind, který je napsaný v Javě. Můžeme zde nejen vytvářet podúrovně vazeb, ale doplňovat je velmi jednoduše ikonkami, barevně odlišovat a spojovat do jakýchsi seskupení.



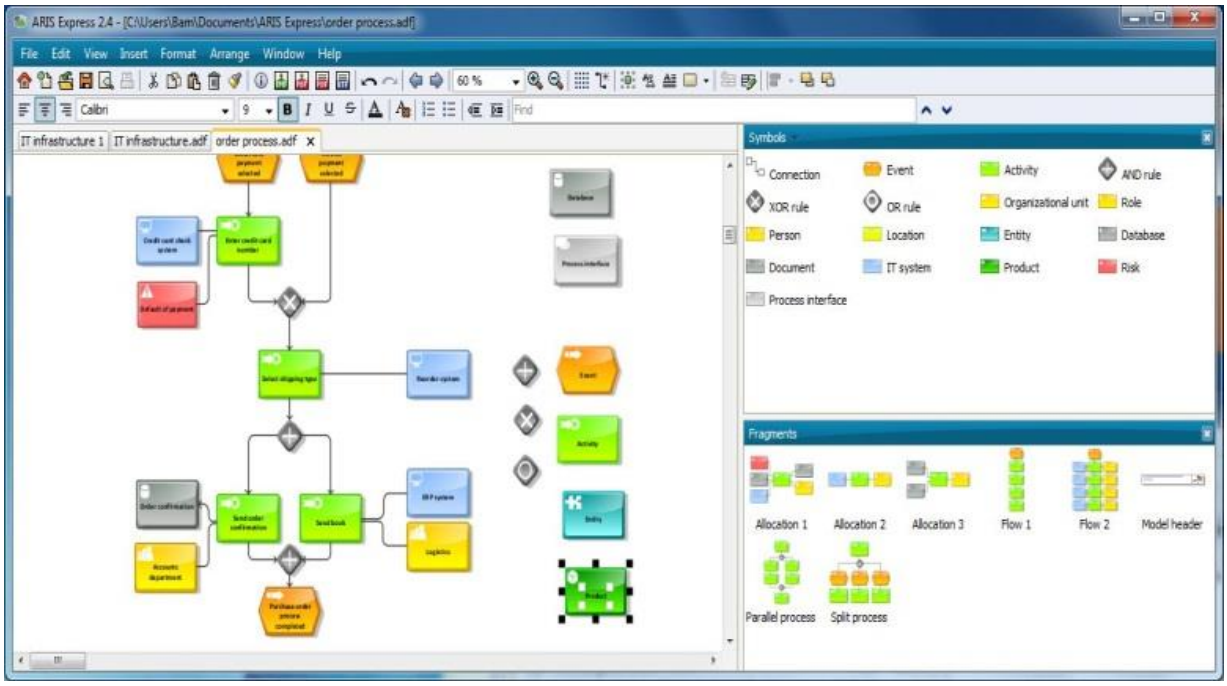
Obrázek 21 Ukázka grafického ztvárnění v programu FreeMind [1]

Další z řady dobře využitelných programů při vytváření grafického zobrazování je program ARIS Expres.

Tento program je směrován hlavně do obchodních a podnikatelských kruhů. Má dobře definovány prvky pro návrh IT techniky v podniku (vytváření mapy podnikové sítě).

Obrázek 22 Ukázka základního výběru v programu Aris [1]

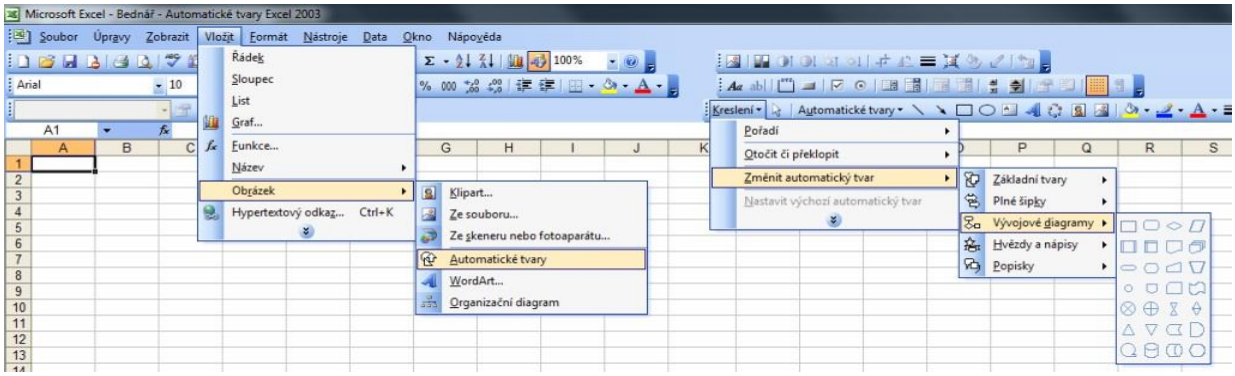
Tento program se dá s velkým úspěchem použít při vytváření diagramů na navrhovaný program, který bude vytvářet neprogramátor při svých požadavcích na programátora.



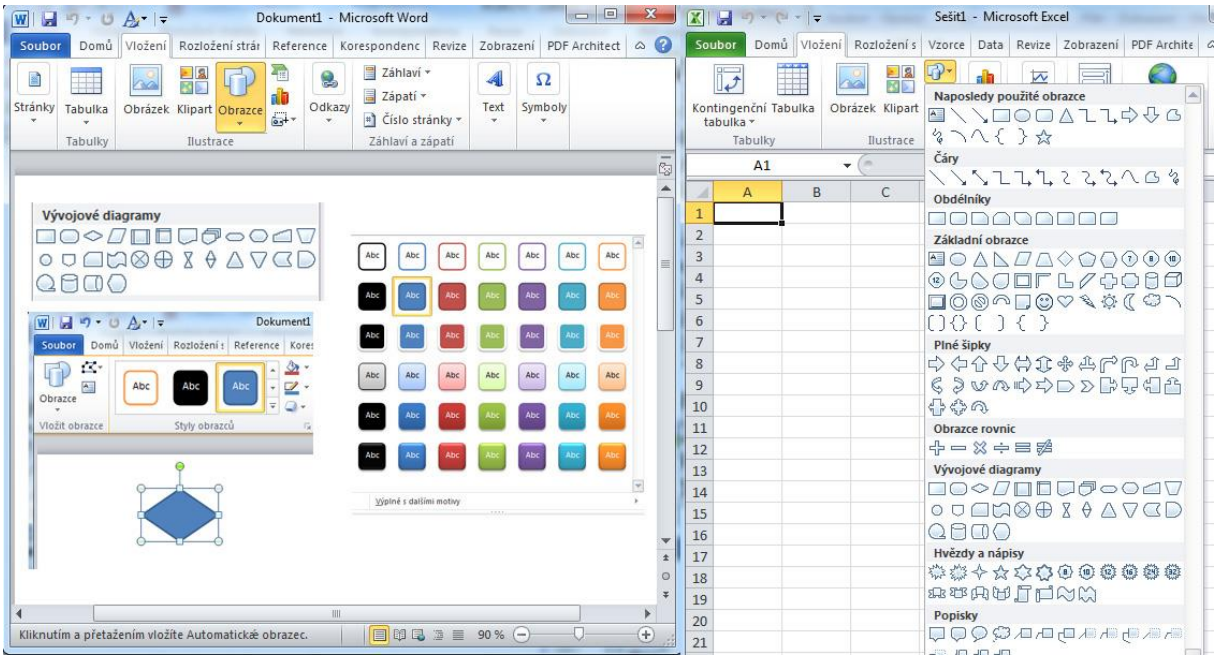
Obrázek 23 Ukázka základního výběru v programu FreeMind [1]

4.1. Vývojové diagramy v programování

Pro zakreslování vývojových diagramů používáme v programování již zavedené symboly. Můžeme použít tyto symboly z kancelářského balíku MS Office. Samozřejmě existuje více způsobů, jak se k automatickým tvarům dostat. Někomu vyhovuje nabídkové menu a jiný upřednostňuje výběr prvků z vrchní nabídkové lišty (můžeme si jej samozřejmě umístit jak na bok, tak na spodní část obrazovky, pouhým přetažením, podle toho, jak to uživateli vyhovuje).

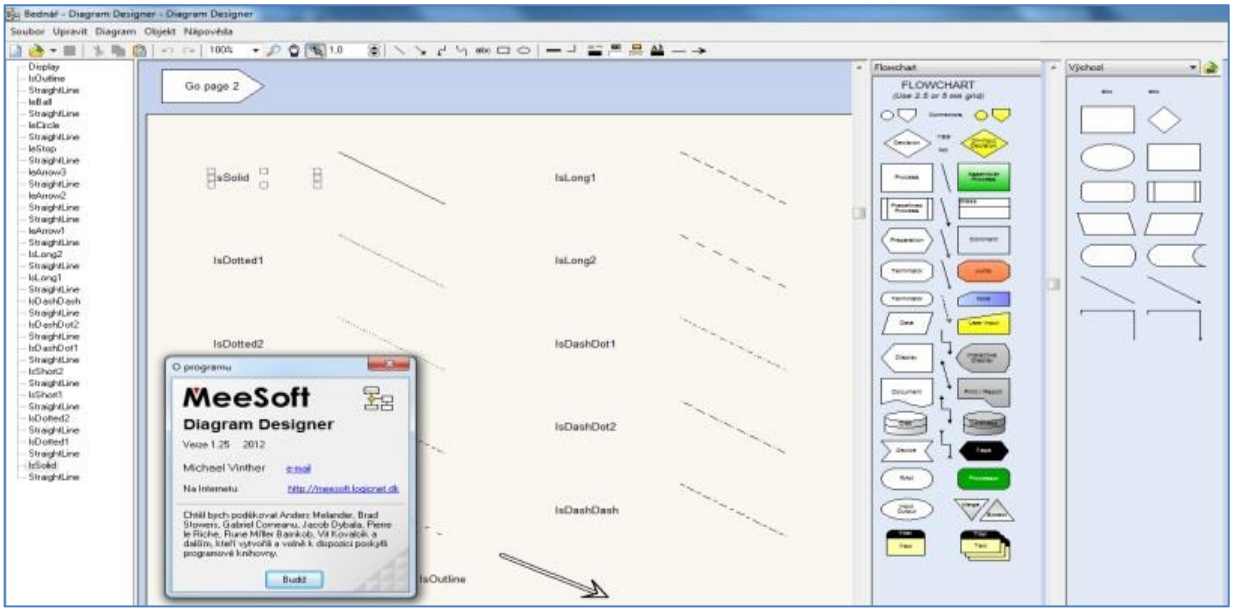


Obrázek 24 Ukázka umístění symbolů v Excelu 2003 [1]



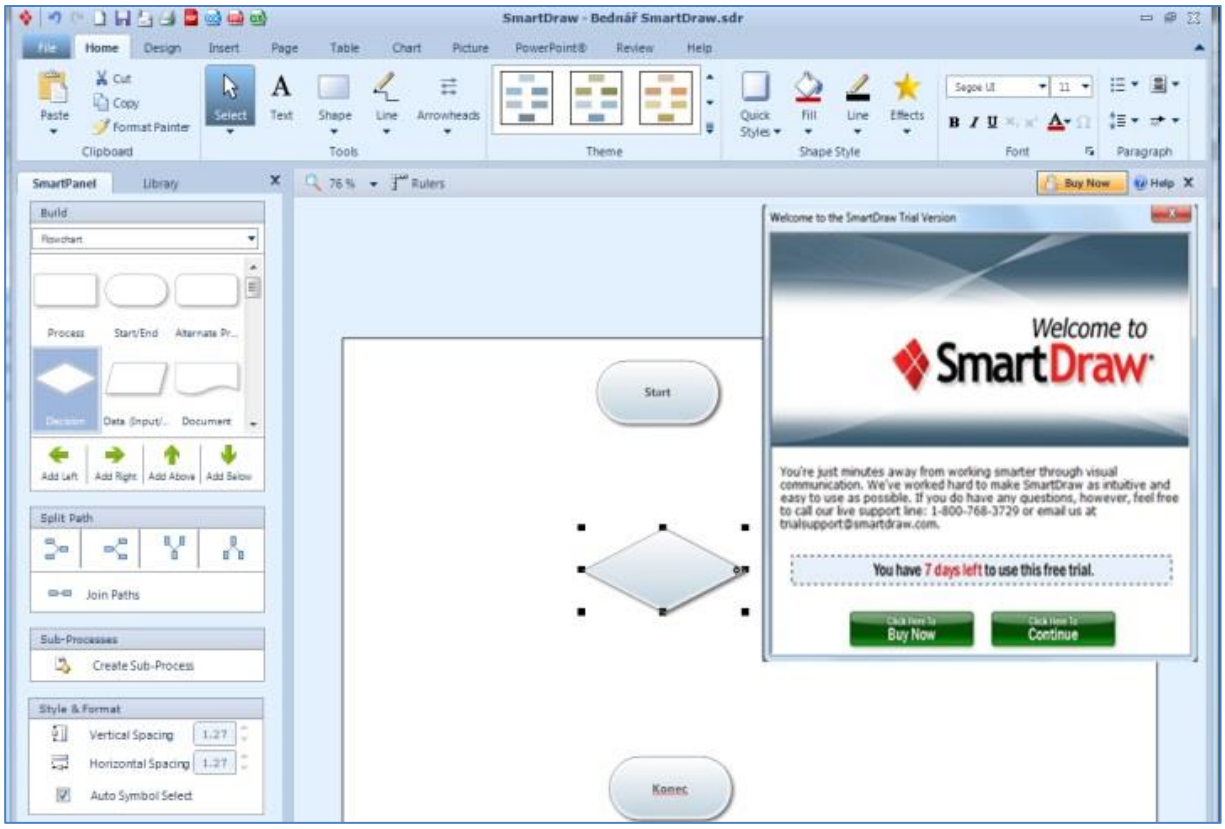
Obrázek 25 Ukázka symbolů pro vývojové diagramy ve Wordu a Excelu 2010 [1]

Pro účely nákresů vývojových diagramů vzniklo množství specializovaných programů. Není žádná úměra, že placené programy jsou lepší než neplacené. Záleží spíš na požadavcích uživatele. Jedním z volně použitelných programů pro specializované kreslení vývojových diagramů je program Diagram Designer

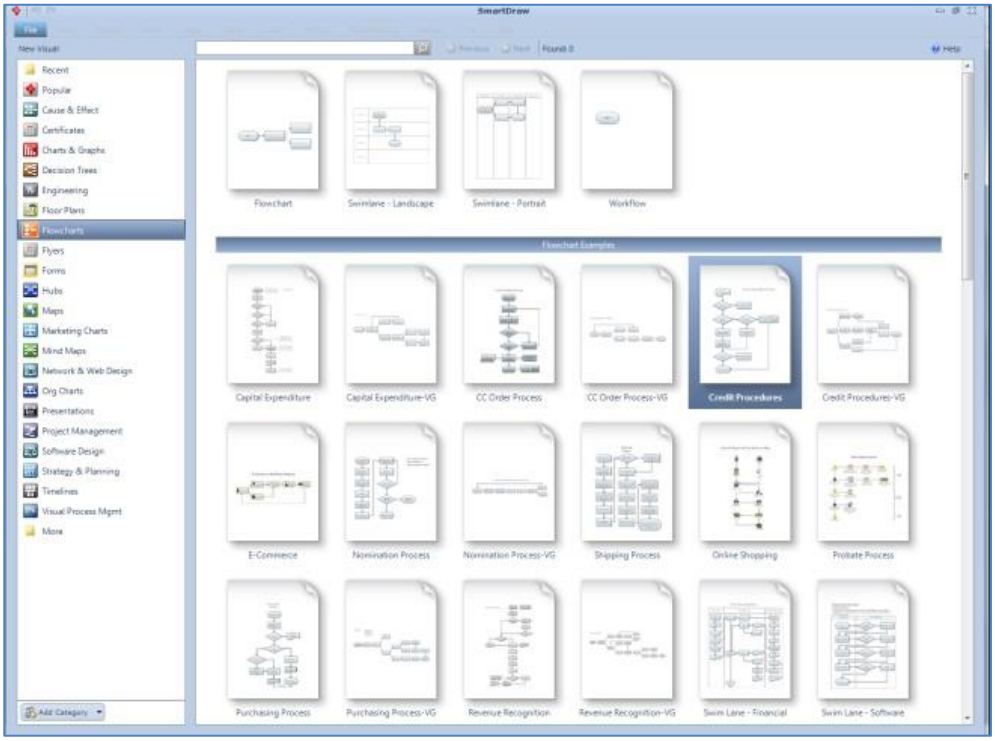


Obrázek 26 Vzhled programu Diagram Designer [1]

K řadě placených programů se řadí program SMARTDRAW. Uživatel po nainstalování dostává týdenní možnost si program vyzkoušet.



Obrázek 27 Vzhled programu SMARTDRAW [1]



Obrázek 28 Výběrové možnosti v programu SMARTDRAW [1]

4.2. Symboly používané při kreslení vývojových diagramů

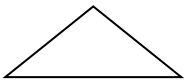
Pro vyjadřování struktury programu se nepoužívá slovní zápis algoritmu, který má dosti omezené možnosti. Byly tedy navrženy grafické symboly, které jsou unifikované a použitelné pro větší okruh uživatelů. Tyto značky nám umožňují graficky přehledně vyjádřit algoritmus programu pomocí vývojových diagramů. Dříve se kreslilo podle plastových šablonových pravítek. V dnešní době se kreslí vývojové diagramy na počítači.

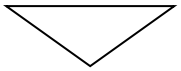
Základních grafické symboly, které se nejčastěji používají:

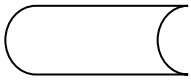
Značka přiřazení	(Postup)	b=0
Značka rozhodování	(Rozhodnutí)	c>1
Vstup nebo výstup dat	(Udaje)	Vypiš b
Konec (koncová značka)	(Ukončení)	
Značka podprogramu	(Předefinovaný postup)	

Ostatní grafické symboly:

Alternativní postup		Vnitřní paměť	
Dokument		Více dokumentů	
Ruční vstup		Ruční operace	
Spojka (spojnice)		Spojka mezi stránkami	
Štítek		Děrná páska	
Sumační bod		Nebo	
Porovnání		Řazení	

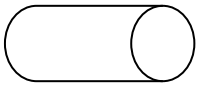
Vyjmutí 

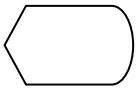
Sloučení 

Uložená data 

Zpoždění 

Magnetický disk 

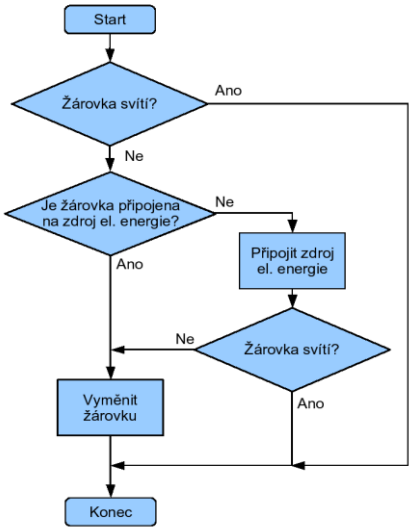
Paměť s přímým přístupem 

Zobrazení 

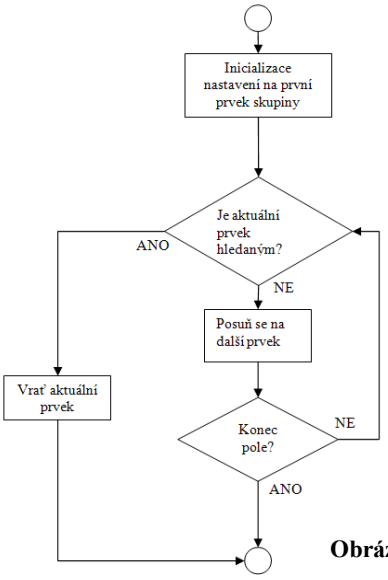
Paměť se sekvenčním přístupem 

Význam symbolů vývojového diagramu

Pro spojování symbolů používáme spojovací čáry, které mohou mít použité šipky. Tyto šipky nám ukazují směr zpracování algoritmu. Na následujícím obrázku je ukázán jednoduchý vývojový diagram (převzatý z Wikipedie). Znázorňuje nám jednoduchý postup, co se může dít, pokud zapneme vypínač a žárovka se nám nerozsvítí. Pokud žárovka po zapnutí vypínače svítí, tak se nic neřeší a graficky znázorněná cesta jde přímo k ukončení (konec). Pokud žárovka nesvítí, měli bychom zkontrolovat, zda máme v rozvodné síti za vypínačem napětí. Pokud ano, tak bychom měli vyměnit žárovku. Tento vývojový diagram je určen pro neelektrikáře a nedává nám na výběr, jak postupovat, pokud by na žárovce bylo fázové napětí, žárovka byla dobrá a stejně by nesvítila. Vývojový diagram je tedy určen pro konkrétní použití a nemusí mít v sobě zakomponovány všechny možné alternativy.



Obrázek 29 Ukázka vývojového diagramu pro svícení žárovky [10]



Na dalším obrázku je ukázka vývojového algoritmu pro sekvenční vyhledávání. Šipky se v nákresu, pokud je tok informací jednoznačný, nemusí kreslit. Musí se však použít v případě, že tento směr je jiný než by mohl navozovat dojem z nákresu, nebo když je třeba směr toku informace zvýraznit. Kruhy (spojka) na začátku a konci algoritmu nám naznačují, že se jedná o podprogram, který na něco navazuje.

Obrázek 30 Algoritmus sekvenčního vyhledávání [11]

ČAS POTŘEBNÝKE STUDIU 130 minut.

CÍL: Pochopit zápis algoritmu a jeho strukturu.

5. ALGORITMUS

Algoritmem by se dal nazvat určitý postup řešení daného problému. Je to předem daná posloupnost více kroků, ze které pak vzniká program. Program sám o sobě předepisuje počítači množinu procedur (funkcí, algoritmů), kterými se daný počítač řídí, aby provedl určitý výstup. Je to téměř jako recept, který předepisuje kuchaři postup, jak upéct např. jahodový koláč. [29]

Jednoduché postupy (algoritmy) můžeme zapsat slovně. U složitějších postupů je slovní popis značně nepřehledný a nesrozumitelný. Příkladem algoritmů zapsaného slovně jsou třeba návody natisknuté na obalech instantních polévek. Na některých návodech máme již grafické symboly, které mají předem definovaný význam. Nejrozšířenější formy grafického zápisu jsou vývojové diagramy a strukturogramy.

Aby mohl být algoritmus na počítači prováděn, musíme ho v počítači (respektive procesoru) předat "zakódovaný", tedy přeložený z řeči lidí do řeči strojů. Tak například čísla nebo znaky musíme počítači předávat ve dvojkové soustavě (tedy ty jedničky a nuly, jako v úvodu filmu Metrix). Protože člověk je tvor lenivý, vytvořil si program, který tohle všechno udělá za něj – tzv. překladač (angl. compiler). Ten je pro každý programovací jazyk odlišný, protože každý jazyk bude mít jiný způsob, jak program v něm napsaný převádět. [29]

5.1. Programovací jazyk

Programovací jazyk je nástroj, pomocí kterého zapisujeme algoritmus ve formě bližší člověku. Programovací jazyky dělíme na nízkoúrovňové – LLL (Low Level Language), a vysokoúrovňové – HLL (High Level Language). Nízkoúrovňové jazyky využívají především zápisu instrukcí procesoru. Pro své vlastnosti jsou používány hlavně odborníky v systémovém programování, a to především díky tomu, že umožňují větší kontrolu nad počítačem a hardwarem samotným. Nevýhodou může být hardwarová závislost. Mezi nízkoúrovňové jazyky bezpochyby patří Assembler. [29]

Vysokoúrovňové jazyky používají pro zápis algoritmů postupy bližší lidskému chápání. Algoritmus se zapisuje pomocí příkazů, které nakonec tvoří samotný program.

Ve většině programovacích jazyků je každý příkaz zakončen středníkem (;). Příkladem vysokoúrovňového jazyka by mohly být programy Eclipse, JCreator, NetBeans IDE, Turbo Pascal, [29]

5.2. Základní struktura algoritmu

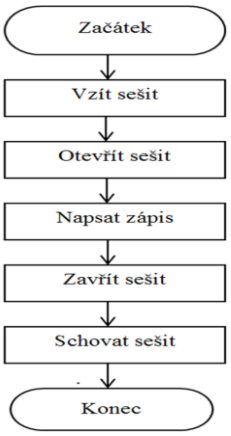
Algoritmus je vlastně přesně daný postup, který je zapsán tak aby byl srozumitelný jak programátorovi, ale hlavně počítacímu stroji, který nedokáže uvažovat a vykonává přesně zadané instrukce. Počítač, pokud nepůjdeme směrem do umělé inteligence, nemá možnost intuice ani samostatného uvažování. Musí zde být všechno co nejlépe popsáno a ošetřeny všechny možné případy. Pokud programátor opomine určitý stav zapsat a následně jej vyřešit, potom hotový program může vykazovat nestabilitu, pracovat nesprávně, či v určitém případě zamrznout či se zhroutit. Při zápisu základního algoritmu používáme tři základní programové struktury:

- sekvence (posloupnosti příkazů),
- rozhodování (větvení, výběr alternativy),
- opakování (podmíněné či nepodmíněné cykly, smyčky, interace).

5.2.1. Zápis posloupnosti příkazů (sekvence)

Posloupnost jsou definovány navazující kroky, jejichž pořadí je přesně předem pevně určeno. V posloupnosti nemůže být žádný krok přeskočen ani vynechán. Posloupnost je definována začátkem a koncem, mezi nimiž jsou určeny operace k provádění.

Příkladem pro vysvětlení sekvence může být zápis do sešitu. Musíme mít sešit, který otevřeme, a potom do něj můžeme teprve psát. Musíme dodržet přesně pořadí operací, abychom do sešitu něco zapsali. Nemůžeme napřed třeba začít psát, aniž bychom napřed nevzali sešit a neotevřeli jej na stránce určené k zápisu. Posloupnost se v algoritmech objevuje samostatně, nebo jako další součást složitějších struktur (rozhodování, cyklů, smyček).



Obrázek 31 Algoritmus sekvence [1]

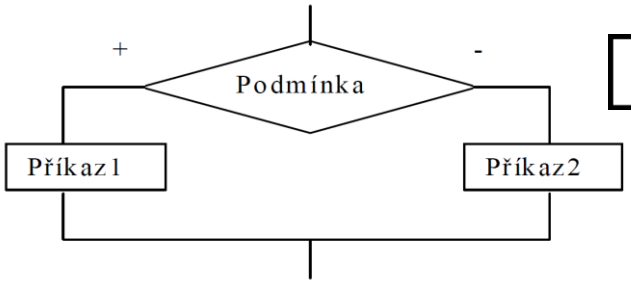
5.2.2. Rozhodování (větvení, výběr alternativy)

Větvení použijeme tam, kde podle okolností mají být některé kroky v posloupnosti vynechány, přidány nebo nahrazeny jinými. Větvení obsahuje obvykle tři části. První částí je otázka, na kterou existuje kladná a záporná odpověď. Druhou částí je krok, který se provede v případě kladné odpovědi na otázku. Třetí částí je krok, který se provede v případě záporné odpovědi na otázku. První část větvení (otázka) je povinná, zbylé dvě části jsou nepovinné. Pokud však současně chybí druhý i třetí krok, ztrácí větvení smysl. [12]

Při zápisu algoritmu ve vývojovém diagramu používáme tři typy větvení. Jde o úplné větvení, neúplné větvení a vnořené větvení.

Úplné větvení

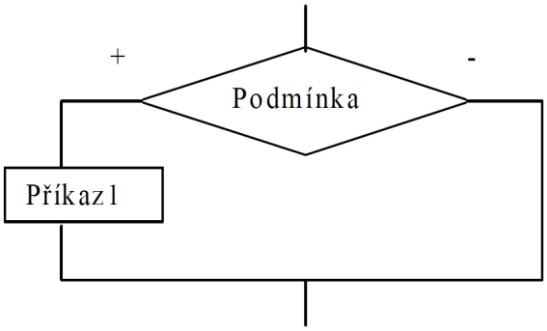
Úplné větvení znamená, že jsou zařazeny kroky pro kladnou i zápornou odpověď. Podle toho, jak je sestavena podmínka, budou zařazeny další činnosti. Je proto nutné nějakým způsobem označit, která větev se provádí v případě, že podmínka je pravdivá (říkáme také že podmínka platí) a která větev se provádí, když podmínka není pravdivá (neplatí). V dalším textu je vždy pravdivost podmínky označena symbolem + (plus) a nepravdivost symbolem – (mínus). [12]



Obrázek 32 Algoritmus úplné větvení [12]

Neúplné větvení

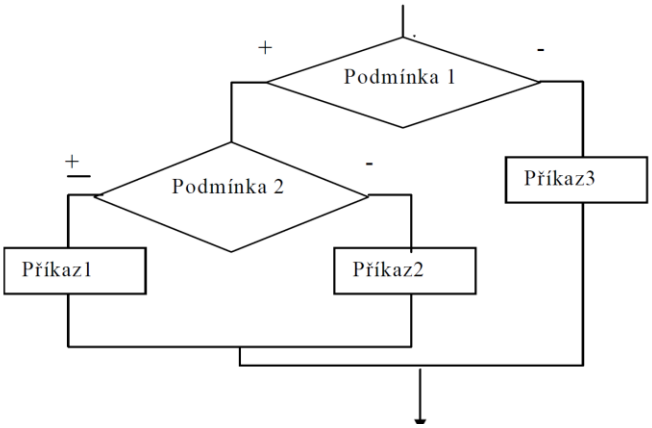
Neúplné větvení znamená, že chybí krok pro kladnou nebo pro zápornou odpověď (častější je situace, kdy chybí krok pro zápornou odpověď). V logickém sledu činností sice chybí krok pro zápornou odpověď, ale ve vývojovém diagramu je nutné tuto skutečnost zakreslit. [12]



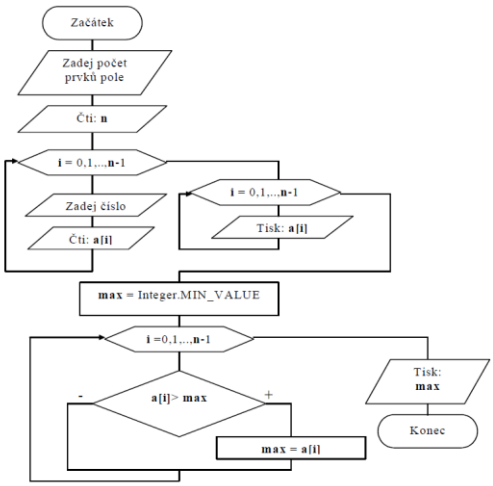
Obrázek 33 Algoritmus neúplného větvení [12]

Vnořené větvení

Vnořené větvení znamená, že krok pro kladnou nebo pro zápornou odpověď (nebo pro obě odpovědi) je tvořen opět větvením. [12]



Obrázek 35 Algoritmus vnořeného větvení [12]



Obrázek 34 Algoritmus vyhledání maximální hodnoty mezi n celými čísly. Čísla jsou uložena do paměti ve formě pole [13]

5.2.3. Opakování (cykly)

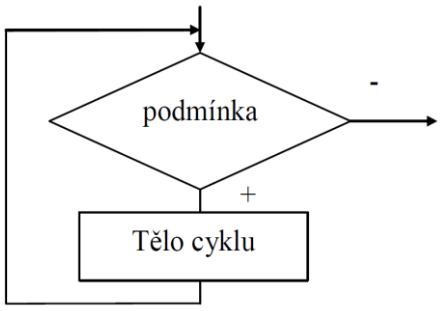
V algoritmech velmi často nastává situace, kdy musíme některé činnosti zopakovat. To, zda se opakování provede či nikoliv, závisí vždy na vyhodnocení určité podmínky (otázky). Buď přesně známe, kolikrát se má činnost opakovat, pak podmínkou kontrolujeme, zda již bylo opakování provedeno v potřebném počtu. Nebo opakování závisí na vzniku určité situace, např. při výpočtu dojde k překročení nějaké extrémní hodnoty, pak podmínkou kontrolujeme vznik této situace. Existují dva základní typy cyklů, a sice cyklus, který nejdříve vykoná určité příkazy a pak teprve vyhodnocuje podmínku opakování, a pak cyklus, který nejdříve vyhodnocuje podmínku opakování a pak provádí příkazy. V dalším výkladu je zařazen ještě cyklus s řídící proměnnou, který je určitým zjednodušením cyklu s podmínkou na začátku. Tento cyklus je velkým zjednodušením práce programátora. Budeme tedy rozlišovat 3 typy cyklů:

- cyklus s podmínkou na konci (s výstupní podmínkou),
- cyklus s podmínkou na začátku (se vstupní podmínkou),
- cyklus s řídící proměnnou. [12]

- **Cyklus s podmínkou na začátku (se vstupní podmínkou)**

Cyklus s podmínkou na začátku je nejčastěji používaný cyklus. Může se stát, že u tohoto cyklu se tělo cyklu neprovede ani jednou. To nastane, v případě, že podmínka má hodnotu false, již při prvním vyhodnocení.

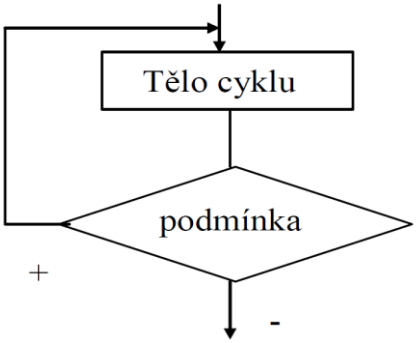
Nejdříve dojde k vyhodnocení podmínky. Má-li podmínka hodnotu true, provede se tělo cyklu (jeden nebo více příkazů) a pak se provede automaticky návrat k podmínce, ta se opět vyhodnotí. Pokud má podmínka opět hodnotu true, celá činnost se opakuje. Cyklus je ukončen až tehdy, když podmínka nabude hodnoty false. [12]



Obrázek 36 Cyklus se vstupní podmínkou [12]

- **Cyklus s podmínkou na konci (s výstupní podmínkou)**

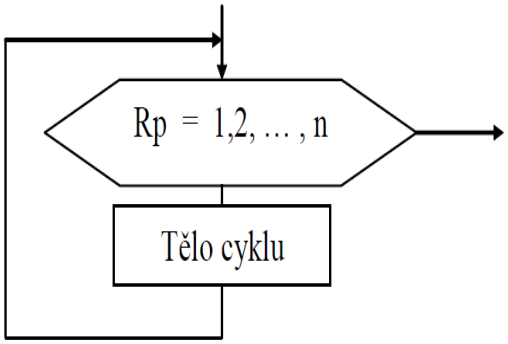
Nejdříve se vykonají příkazy těla cyklu, pak se provede vyhodnocení podmínky. Má-li podmínka hodnotu true (není pravdivá), provede se návrat na začátek těla cyklu, provedou se příkazy těla cyklu a opět se vyhodnotí podmínka. Tato činnost se opakuje tak dlouho, až podmínka nabude hodnotu false (je pravdivá). Pak je cyklus ukončen. [12]



Obrázek 37 Cyklus s výstupní podmínkou [12]

- **Cyklus s řídící proměnnou**

Cyklus s řídící proměnnou je poněkud zjednodušený cyklus se vstupní podmínkou. Lze jej použít pouze tehdy, jestliže počet opakování je dán explicitně a nezávisí na činnosti prováděné v těle cyklu. Cyklus je řízen řídící proměnnou. Hodnoty řídící proměnné jsou omezeny počáteční (initial) a koncovou (final) hodnotou cyklu. Na začátku provádění cyklu se do řídící proměnné uloží počáteční hodnota.



Obrázek 38 Cyklus s řídící proměnnou [12]

Pokud je pak hodnota řídící proměnné menší nebo rovna koncové hodnotě, provedou se tyto činnosti:

- vykoná se tělo cyklu,
- provede návrat na začátek cyklu,
- automaticky se zvýší hodnota řídící proměnné o 1.

Pokud je hodnota řídící proměnné menší nebo rovna koncové hodnotě, celá činnost se opakuje. Cyklus končí tehdy, když řídící proměnná dosáhne hodnoty vyšší než je hodnota koncová.

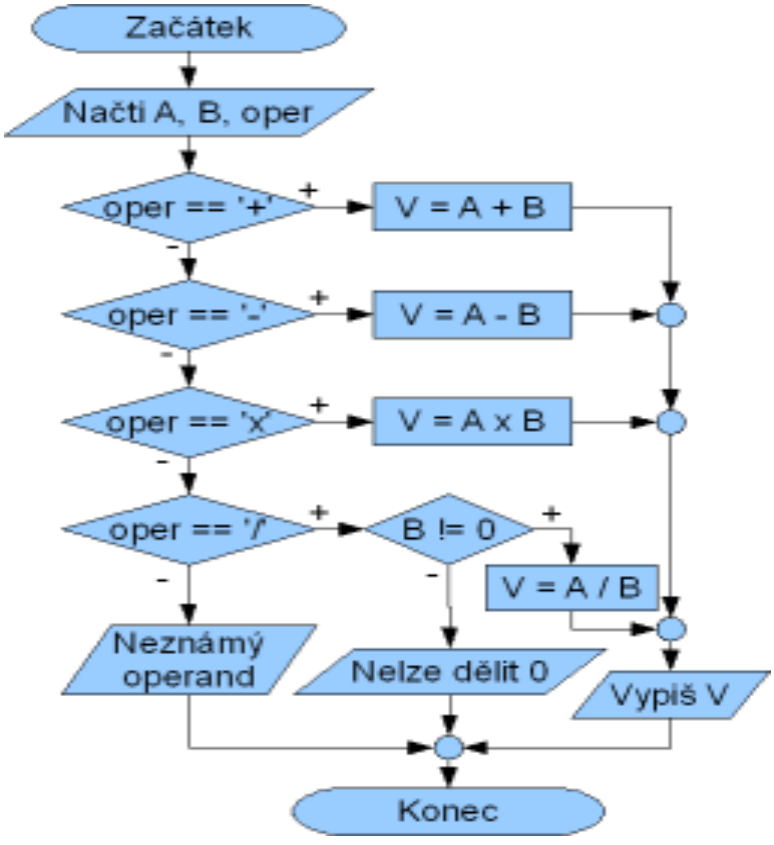
Nejvíce obecný je cyklus s podmínkou na začátku a dá se říci, že by nám úplně stačil jen tento cyklus pro vyřešení všech cyklických situací. Ostatní dva cykly jsou zařazeny pro zjednodušení práce programátora a nesou s sebou při použití určitá omezení či rizika. Omezení se týká cyklu s řídící proměnnou, kdy musíme předem znát počet opakování. Určitá rizika jsou spojena s cyklem s výstupní podmínkou, kdy může dojít ke zkreslení výsledku vlivem toho, že vždy alespoň jednou se tělo cyklu vykoná. [12]

SHRNUTÍ POJMŮ

……

OTÁZKY

1. Jaké vlastnosti by měl mít algoritmus?
2. Jaká omezení vzniknou, pokud při sestavení algoritmu použijeme pouze posloupnost příkazů?
3. Kdy do algoritmu zařazujeme větvení?
4. Jaký je rozdíl mezi úplným a neúplným větvením?
5. Kdy je vhodné použít vnořené větvení?
6. K čemu slouží v algoritmu cyklus?
7. Jak se provádí cyklus s podmínkou na konci?
8. Jak se provádí cyklus s podmínkou na začátku?
9. Jak se provádí cyklus s řídící proměnnou?
10. Jaká omezení má cyklus s řídící proměnnou?
11. Které z cyklů je sestaven tak, že se tělo cyklu nemusí provést ani jednou?
12. Který z cyklů je sestaven tak, že se tělo cyklu provede vždy alespoň jednou? [12]
13. Zkuste popsat tento algoritmus:



Obrázek 39 Algoritmus jednoduché kalkulačky [14]

ČAS POTŘEBNÝ KE STUDIU 20 minut.

CÍL: Co je to ccccccccccccccc

5.3. UML – unifikovaný modelovací jazyk

Zkratka tohoto jazyka napovídá, že jde unifikovaný modelovací jazyk (UML neboli Unified Modeling Language), který má, na rozdíl od převážně textově orientovaných programovacích jazyků, vlastní grafickou syntaxi (pravidla a vazbové systémy pro sestavování jednotlivých dílčích bloků do větších celků) a sémantiku (pravidla která určující jednotlivým výrazům jejich význam). Nejedná se přímo o programovací jazyk, avšak ve vývojových prostředích (CASE nástrojích) využívajících UML jsou často integrovány i generátory zdrojového kódu v různých programovacích jazycích (C++, JAVA).

Při použití UML jako pseudo programovacího jazyku vývojář nakreslí UML diagramy, ze kterých se vygeneruje přímo spustitelný kód.

V současné době má jazyk UML největší význam při návrhu softwarových systémů, protože objektově orientovaný návrh každé složitější aplikace je nezbytným předpokladem pro její úspěšnou a rychlou implementaci. Pro objektově orientovaný návrh je samozřejmě možné použít různé podpůrné prostředky, zejména další odlišné typy diagramů. Je již z principu UML nutné, aby vytvářené diagramy a grafy měly vnitřní konzistenci a přesně danou sémantiku, což u jiných typů grafů nemusí být obecně zaručeno. UML diagramů existuje několik typů lišících podle toho, jaké se pomocí nich plánují či zpracovávají úlohy. Tyto diagramy se od sebe odlišují především repertoárem použitých značek, způsobem jejich vzájemného propojení a s nimi související sémantikou.

Mezi velké přednosti jazyka UML i na něm postavených UML diagramů patří existence otevřeného a rozšiřitelného standardu, podpora celého vývojového cyklu aplikace či jiného (ne nutně programového) systému a velká podpora pro různé aplikační oblasti. Pro širší využití jazyka UML v praxi mluví také významný fakt, že je podporován celou řadou vývojových nástrojů, ať už samostatných aplikací určených pouze pro práci s UML, nebo i integrovaných vývojových prostředí (IDE), které v některých případech dovolují provádět převod informací mezi UML diagramem a algoritmem zapsaným v programovacím jazyce (a samozřejmě i opačným směrem, ten je však z pochopitelných důvodů složitější a ne vždy uskutečnitelný).

Celý jazyk UML je založený na třech elementech, které ale nejsou z uživatelského hlediska reprezentovány v textové podobě, ale grafickými značkami v plošném (tj. dvourozměrném) grafu.

Tři základní elementy jazyka UML se dle své funkce nazývají **předměty, relace a diagramy**.

ČAS POTŘEBNÝKE STUDIU 300 minut.

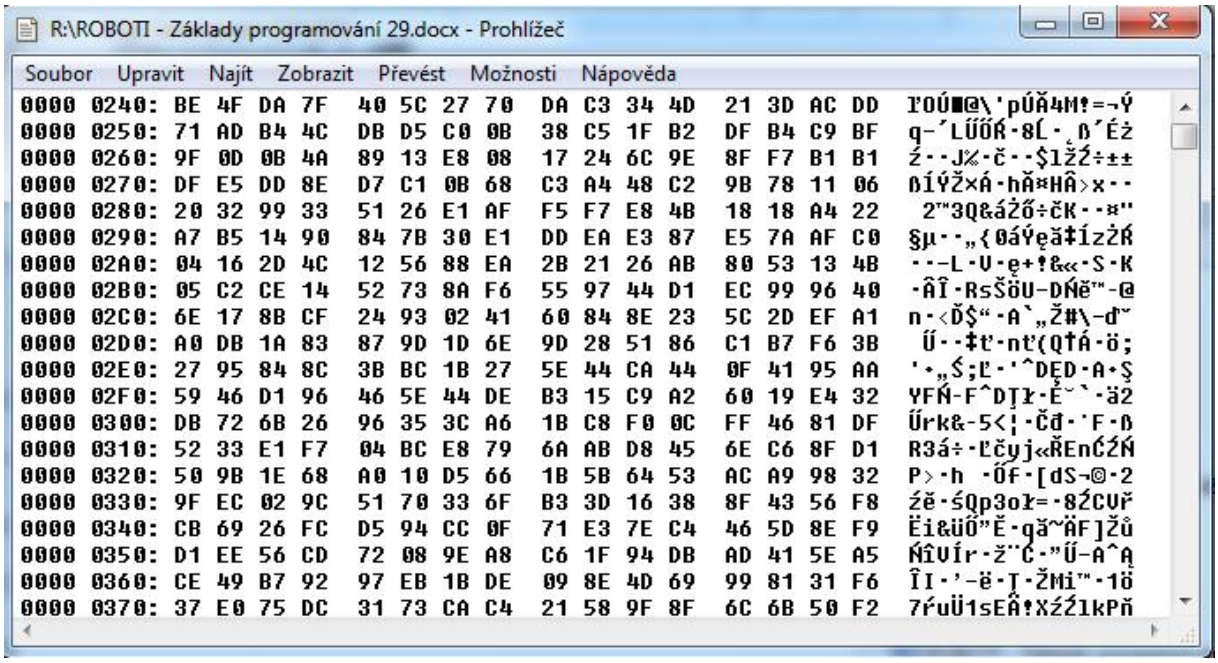
CÍL: Seznámeni se strojovým kódem, asamblerem, a programovacími jazyky.

5.4. ROZDĚLENÍ PROGRAMŮ

Počítačový program je prostředek, jak přinutit počítač, aby provedl programátorem sestavený výpočetní algoritmus. Tak jak mohou být rozmanité zápisy algoritmů, tak rozmanité mohou být prostředky na sestavení tohoto programu. Buď programátor zná dokonale konkrétní počítač, tj. jeho strukturu, ovládání řídících operací, ovládání výpočetních operací, práci s pamětí a vstupními a výstupními zařízeními. V tom případě může program sestavit přímo ve strojovém kódu. Ve svém algoritmu používá jednotlivé strojové instrukce, které jsou definovány konstruktérem počítače, zejména části která je centrální jednotkou, nazývanou procesor. Instrukce, někdy příkazy, určují, jaké operace se provádějí s obsahem registrů, ve kterých jsou uložena zpracovávaná data. Vlivem binárního provedení elektronických obvodů se ustálila reprezentace dat ve formě slabik (bytů) a slov, složených ze slabik. [14]

5.5. Strojový kód

Každý počítač dokáže zpracovávat jen určitý soubor instrukcí jemu vlastní. (V novější době sdílí často několik příbuzných typů počítačů tentýž strojový kód; mluvíme pak o rodině počítačů).



Obrázek 40 Zápis ve strojovém kódu

Program ve strojovém kódu se skládá z jednoduchých příkazů – instrukcí.

Instrukce má většinou dvě základní části – kód operace, který udává, co se má udělat, a adresy (někdy i více adres), který říká, s jakými daty se má operace provést. Instrukce jsou zapsány čísly (navíc zapsanými ve dvojkové či šestnáctkové soustavě), strojový kód je proto pro člověka velmi nesrozumitelný. Z toho důvodu se v něm programuje jen zcela výjimečně a byla postupně vytvořena řada programovacích jazyků, které jsou pro člověka přece jenom srozumitelnější. Program zapsaný v takovém jazyce ovšem počítač přímo vykonávat neumí, a musí být proto speciálním programem, tzv. překladačem, přeložen do strojového kódu, který se teprve může provést.

V prvopočátcích se používal pouze strojový kód. Tyto zápisy byly ovšem příliš dlouhé, náročné na programátora, a velmi obtížně se hledaly chyby. Vznikl jazyk symbolických adres označovaný jako Assembler. Protože je ale programování pomocí čísel velmi náročné a nepohodlné, byl stvořen jazyk assembler. V tomto jazyce je každému instrukčnímu kódu přiřazen člověku srozumitelnější příkaz. [31]

Protože procesory mají omezený počet instrukcí zaměřený na základní matematické operace, je psaní programů ve strojovém kódu úmorné, zejména pokud potřebujeme některé operace opakovat a podobně. Proto vzniklo vyšší programovací prostředí, asemblery. To ulehčilo práci programátora, avšak stále byl asembler poplatný konstrukci procesoru. Situaci poněkud vylepšily tzv. autokódy, které umožnily programátorovi obecněji zapisovat požadované operace. Je třeba si uvědomit, že procesor dovede vykonat operace zapsané ve strojovém kódu. Pro program napsaný v asembleru, autokódu nebo vyšším programovacím jazyku je potřeba provést "překlad" do strojového kódu. Překladače mohou pracovat samostatně, tj. napřed se provede překlad z programovacího jazyka do tzv. spustitelného kódu, a potom se tento kód spustí. Nebo se překlad provádí v průběhu výpočtu např. po jednotlivých programových řádcích, a to jsou interpretery, jejich představitelem je programovací jazyk Basic a jeho varianty. Od této úrovně už programátor při realizaci svých algoritmů nepotřebuje důkladnou znalost procesoru. Stačí znalost vybraného programovacího jazyka, lépe vyjádřeno s ohledem na nabízené možnosti programovacího prostředí. [14]

Z doby rozvoje výpočetní techniky se zachovaly některé požadavky, které je dobré respektovat i nyní:

- program nechť je krátký, přehledný a do detailu okomentovaný,
- využívat optimálně paměť,
- využívat jednoduché a rychlé instrukce,
- kontrolovat dobu provádění programu.

Větší projekty rozdělit do menších částí, které lze odladit samostatně.

5.6. Assembler

Assembler (neboli jazyk symbolických adres) je nízkoúrovňový programovací jazyk, který umožňuje psát programy přímo ve strojovém kódu procesoru. Program napsaný v assembleru se skládá z instrukcí. Jedna instrukce je jakýmsi povelem pro procesor, aby vykonal určitou činnost. Programy psané v assembleru není možné přenést na jiný procesor, než pro který byl napsán, pokud oba jsou procesory vyrobeny jinou architekturou. Program napsaný pro procesor Intel Pentium je například možné spustit na procesorech AMD odpovídající řady, ale nikoliv už na procesorech Motorola nebo Power PC. Blízkost assembleru s procesorem umožňuje velice dobrou optimalizaci programu, a to jak ve velikosti, tak v rychlosti. Nevýhodou assembleru je zdlouhavost psaní rozsáhlejších programů a také již uváděná nepřenositelnost mezi procesory (Programy napsané v C/C++ po překompilování přenositelné jsou). [31]

Assembler (oficiálně česky Jazyk symbolických adres) vznikl někdy na konci čtyřicátých let minulého století. Patří mezi nižší programovací jazyky (strojově orientované). To znamená, že "slova" jazyka víceméně odpovídají instrukcím strojového kódu. Jejich podoba je však samozřejmě zapamatovatelnější (alespoň pro ty, kteří umějí anglicky). Kompilátor má tedy podstatně jednodušší práci než u vyšších jazyků (to jsou prakticky všechny ostatní). Výhodou je vysoká rychlost kompilace, a někdy i programu. [32]

```
.data

dwValue    dd      12345678h            ; Proměnná o velikosti 32 bitů

.code

WinMain PROC    hInstance    :HANDLE,
                hPrevInstance:HANDLE,
                lpszCmdParam :LPSTR,
                nCmdShow     :WORD

        mov     eax, 1                   ; Naplní registr EAX hodnotou 1
        mov     ebx, eax                 ; Zkopíruje obsah registru EAX do registru EBX
        mov     ecx, dword ptr dwValue   ; Naplní obsah registru ECX hodnotou uloženou
                                         ; v proměnné dwValue
        mov     ah, al
        mov     ch, cl
        mov     bx, ax
        ret
WinMain ENDP
```

Obrázek 41 Ukázka zápisu v Assembleru [31]

Většina příkazů assembleru přímo odpovídá instrukcím mikroprocesoru. Procesor má definovanou určitou množinu instrukcí a veškerá jeho činnost spočívá v jejich provádění. Každá instrukce je reprezentována instrukčním kódem, což je obyčejné číslo, které se zapíše do programové paměti. Procesor při své práci programovou paměť prochází, čte instrukční kódy a provádí odpovídající instrukce. Tyto instrukce jsou vyjmenovány a popsány v instrukční sadě, kterou můžeme najít u dokumentace k mikroprocesoru.

I assembler má ale svoje nevýhody. Je nepřenosný a je příliš blízko hardwaru. Pokud napíšeme v assembleru program pro konkrétní typ mikroprocesoru, nemáme v žádném případě zaručeno, že takový program bude fungovat i na jiných procesorech. Instrukční sada procesoru je navíc velmi omezená a složená pouze z primitivních instrukcí, proto je psaní složitějších programů v assembleru obtížné a zdlouhavé. Pro odstranění těchto neduhů máme vyšší programovací jazyky třeba C++. Tento jazyk je uživatelsky mnohem přístupnější, není nutné psát hromady assemblerovských instrukcí. Navíc je přenosný, protože program v C++ před spuštěním přeložíme v překladači, který jej převede na assemblerovský kód. My pouze překladači řekneme, pro který typ mikroprocesoru chceme překládat.

Přestože jazyk C++ přináší mnohé výhody, na některé věci je přece jen lepší assembler. Při programování v C++ jsme totiž příliš vzdáleni od hardwaru a nemáme přehled o tom, co vlastně procesor provádí za operace. Překladač náš kód přeloží do assembleru (tedy rozloží na instrukce) dle svého uvážení a my to nemůžeme nijak ovlivnit. U některých operací by se nám však velmi hodilo, kdybychom mohli procesoru sami diktovat instrukce, např. při přístupu k hardwaru. [31]

5.7. Značkovací jazyky

Značkovací jazyk je jakýkoli jazyk, který vkládá do textu značky vysvětlující význam nebo vzhled jednotlivých jeho částí. Vzhledové značky se původně používaly jen pro formátování textu v nakladatelstvích. Dodnes se používá formátovací jazyk TeX (formátování knih do tisku). Dalšími jazyky jsou Troff, PDF.

Z důvodů špatné kompatibility mezi jednotlivými formátovacími jazyky se začaly vyvíjet obecné značkovací jazyky. Jsou určeny k vytváření dalších značkovacích jazyků (jejich aplikací). Obecné vlastnosti těchto jazyků jsou dobře přenosné mezi různými operačními systémy (nejsou na žádném operačním systému závislé), stejně tak nejsou závislé ani na způsobu zobrazení (na tom mohou být závislé až jejich aplikace), samy o sobě obsahují jen informace o struktuře dokumentu (ne o jeho formátování, tzn. ne jak se má zobrazit). Díky tomu mohou být zobrazené v mnoha médiích (počítač, mobil, tiskárna). Značky vkládané do textu se zde nazývají tagy. Stejně tak se nazývají i u aplikací těchto jazyků.

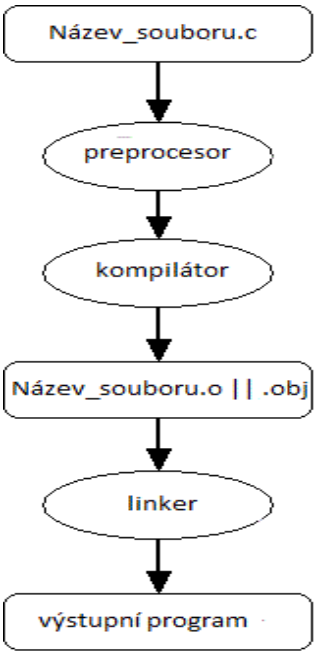
Při vytváření dalšího formátu si jeho tvůrce vymyslí vlastní tagy, které potom určí v tzv. DTD (Document Type Definition – definice typu dokumentu), která popisuje vlastnosti jednotlivých tagů (kde se mohou nalézat, zdali jsou povinné, ale nepopisuje například, jak se mají zobrazit) a vztahy mezi nimi.

XML, HTML, WML apod. nejsou programovací jazyky, ale jazyky značkovací.

Nepleťte si značkovací a programovací jazyky.

5.8. Vývojové prostředí (IDE) a kompilace

IDE neboli vývojové prostředí, slouží pro překlad algoritmů napsaných v nějakém vyšším programovacím jazyce do jazyka strojového neboli strojového kódu pomocí kompilátoru. Kompilátor je program, který nám převádí (kompiluje) náš zdrojový programový kód z jazyka vstupního do jazyka výstupního. Poté pomocí linkeru nám poskládá objektový kód knihovny, tak aby program šel pustit. [43]

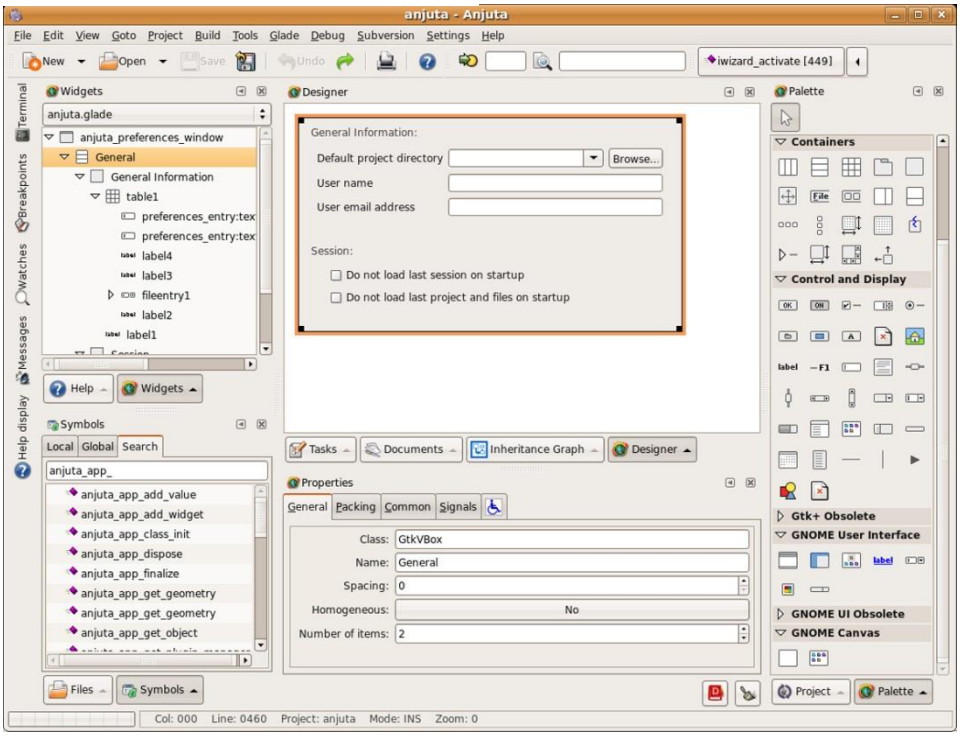


Vstupem do kompilátoru je náš zdrojový kód s příponou *.c. Ten se dostane do preprocesoru který nám "před připraví" náš kód (například smaže komentáře) poté kód vstupuje do kompilátoru.

Kompilátor nám vytvoří program v objektovém kódu, který ještě nejde spustit.

Poté kód vstupuje do tzv. linkeru a právě linker nám poskládá už objektový kód, knihovny tak aby program nyní šel spustit.

Obrázek 42 Znázornění kompilace [43]



Obrázek 43 IDE (vývojové prostředí) pracujícím na operační systému Linux Anjuta pro programování v C / C + +

5.9. Programovací jazyky

Programovací jazyky jsou jazyky sloužící k tvorbě počítačových programů (programování). Programování je proces algoritmizace dané úlohy, tj. vytváření postupu, jenž vede k řešení dané úlohy.

XML, HTML, WML apod. nejsou programovací jazyky, ale jazyky značkovací.

Programovací jazyky se dají rozdělit podle mnoha kritérií. Nejčastější způsoby jsou na vyšší a nižší.

Nižší programovací jazyky jsou jazyky primitivní, jejichž instrukce (víceméně přesně) odpovídají příkazům procesoru. To znamená, že procesor bude vykonávat ty instrukce, které programátor napíše. Jsou závislé na svém procesoru a nepřenositelné na jiný (nepříbuzný) procesor (program z 386 na Pentiu pojede, ale na Atari (s procesorem Motorola) ne).

V praxi to vypadá tak, že programátor musí vypisovat každou maličkost, i jednoduchý program má neúměrně složitý zdrojový kód. Výhodou je, že programátor má takto přístup i k funkcím počítače, které by měl ve vyšším programovacím jazyce nedosažitelné.

Patří sem jazyk symbolických adres (Assembler) a strojový kód (to, co uvidíte, když se vám podaří zobrazit obsah exe souboru v textovém editoru). Zvláštním typem nižšího jazyka je tzv. autokód, který spojuje prvky nižších a vyšších jazyků. Vznikl rozšířením Assembleru o jednoduché příkazy pro často používané skupiny instrukcí.

Vyšší (problémově orientované) programovací jazyky jsou podstatně srozumitelnější, struktura jejich zdrojových kódů je logická, nejsou závislé na strojových principech počítače. Do strojového kódu se převádějí kompilátorem (případně se rovnou spouštějí interpretrem). V praxi je vyšší programovací jazyk vše, co není Assembler, to znamená: Pascal, Basic, Prolog, Lisp, Algol, Fortran atd.

Často se uvádí, že jazyk C je jakýmsi přechodem mezi vyššími a nižšími jazyky, má však blíže k vyšším.

Imperativní, logické a funkcionální jazyky

Imperativní (též procedurální) jazyky jsou téměř všechny jazyky, které se běžně používají. K řešení úlohy se používá algoritmu (postupu, jak se má daná úloha vyřešit). Např. Pascal, C, Basic, PHP, Perl, Java.

U logických jazyků programátor pouze popíše daný problém pomocí logických výroků. Program z nich potom vyvozuje požadované informace.

Někdy se používají ke tvorbě umělé inteligence, k praktickému programování jsou nevyužitelné, např. Prolog.

Ve funkcionálních jazycích se vše popisuje pomocí funkcí, často neexistují proměnné, program je vlastně jen složitou soustavou funkcí. Často se pracuje se seznamy. Tento způsob programování je bližší klasické matematice, obecně je však velmi nepřehledný. Avšak pro některé problémy jsou tyto jazyky vhodnější než imperativní. Např. Lisp, Haskell, Miranda.

Logickým a funkcionálním jazykům se někdy společně říká deklarativní jazyky (také neprocedurální). Některé jazyky mají prvky funkcionální i imperativní. Například víceméně funkcionální jazyk Schéma dovoluje programovat i imperativně, naopak imperativní Python umožňuje i funkcionální styl programování.

Interpretované a kompilované jazyky

Interpretované jazyky jsou překládány až za běhu programu. Jsou pomalejší, ale nemají tak velké formální požadavky (není potřeba inicializovat proměnnou, její datový typ se může za běhu měnit, ukazatele jsou zbytečné). Překládají se interpretrem, ten instrukce zároveň při překladu provádí. Hlavní nevýhodou těchto jazyků je, že se musejí vždy spouštět v interpretru. Do této skupiny patří většina verzí Basicu, všechny skriptovací jazyky (PHP, Python).

Kompilované jazyky jsou celé přeloženy a až potom mohou být spuštěny. Jsou rychlejší, mají vyšší nároky na formální správnost kódu. Překládají se kompilátorem, výsledkem překladu je (většinou) exe soubor. Patří sem většina klasických programovacích jazyků.

Teoreticky může mít jeden programovací jazyk verzi interpretovanou i kompilovanou.

Existuje ještě celá řada způsobů, jak dělit programovací jazyky. Např. u imperativních jazyků se mnohdy rozlišují ty, které podporují objektově orientované programování (OOP) nebo programování strukturované.

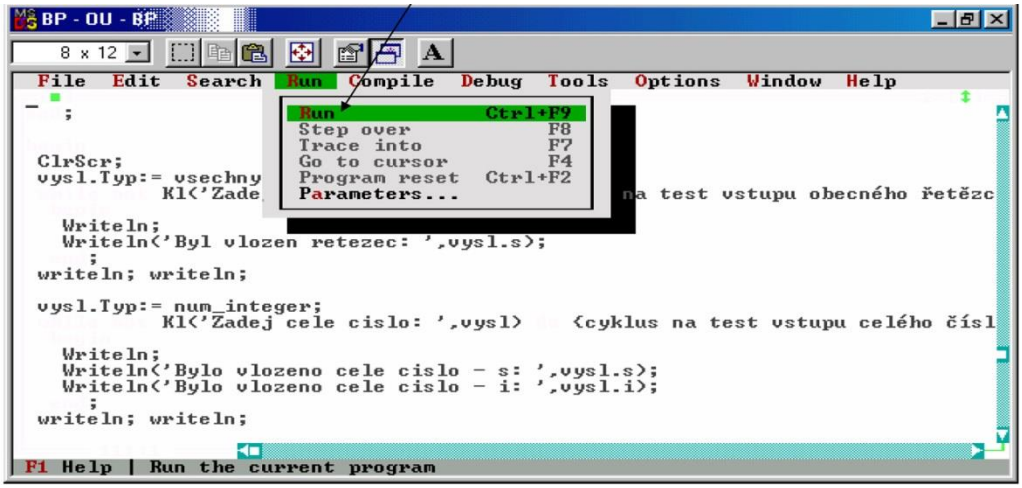
Dělení podle účelu. Zde je velmi obtížné přesně vymezit jednotlivé skupiny. Mnohé jazyky mají velmi široké použití, mnohdy jsou rozšířeny v mírných úpravách na většině operačních systémů. Avšak dosud nebyl vynalezen jazyk, který by byl vhodný úplně na všechno. A asi ani nikdy vynalezen nebude. Například velmi rozšířený jazyk C je vhodný hlavně na programování operačních systémů a dalších „nízkých“ aplikací. Assembler se používá k přístupu k hardwaru. PHP se dá použít pouze u internetových stránek. Atd. atd.

5.9.1. Pascal

Pascal je asi nejlepší strukturovaný výukový jazyk pro Dos. Vymyslel jej v roce 1970 Švýcar N. Wirth. Lze se setkal s dvěma verzemi – Borland Pascalem a Turbo Pascalem.

Verzí Pascalu je samozřejmě víc. Známý je Object Pascal – jeho objektově orientovaná verze. Delphi je Object Pascal doplněný ohromným množstvím standardních knihoven pro programování pod Windows.

V dnešní době začíná Pascal (a celkově strukturované programování) poněkud zastarávat a objevují se snahy nahradit jej na pozici učebního jazyka moderním objektově orientovaným Pythonem. Z původního zaměření Pascalu jako výukového jazyka vyplývá, že není zrovna nejvhodnější pro hodně velké projekty. Pascal má obrovské možnosti v takzvaných strukturovaných datových typech (složené datové typy, které spojují několik jednoduchých). Má velké formální požadavky – programátor musí definovat téměř úplně vše, a to v přesně daném pořadí. Z Pascalu vychází i velmi dobrý jazyk pro operační systém Windows Delphi. [38]



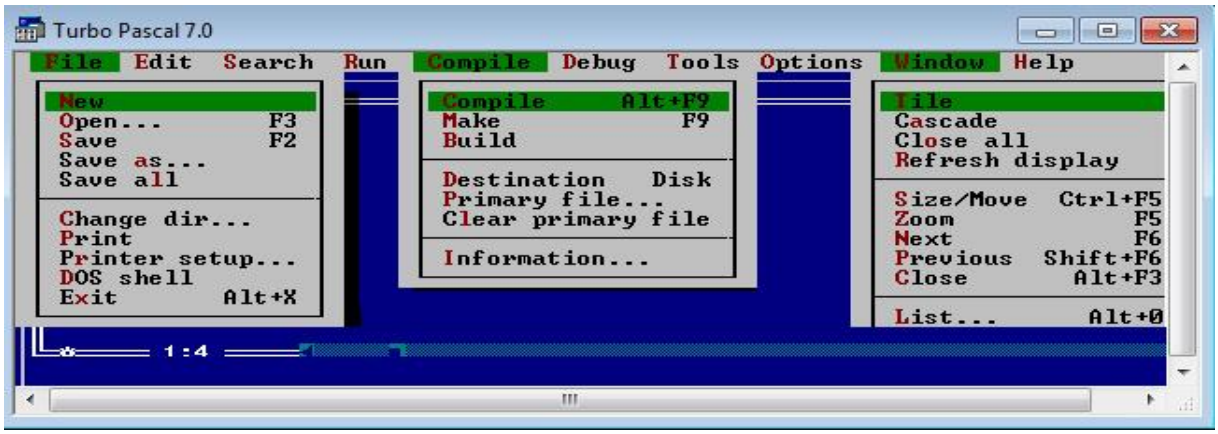
Obrázek 44 Ukázka zápisu Borland Pascalu programovacím jazyku Pascal [39]

```
Program Program1;
Var I : Integer;

Begin
        writeln('Vitam te u sebe.');

        For I := 1 to 7 do
                Begin

                        write(I);
                        writeln(' Bedy te vita.');
                End;
end.
```

Text 6 Ukázka zápisu v programovacím jazyku Pascal [1]

5.9.2. Turbo Pascal

Turbo Pascal (Borland Pascal) je nejrozšířenější implementací programovacího jazyka Pascal. Na rozdíl od prvních implementací původního Pascalu, navrženého v roce 1971 profesorem N. Wirthem především pro výuku programování je Turbo Pascal značně výkonnější prostředek.



Obrázek 45 Turbo Pascal 7.0 [1]

Vývojové prostředí Turbo Pascalu integruje textový editor (pro pořizování a úpravy zdrojových textů), kompilační překladač, linker (prostředek pro sestavení cílového kódu produktů překladu jednotlivých modulů programu) a debugger (ladicí prostředek).

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

www

http://pascal.webz.cz/index.php?part=down
http://mx-3.cz/dkx/www/progpasc.htm
http://fyzika.fce.vutbr.cz/file/schauer/automatizace/TurboPascal.pdf

5.9.3. C

Programovací jazyk C je známý svou přenositelností a rychlostí. Byl navržen jako poměrně malý jazyk, kombinující efektivitu a výkonnost. Byl napsán pro operační systém UNIX. Postupem času vznikla ANSI (American National Standard Institute) norma jazyka C. Shodný standard definuje ISO (International Stnadards Organisation) norma.

```
#include <stdio.h>

void main()
{
        printf("Vitam te u sebe.\n");

        for(int i=1; i<6; i++)
        {
                printf("%d Bedy te vita.\n", i);
        }
}
```

Text 7 Ukázka zápisu v programovacím jazyku C [1]

Proto se někdy uvádí spojení ANSI/ISO norma. Pokud budete ANSI normu dodržovat, máte velkou šanci, že budete moci svůj program přenášet na různé platformy. Tento jazyk se používá většinou pro psaní systémového SW, ale dá se samozřejmě použít i v aplikacích. Tento jazyk je oblíbený i v dnešní době a někteří dávají přednost C před C++.

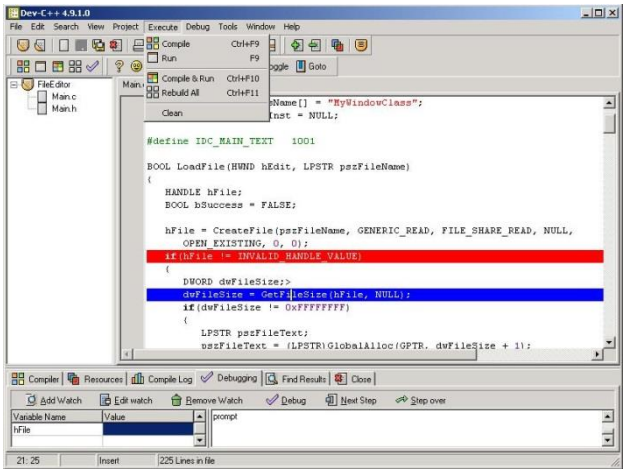
5.9.4. C++

Jazyk C++ je pokračovatelem jazyka C (jehož vývoj se již téměř zastavil). Nejdříve byl označován jako C with Classes, od roku 1983 C++.

Tato moderní inkarnace jazyka C v sobě spojuje sílu tradičního „céčka“ a navíc přidává vysokoúrovňové vlastnosti vyšších jazyků. Hlavním rozdílem mezi C a C++ je způsob programování.

C je ryze procedurální jazyk, to znamená, že zdůrazňuje algoritmy (pozn. program jako takový se skládá ze dvou částí: dat a algoritmů. Data jsou informace, které program používá, zpracovává nebo vytváří pomocí algoritmů). Naproti tomu C++ je objektově orientovaný jazyk, to znamená, že zdůrazňuje data. Myšlenka OOP spočívá v návrhu datových tříd, které popisují určitý souhrn vlastností a jak s těmito vlastnostmi zacházet.

Obecně se má, zato, že C/C++ jsou jazyky složité. Je to pravda. Není to ani tak kvůli syntaxi jazyka, ta je naopak relativně jednoduchá. Je to spíše tím, že tyto jazyky umožňují snad všechno, na co si vzpomenete (to však bývá i kamenem úrazu). Příkladem může být práce s adresami. Je k nim k dispozici obrovské množství knihoven a není v lidské moci ovládat celou stránku tohoto jazyka. [37]



Obrázek 46 Integrované vývojové prostředí Dev-C + + pro programovací jazyk C / C + + [40]

```
#include <iostream>
using namespace std;

int main ()
{
        cout << "Vitam te u sebe.\n";
        for(int i=1; i<6; i++)
        {
                cout << i << " Bedy te vita.\n";
        }

 return 0;
}
```

Text 8 Ukázka zápisu v programovacím jazyku C++ [1]

5.9.5. Basic

BASIC je skupina programovacích jazyků. Byl značně rozšířen. Jeho aplikace byla implantována hlavně na domácích osmibytových mikropočítačích. BASIC je zkratka z anglických slov Beginner's All-purpose Symbolic Instruction Code. V českém volném překladu to znamená univerzální symbolické kódové instrukce pro začátečníky.

Jeho typickou vlastností bylo číslování řádků. Instrukce se prováděly od řádku s nejmenším číslem až po řádek s největším číslem. Byla možná takzvaná adresace řádku.

Basic byl zaveden jako jednoduchý nástroj pro výuku programování. K jednoduchosti přispívalo i to, že klíčová slova jazyka vychází z běžné angličtiny. V Československu s ním byly vybaveny počítače IQ151, PMD85.

```
01 CLS
02 PRINT "Vitam te u sebe"
03 PRINT "Stisknete cokoliv pro pokracovani"
05 PAUSE
06 CLS
10 FOR a=1 TO i-
15 PRINT "Bedy te vita.",i
20 PLOT 1+(p(2,a)-xmin)*rx, 1+(p(1,a)-ymin)*ry
21 DRAW rx*(p(2,a+1)-p(2,a)),ry*(p(1,a+1)-p(1,a))
30 NEXT a
```

Text 9 Ukázka zápisu v Basicu

Obrázek 47 Osmibitové mikropočítače s programovacím jazykem Basic

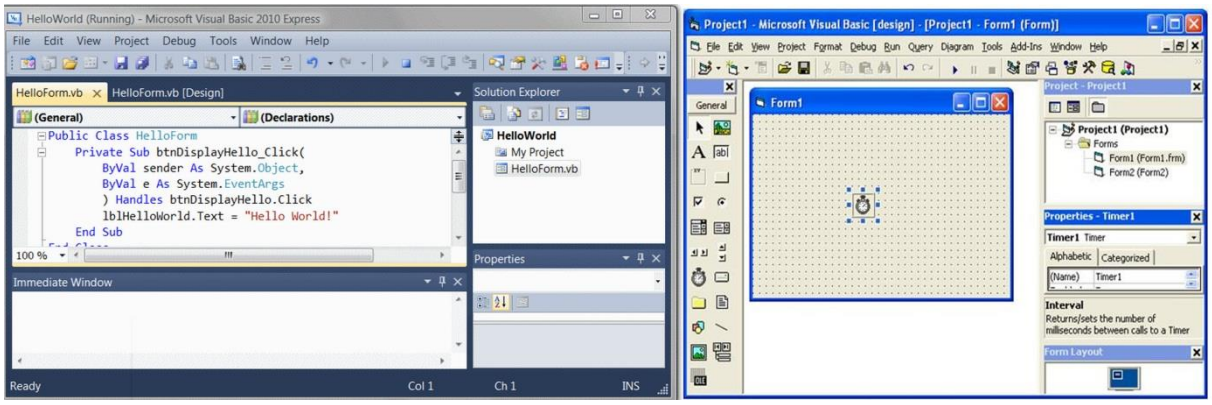
5.9.6. Visual Basic

Visual Basic prošel mnoha verzemi, z nichž poslední byla Visual Basic 6.0 vydaná v roce 1998. Jeho vývoj se zastavil a byl dán uživatelům Visual Basic. Tato verze už byla koncipována právě pro tvorbu programů v prostředí .NET Framework.

.NET Framework je prostředí spustitelné v operačním systému Windows nutné pro běh programů napsaných v jakémkoli jazyce pro .NET. Většina uživatelů má .NET Framework už nainstalován. Aplikace a programy, které jsou v tomto prostředí naprogramovány, pro svůj běh potřebují právě toto pomocné prostředí (Framework).

Microsoft NET Framework vytváří prostředí, ve které aplikace běží, jedná se o jistou obdobu v případě prostředí pro jazyk Java-JRE – Java Runtime Environment. NET Framework dále poskytuje pro v něm napsané aplikace základní třídy a knihovny.

Od firmy Microsoft pochází Visual Basic 2008 Express Edition, který je ke stažení zadarmo.



Obrázek 48 Prostředí Microsoft Visual Basic [1]

```
Module Module1
    Sub Main()
        Console.WriteLine("Vitam te u sebe.")
        For i As Integer = 1 to 5
            Console.WriteLine("{0} Bedy te vita.",i)
        Next
    End Sub
End Module
```

Text 10 Ukázka zápisu ve Visual Basicu [1]

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

www

http://www.gvp.cz/local/new/ucebnice/VisBas/visbas.htm
http://www.visualbasic.freegate.cz/odkazy.php
http://www.devbook.cz/c-sharp-zaklady-navody-tutorialy-zdrojove-kody-ke-stazeni

5.9.7. C # (C SHARP)

C# se vyslovuje anglicky jako C Sharp. Jde o vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft zároveň s platformou NET Framework.

C# je vlastně vylepšená a zjednodušená objektová verze programovacího jazyka C++. Nezbytnou podmínkou pro programování v jazyce C# je prostředí NET Framework. Toto prostředí se skládá se z několika částí.

Common Language Runtime (CLR) – jedná se o společné běhové prostředí. Toto prostředí zajišťuje běh programů přeložených z různých programovacích jazyků do mezijazyka Microsoft Intermediate Language (MSIL). CLR umožňuje jejich vzájemnou spolupráci, takže různé součásti programu mohou být napsány v různých programovacích jazycích.

Basic Class Library (BLC) je knihovna tříd. Třídy slouží pro ukládání různých druhů dat. Nad touto knihovnou jsou ještě knihovny pro tvorbu grafického uživatelského rozhraní, programů a knihovny pro webové služby. Nejvyšší vrstvu tvoří překladače různých programovacích jazyků.

```
using System;

class Program
{
   static void Main()
   {
                Console.WriteLine("Vitam te u sebe.");

                for(int i=1; i<6; i++)
                {
                        Console.WriteLine("{0} Bedy te vita.",i);
                }
   }
}
```

Text 11 Ukázka zápisu v C # (C SHARP) [1]

5.9.8. Perl

Perl je interpretovaný programovací jazyk. Nemusíme zde kompilovat samostatně. Program je zkompilován po každém spuštění automaticky a můžeme kdykoli „přikompilovat“ další kód.

```
#!/usr/bin/perl

print "Vitam te u sebe.";

for ($i = 1; $i < 6; $i++)
{
        print "$i Bedy te vita.";
}
```

Text 12 Ukázka zápisu v Perlu [1]

5.9.9. SmallTalk

Smalltalk je čistý objektově orientovaný jazyk (podobně jako Java, C#) a použitelný v podobném okruhu aplikací jako Java (sítě, web, multimédia, vestavěné systémy). Na rozdíl od Javy však jde o jednoduchý dynamický jazyk (co do jednoduchosti a dynamičnosti srovnatelný s LISPem a Prologem) použitelný i pro rychlé prototypování a pro aplikace z oblasti umělé inteligence (mj. protože s programy lze manipulovat jako s daty a mohou za běhu samy sebe modifikovat).

Jelikož Smalltalk uplatňuje objektovou orientaci naprosto důsledně a bez výjimek (na rozdíl od Javy nezná pojem „primitivní typ“), je extrémně jednoduchý.

Jazyk a vývojové prostředí tvoří konzistentní celek, zastřešený grafickým uživatelským rozhraním.

Smalltalk není jen jazyk (na rozdíl od Javy, C#). Jde o systém s vlastnostmi běžně očekávanými od operačního systému. Jazyk je jeho součástí a v tomto jazyce je celý systém vytvořen. Díky tomu je schopen sám sebe svými vlastními prostředky vyvíjet (inkrementálně, za běhu, bez restartu). Je zcela otevřený a přístupný jakýmkoliv modifikacím a rozšířením, a to i ve svých nejhlubších vrstvách (plánovač procesů, kompilátor). Celý systém běží na virtuálním stroji, a proto se chová bitově/pixelově totožně na všech hardwarových a softwarových platformách a ke své činnosti ani nutně nepotřebuje žádný operační systém (virtuální stroj nijak nezávisí na souborovém systému a může běžet přímo na holém hardwaru). [46]

```
'Vitam te u sebe.' displayNl
(1 to: 5) do: [ :item | ('%1 Bedy te vita.' % { item }) displayNl ]
```

Text 13 Ukázka zápisu v Smalltalku

5.9.10. PHP

PHP je scriptovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací (formáty WML, HTML, XHTML).

PHP lze použít i k tvorbě konzolových a desktopových aplikací. Pro desktopové použití existuje kompilovaná forma jazyka.

Při použití PHP pro dynamické stránky jsou skripty prováděny na straně serveru – k uživateli je přenášen až výsledek jejich činnosti. Interpret PHP skriptu je možné volat pomocí příkazového řádku, dotazovacích metod HTTP nebo pomocí webových služeb. Syntaxe jazyka je inspirována několika programovacími jazyky (Perl, C, Pascal a Java).

PHP je nezávislý na platformě, rozdíly v různých operačních systémech se omezují na několik systémově závislých funkcí a skripty lze většinou mezi operačními systémy přenášet bez jakýchkoli úprav. PHP je nejrozšířenějším skriptovacím jazykem pro web. Oblíbeným se stal především díky jednoduchosti použití a bohaté zásobě funkcí. V PHP jsou napsány i ty největší internetové projekty, včetně Wikipedie. [46]

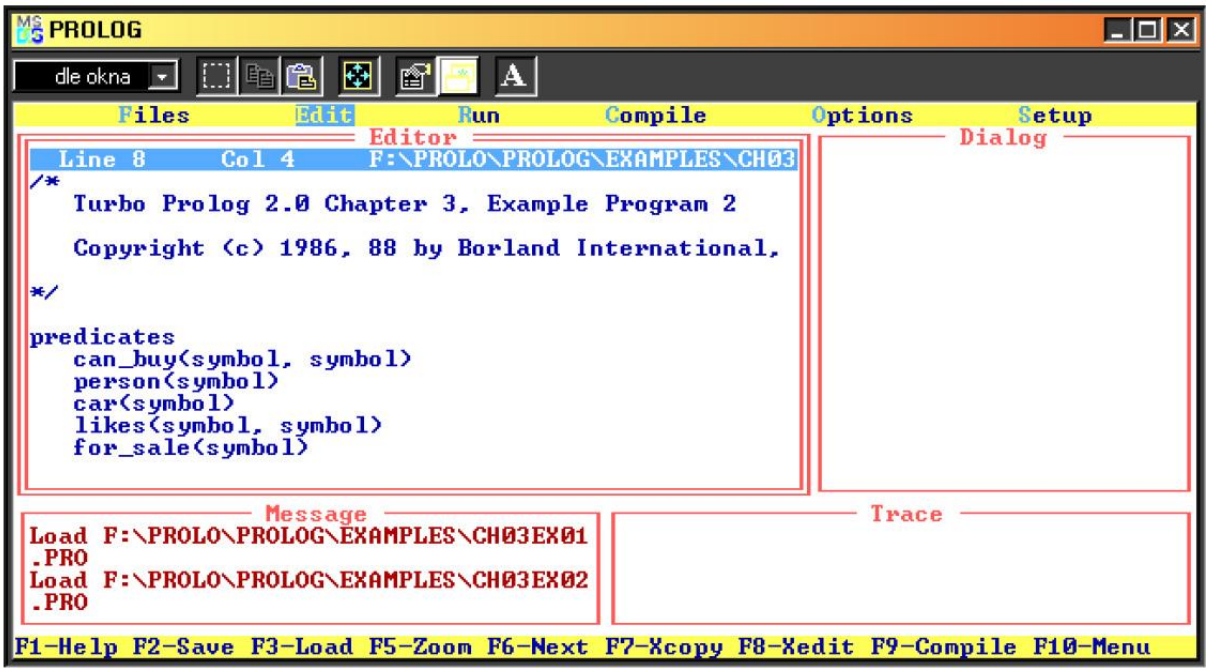
```
<?php

print("Vitam te u sebe");

for($i = 1; $i < 6; $i++)
{
        print(i + " Bedy te vita.");
}

?>
```

Text 14 Ukázka zápisu v PHP

5.9.11. Prolog

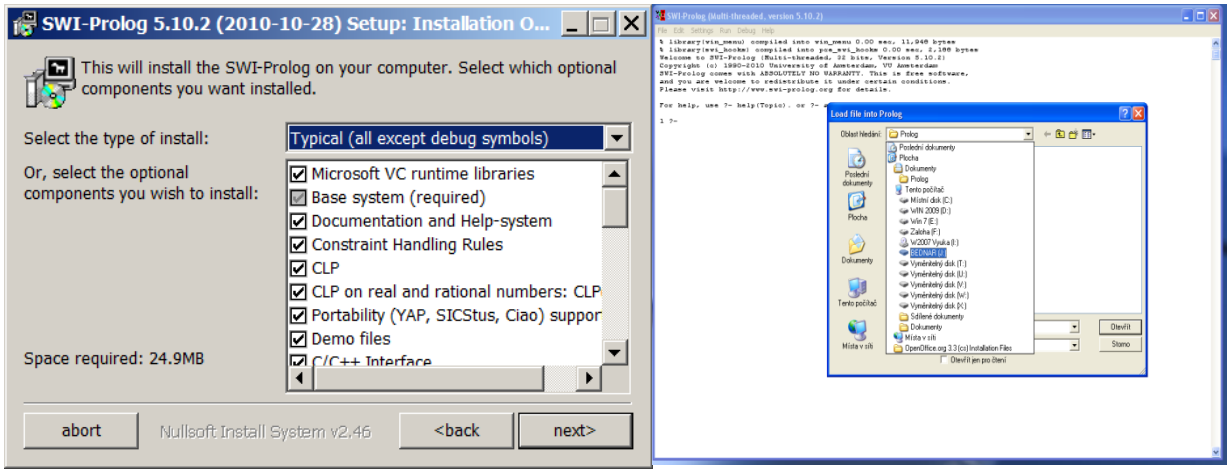
Prolog je jazyk pro programování symbolických výpočtů. Jeho název je odvozený ze slov Programming in Logic a vychází z principů matematické logiky. Jeho úspěch byl podnětem pro vznik nové disciplíny matematické informatiky – logického programování, což je perspektivní styl programování na vyšší abstraktní úrovni. Prolog je také strojovým jazykem nejmodernějších počítačů. Má doposud specifické oblasti použití jako je umělá inteligence, znalostní inženýrství. [48]

Prolog je založen především na matematické logice



Text 15 Ukázka zápisu v Turbo Prologu [49]

Prolog se výborně hodí pro programování logických vazeb a s tím spojených symbolických výpočtů.



Obrázek 49 Ukázka SWI-Prologu pod Windows [1]

Základním stavebním kamenem programů v Prologu je výroková logika (logiku máme výrokovou a predikátovou).

```
uvod(Sez):- Sez=[1,2,3,4,5,6,7,8,9],tabulka(Sez).    % nacteni seznamu cisel do Sez, spusti funkci tabulka a vrati Sez do S

vymen(X,Co,[X|T],[Co|T]).                    % Vymeni zadane cislo (X) v seznamu(T)
vymen(X,Co,[H|T],[H|T1]):-vymen(X,Co,T,T1).  % za znak hrace (Co) - kolecko nebo krizek

vytkni(X,[X|T],T).                           % vezme cast seznamu a vrati jeji cast
vytkni(X,[H|T],[H|T1]):-vytkni(X,T,T1).

spoj([],Sez,Sez).                            % spoji dve rozdelene casti hraci plochy
spoj([H|T],S,[H|W]):-spoj(T,S,W).

tabulka([A1,A2,A3,B1,B2,B3,C1,C2,C3]):-      % fuknce tabulka prijma 9 cisel v poli
```

Text 16 Ukázka zápisu v Prologu

Výroková logika používaná v prologu je značně jednoduchá. Formuluje věty pomocí výroků a logických spojek (spojuje složitější formule např. do tvaru implikace – podmínky). Toto dává Prologu schopnost řešit velmi malou skupinu problémů, jeho pravé využití mu dává obohacení o prvky predikátové logiky.

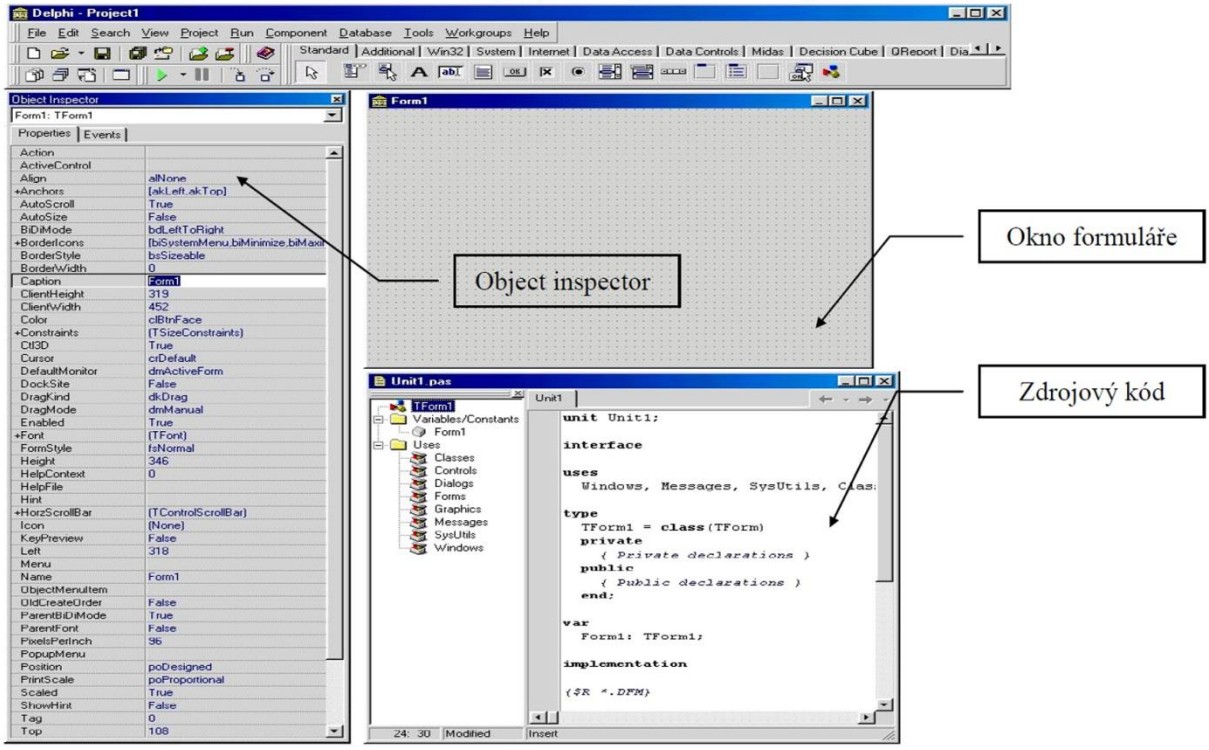
Predikátová logika používá také predikáty, funktory a proměnné (umožňuje formulovat vztahy a vlastnosti objektů pomocí relací). Důležitou charakteristikou predikátové logiky jsou kvantifikátory, které je nutno při logickém programování obětovat.

Všechny proměnné jsou chápány jako univerzální. Predikátová logika dává schopnost pracovat nejen s elementárními výroky, ale také rozlišit objekty a jejich vztahy.

5.9.12. Delphi

Delphi je pojmenování prostředí, které umožňuje navrhovat a vytvářet programy v jazyce Objektový Pascal. Objektový Pascal se v průběhu času (v polovině osmdesátých let) vyvinul z jazyka Pascal. Programátorské prostředí Delphi je IDE (Integrated Development Environment). To již podle názvu znamená, že toto prostředí v sobě integruje všechny prvky, které programátor potřebuje pro návrh a vývoj programů, pro jejich kompilaci (překlad), testování a ladění. Prostředí je založeno na vizuálním principu. Všechno, co bude v běžícím programu vizuálně zobrazeno, programátor během návrhu programu vše vizuálně skládá z předpřipravených částí (komponent). [41]

Standardní okna vývojového prostředí Delphi



Obrázek 50 Vývojové prostředí Delphi [42]

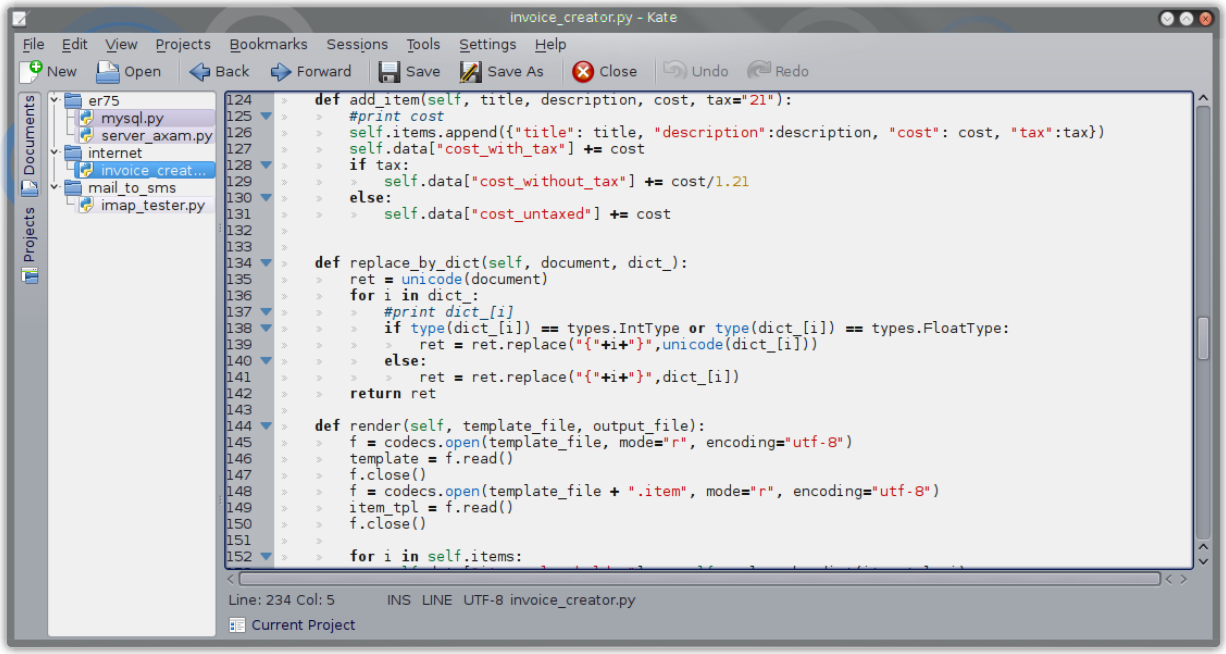
INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

http://k-prog.wz.cz/pascal/delphi.php
http://info.spsnome.cz/Delphi/Delphi

5.9.13. Python

Python je dynamický, objektově-orientovaný programovací jazyk. Naučit se dá zhruba za měsíc. Jde o dynamický interpretovaný jazyk. Někdy bývá zařazován mezi takzvané skriptovací jazyky. Jeho možnosti jsou ale větší. Python byl navržen tak, aby umožňoval tvorbu rozsáhlých, plnohodnotných aplikací (včetně grafického uživatelského rozhraní).

Python je hybridní jazyk (nebo také víceparadigmatický), to znamená, že umožňuje při psaní programů používat nejen objektově orientované paradigma, ale i procedurální a v omezené míře i funkcionální, podle toho, komu co vyhovuje nebo se pro danou úlohu hodí nejlépe. Python má díky tomu vynikající vyjadřovací schopnosti. Kód programu je ve srovnání s jinými jazyky krátký a dobře čitelný.



Obrázek 51¨Programátorský editor Kate se zvýrazněním syntaxe a zápisu programu v jazyku Python [1]

K význačným vlastnostem jazyka Python patří jeho jednoduchost z hlediska učení. Bývá dokonce považován za jeden z nejvhodnějších programovacích jazyků pro začátečníky. Tato skutečnost je dána tím, že jedním z jeho silných inspiračních zdrojů byl programovací jazyk ABC, který byl jako jazyk pro výuku a pro použití začátečníky přímo vytvořen. Python ale současně boural zažitou představu, že jazyk vhodný pro výuku není vhodný pro praxi a naopak. Podstatnou měrou k tomu přispívá čistota a jednoduchost syntaxe, na kterou se při vývoji jazyka hodně dbá.

Význačnou vlastností jazyka Python je produktivnost z hlediska rychlosti psaní programů. Týká se to jak nejjednodušších programů, tak aplikací velmi rozsáhlých.

U jednoduchých programů se tato vlastnost projevuje především stručností zápisu. S vysokou produktivností souvisí dostupnost a snadná použitelnost široké škály knihovních modulů, umožňujících snadné řešení úloh z řady oblastí.

Python se snadno vkládá do jiných aplikací (embedding), kde pak slouží jako jejich skriptovací jazyk. Tím lze aplikacím psaným v kompilovaných programovacích jazycích dodávat chybějící pružnost.

Jiné aplikace nebo aplikační knihovny mohou naopak implementovat rozhraní, které umožní jejich použití v roli pythonovského modulu. Jinými slovy, pythonovský program je může využívat jako modul dostupný přímo z jazyka Python, tj. extending. [34]

Jedna z největších výhod Pythonu je podpora velkého množství datových typů. Podporuje seznamy, asociativní pole, posloupnosti a spoustu dalších typů. Díky tomu, že je interpret, dokáže pracovat s proměnnými dynamicky – není problém změnit datový typ proměnné, přidat nebo odebrat prvek ze seznamu. Takto se dají plně nahradit dynamické datové struktury. Python také může využívat knihovny napsané pro jazyk C nebo C++. Pro samotný Python bylo napsáno mnoho knihoven (jsou standardně dodávané v instalačním balíku). Umožňují práci skoro se vším: od zpracování HTML, přes práci se zvukem a několika grafickými rozhraními, po komunikaci přes protokol Telnet. [36]

```
#!/usr/bin/python

print "Vitam te u sebe."

for i in range(4):
        print i+1, "Bedy te vita."
```

Text 17 Ukázka zápisu v Pythonu

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

www

http://www.python.org/doc/
http://www.py.cz/FrontPage

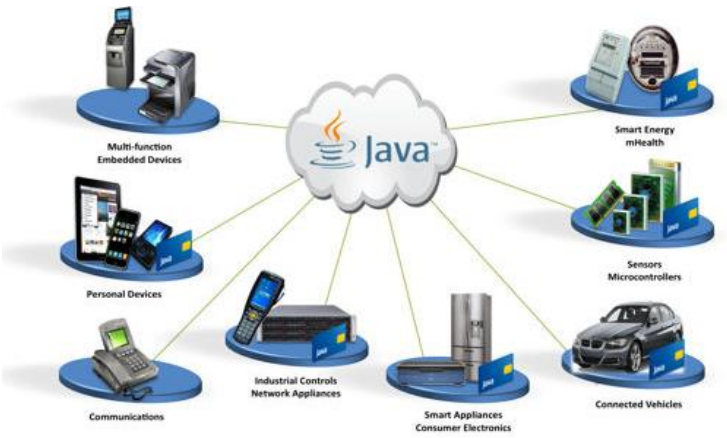
5.9.14. Java

Java se čte jako „džava“. Jde programovací jazyk pocházející od firmy Sun Microsystems, později koupené firmou Oracle. Jedná se o objektově orientovaný jazyk vycházející z C++, ke kterému má také syntakticky nejblíž - nepočítáme-li C# [čti: sí šárp], který vznikl až po příchodu Javy. Oproti svému předchůdci Java neobsahuje některé konstrukce, které způsobovaly při programování největší potíže, a navíc přidává mnoho užitečných vlastnosti. Přidělování a uvolňování paměti je zde obstaráno automaticky (pomocí *garbage collectoru*). Objekt se neruší pomoci delete nebo free(), ale pouze se "nabídne" ke zrušení (např. přiřazením neplatné reference null).

Klasický problém z C/C++, ukazatele, je zde zcela odstraněn, neboť ty zde prostě nejsou (resp. jsou nahrazeny referencemi). Dereferencovaní samozřejmě provádí runtime. Programátor je zde ušetřen notorické chyby zápisu pointerem mimo datovou oblast.

Je implementován mechanismus vláken (threads) a lze tudíž spouštět více úloh v rámci jednoho programu. Bylo samozřejmě pamatováno i na jejich synchronizaci pomoci tzv. monitorů. Vytvořené objekty lze automaticky serializovat, tj. ukládat do souboru, zasílat po síti apod. Lze provádět reflexi, neboli zjišťování informací o objektu (jaké má proměnné, metody atd.), Je implementován mechanismus výjimek, takže veškeré runtime chyby je možné odchytit a zpracovat. Výjimky jsou samozřejmě objektové, takže lze zachytit i celou hierarchii výjimek v jednom hlídaném bloku.

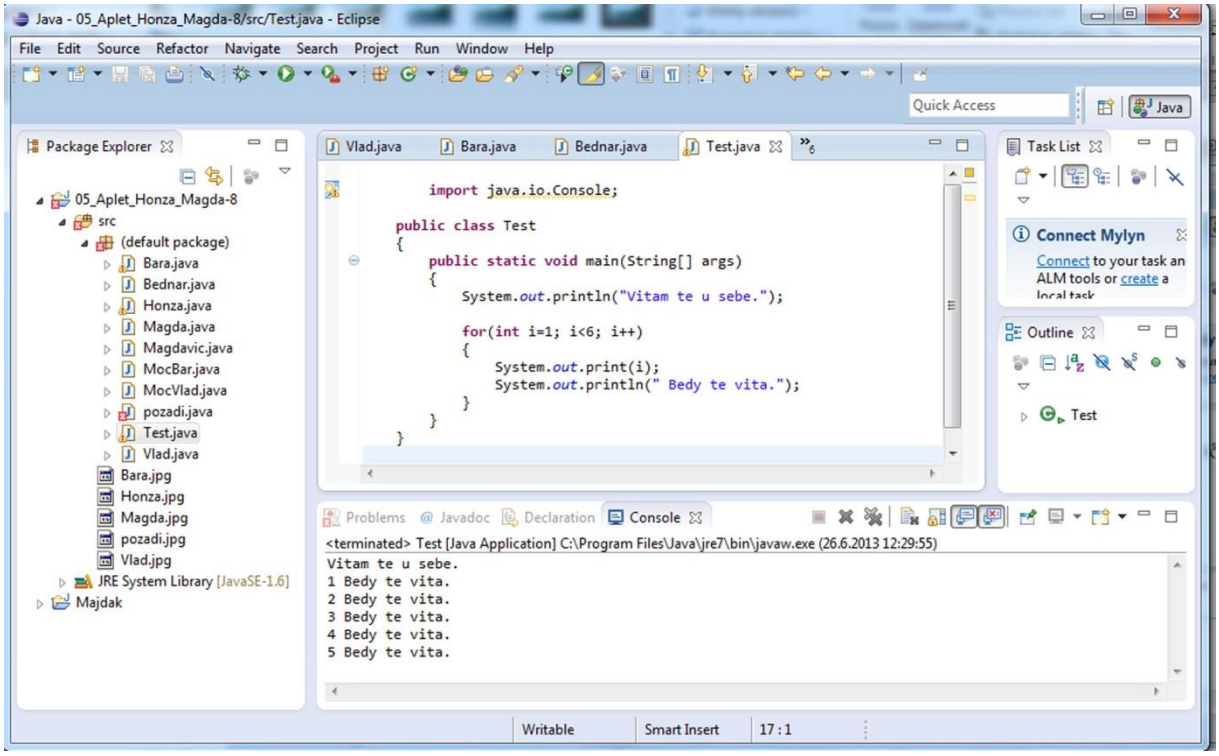
Velkou výhodou Javy je také její hardwarová nezávislost, neboť je překládaná do speciálního mezikódu (bytecode), který je na konkrétním počítači nebo zařízení (PC, handheld, mobilní telefon apod.) interpretován, příp. za běhu překládán do nativního kódu. Programátor tedy může napsat javovský program například na PC pod Windows a spustit jej na PC s Linuxem, na Macu, zkrátka všude, kde je k dispozici Java runtime. [44]



Obrázek 52 Možnosti použití Javových aplikací [44]

Java podporuje tvorbu dynamicky rozšiřitelných aplikaci (plug-inů apod.). Java klade značný důraz na bezpečnost.

Značným ulehčením pro programátory je obsáhlost standardně dodávaných knihoven, se kterou se nemůže srovnávat asi žádný běžně používaný jazyk.



Obrázek 53 Vývojové prostředí Eclipse a program napsaný v Jávě [1]

K dispozici jsou knihovny pro tvorbu grafického uživatelského rozhraní (GUI), vstup/výstup, práci s textem, komunikaci s SQL databázemi, práci s komprimovanými soubory a mnoho dalších.

```
import java.io.Console;

public class Test
{
	public static void main(String[] args)
	{
		System.out.println("Vitam te u sebe.");

		for(int i=1; i<6; i++)
		{
			System.out.print(i);
			System.out.println(" Bedy te vita.");
		}
	}
}
```

Text 18 Ukázka zápisu v Javě [1]

5.9.15. Java Script

JavaScript je internetový jazyk, doplňující HTML kód o některé zajímavé prvky. Běží na klientské straně - prohlížeč si stáhne zdrojový kód (který se vepisuje přímo do HTML kódu) a teprve potom ho spustí (pokud to umí). Všechny moderní internetové prohlížeče sice JavaScript podporují ale v dosti rozdílné míře - každý si do něj něco přidává, nebo z něj něco neuznává. Proto vypisuji v Seznamu vlastností a metod i prohlížeče podporující daný prvek.

Příčinou vzniku JavaScriptu byl požadavek na zvýšení uživatelského komfortu pro uživatele stránek. JavaScript můžeme použít například při vstupní kontrole dat vkládaných do formulářů ještě předtím, než jsou vyplněné údaje odeslány na server. Kontrolu údajů nemusí provádět server a výsledkem je rychlejší odezva pro uživatele. JavaScript ve svých prvních verzích navíc umožnil i vytváření interaktivnějšího uživatelského rozhraní stránek. Stránka mohla reagovat na různé události vyvolané uživatelem a například doplňovat pole do formulářů nebo vyvolat otevření nového okna prohlížeče. Tyto možnosti pak dále rozšířil nový koncept dynamického HTML. [55]

Javascript vyvinuly firmy Netscape a Sun Microsystems. Je syntaxí podobný Javě, de facto z ní vychází. Java je však komplexnější, těžší a může běžet sama (JavaScript jen jako součást HTML kódu). Je case-senzitivní, tzn. že (v tomto případě dost striktně) rozlišuje malá a velká písmena (výrazy "alert" a "Alert" nejsou v žádném případě to samé!).

Celkově nemá mnoho možností, jen zpestřuje a mnohdy znepřehledňuje stránky. Nemůže ovlivňovat jiné soubory (s vyjímkou tzv. cookies). Spouští ho prohlížeč, pokud ho podporuje a má to od uživatele povoleno. Existují různé verze. Jsou zde problémy s kompatibilitou. Verze pro IE se jmenuje JScript.

JavaScript by se měl požívat s mírou, přeplácaná stránka nevypadají dobře. Ovšem JavaScriptem lze udělat i spousta zajímavých věcí: rolování dokumentu, vypsání aktuálního data a času (pokud to má uživatel správně nastavené), ovlivnění stavové řádky (to šedé dole).

JavaScript je, podobně jako mnoho jiných jazyků, jazyk objektový. Objekt se dá (velmi zjednodušeně) představit jako souhrn instrukcí jazyka, které mají vzájemnou logickou spojitost. Např. objekt Window obsahuje instrukce týkající se okna prohlížeče, objekt Date zase aktuálního času a data, atd. Jednotlivé objekty mohou obsahovat objekty další, metody nebo vlastnosti. Rozdíl mezi metodou a vlastností je v tom, že metoda má za sebou závorky (které někdy obsahují bližší instrukce) a funguje vlastně jako (předdefinovaná) funkce, vlastnost obsahuje jen hodnotu (která se může i měnit). [45]

ČAS POTŘEBNÝKE STUDIU 30 minut.

CÍL: Pochopit rozdíl mezi simulací a modelováním

6. SIMULACE

Při simulacích nahrazujeme skutečný dynamický systém (robota) modelem. Pomocí experimentů s modelem se snažíme získat informace, jak by vlastně skutečný dynamický systém (robot) fungoval. Jde v podstatě o to, že pokud bychom neměli fyzicky sestaveného robotka, ale měli již pro něj napsaný program, tak bychom v určitém simulačním prostředí tento program již mohli spustit a simulační program by předváděl a ukazoval jednotlivé kroky podle námi vloženého programu. [51]

Je jedna všeobecně rozšířená definice, co je to vlastně simulace, a to podle Dahla, která říká: ***„Simulace je výzkumná technika, jejíž podstatou je náhrada zkoumaného dynamického systému jeho simulátorem s tím, že se simulátorem se experimentuje s cílem získat informace o původním zkoumaném dynamickém systému.“ [51]***

Čas v reálném životě a v simulátorech hraje obrovskou roli. Podle toho, zda námi zkoumaný objekt zanedbává význam času, se dělí simulační systémy na dynamické a statické.

Statické systémy (již podle názvu) jsou systémy, kde proces není zpětně vázám na ubíhající čas. V tomto systému je plynutí času zanedbatelné a nemusíme s ním nijak počítat (systémy, ve kterých zanedbáváme roli času, se nazývají statické). Není tedy podstatné pro návaznost systému v jakém čase a v jakých časových sledech události v systému probíhají, ať se jedná o proces zapsaný nebo probíhající v reálném životě.

Náš robot může fungovat jako systém statický v tom případě, že bude jen třeba svými čidly něco počítat. Pokud by ale náš robot začal nějak reagovat na podněty, ze svých čidel (roztočil by kolečko, rozpohyboval rameno, či vyslal rádiový signál), začíná se jednat již o systém reagující dynamicky a proto dynamický systém.

Základní fáze simulace.

1. Vymezení objektu poznání.
2. Vymezení zkoumaného systému.
3. Vytvoření aktuální představy o systému.
4. Realizace modelu matem (počítačového).
5. Ověřování správnosti modelu testování.
6. Další použití modelu – aplikace modelů.

6.1. Typy simulací

Rozeznáváme tři typy simulací (spojitou, diskrétní a kombinovanou).

Budeme-li se na určitý jev dívat jako na kontinuální (plynulou) akci, pak použijeme simulaci spojitou. Díváme-li se na ten tentýž problém jako na množinu akcí, které probíhají nespojitě (diskrétně) v čase, pak hovoříme o simulaci diskrétní.

Hlavní podstatou simulace je ověřit a případně objevit chyby, či kolizní situace. Dále vytvořit představy, jak se dotyčná aplikace, nebo proces bude chovat v reálném nasazení (pokud ještě není součástí reality). V našich případech se jedná o dynamické simulační systémy. Máme vlastně 3 skupiny simulací co do průběhu času. [51]

A. Běh reálného času je roven simulovanému

Čas v dané simulaci běží stejným tempem (rovnocenně se skutečnou událostí). Použití je v odlaďování konkrétních aplikacích. Může se jednat a jejich zdokonalování či vývoj (nějakého výrobního procesu). Můžeme simulovat chování našeho robota, i když nebudeme mít ještě jeho fyzické provedení. [51]

B. Běh reálného času je mnohem pomalejší, než čas simulovaný

Čas v dané simulaci běží mnohem rychleji, než jak by daná událost stávala v reálném čase. Simulace tedy proběhne několikanásobně rychleji než v reálné situaci. Simulačním procesem, který trvá třeba jen pár minut nebo i den, dokážeme nasimulovat, jak se bude vyvíjet reálná aplikace během několika hodin, dnů či týdnů. Takovéto simulátory se používají k zjištění kolizních stavů, které se mohou projevit až po delší časové době. Další z mnoha použití je zjišťování namáhání a opotřebení (většinou mechanických dílů). Tyto simulace se taktéž používají pro většiny dlouhodobějších předpovědí (erupce na slunci, počasí na zemi, vzájemné ovlivňování částic, či kosmických těles). Použití tedy všude tam kde události v reálném čase jsou pomalé a my potřebujeme mít výsledky simulace (pro nás) v rozumné době. Modelový příklad je v možnosti ověřit dlouhodobé zatížení našeho robotka, kde počítáme události, které se projevily nebo projeví v rozpětí několika dnů, ale my se snažíme mít výsledky simulací v co nejkratší době, to znamená v rozmezí několika minut (simulovaný čas je mnohem rychlejší než reálný). [51]

C. Běh reálného času je mnohem rychlejší, než čas simulovaný

Toto se využívá k detailnímu studiu procesů (běžících aplikací) probíhajících značně rychle, jež bychom se svými receptory nebyly schopni většinou ani zaznamenávat, natož vyhodnocovat.

Simulace v těchto případech trvá několikanásobně déle než skutečný proces v realitě. Tedy simulovaný čas zde „plyne“ pomaleji než reálný. Typickým příkladem je chování a zkoumání vlivu vysokých rychlosti pohybů dílů našeho robota. Snažíme se zjistit možné kolizní situace, které mohou mít vazbu na mechanickou setrvačnost našeho robota.

OTÁZKY

1. Jak se systémy klasifikují a jaké jsou jejich důležité vlastnosti?
2. Co je to systém a model a jaký je jejich vztah k počítačům?
3. Co je to simulace a modelování v oblasti aplikace výpočetní techniky?
4. Co je třeba si zvláště uvědomit při simulaci a modelování?

6.2. Simulační programy

Všechny jazyky jako je Cobol, Fortran, Basic, Pascal, C, Lisp a podobné mají nutnost napsat skutečný program dle syntaxe jazyka a logiky modelu. Pro potřeby simulačních modelů proto byly vytvořeny speciální simulační jazyky se syntaxí vhodnou pro řešení daného problému jako je SIMSCRIPT, GPSS, SIMULA, MODSIM, ECSL, SIMULA, MOR/DS.

Dále byly upraveny programy s využitím textového a grafického rozhraní pro psaní simulačních programů, jejichž představitelé jsou Xcell+, SIMPROCESS, SIMUL8. V těchto programovacích jazycích používáme ikony a zástupné symboly, program je tvořen v pozadí automaticky často bez vědomí uživatele. Existuje celá řada programů pro specifickou simulaci, či s využitím speciálního softwarového vybavení. Vždy záleží na konkrétních potřebách a na dostupných finančních prostředcích. [51]

Při simulačním programování robotů musíme mít na paměti jejich kinetickou energii a setrvačnost Stejně tak si musíme uvědomit, že mechanický robot má mechanické omezení pohybu i rychlosti. Máme zde antropomorfní limity. Antropomorfní robot je stroj, jehož manipulátor (třeba simulace končetin) obsahuje rotační klouby podobné kloubům lidským.

Podobně robotická zařízení mají fyzická omezení takové, jako věci ve skutečném životním prostředí.

Obrázek 54 Robotické zařízení s klouby [51]

ČAS POTŘEBNÝKE STUDIU 60 minut.

CÍL: Pochopit co se skrývá pod pojmem umělá inteligence

7. UMĚLÁ INTELIGENCE

Pojem umělá inteligence je dnes hodně rozšířen a bohužel se ho používá i tam, kde se se jedná pouze o výběr dat z databází, kde není žádným způsobem zajištěno takzvané učení a rozhodování podle vlastních algoritmů, ale jen podle již dopředu pevně daného výběru.

Umělá inteligence (UI) vlastně spojuje mnoho oborů (minimálně informatiku, kybernetiku, psychologii, matematiku). Umělá inteligence se jako vědní disciplína za posledních 40 let neustále zdokonaluje a vyvíjí.

7.1. Inteligence a IQ

Výraz inteligence je vždy spjata s chováním a zpracováváním informací, které se dostávají k živým organismům. Inteligenci máme snahu již od nepaměti nějakým způsobem hodnotit. Většinou jde o inteligenci jako komplexní schopnost reagování na složité podněty z okolí a jejich vyhodnocování. Jde tedy o zpracování již dříve naučených a získaných informací či dovedností a jejich přizpůsobení na dotyčnou situaci, kterou se snažíme řešit. U živých tvorů je tato schopnost dána kombinací dědičnostních faktorů a současně schopností se učit, rozpoznávat a zařazovat nové informace a podněty z okolí.

Definicí co to vlastně inteligence je, je mnoho a doposud nedošlo ke sjednocení názorů.

"Inteligence je schopnost zpracovávat informace, tedy všechny dojmy, které člověk vnímá." *(J. P. Guilford, dlouholetý prezident Americké psychologické společnosti)*

"Inteligence je všeobecná schopnost individua vědomě orientovat vlastní myšlení na nové požadavky, je to všeobecná duchovní schopnost přizpůsobit se novým životním úkolům a podmínkám." *(Němec Wiliam Stern)*

"Inteligence je vnitřně členitá a zároveň globální schopnost individua účelně jednat, rozumně myslet a efektivně se vyrovnávat se svým okolím." *(Američan David Wechsler)*

Psychologické pojetí inteligence se orientuje na člověka jako na komplex mnoha složek. Některé psychologické směry jasně dokladují, že inteligenci u člověka nemůžeme hodnotit jen na základě nějak sestavených testů.

Lidskou inteligenci můžeme rozdělovat na praktickou, která nám vyjadřuje schopnost manipulovat a pracovat s nástroji, stroji a předměty).

Míra inteligence se s oblibou vyjadřuje a měří pomocí takzvaného IQ testu. Tímto testem lze hodnotit jednak teoretickou schopnost pracovat se symboly, pojmy, znaky, logicky myslet, a jednak sociální schopnost, která je v oblasti v sociálních vztahů a kontaktů. [50]

7.1.1. Inteligenčni kvocient IQ test

Tohoto hodnocení zavedl a popularizoval německý psycholog Stern (Štern) v roce 1912, který pro stanovil vzorec, který říká, že vydělíme mentální věk (změříme jej testem) věkem biologickým. $IQ = \frac{Mentální\ věk}{Biologický\ věk} * 100$

Průměrné hodnocení IQ bylo stanoveno na hodnotě 100. Tento výpočet lze pro jednoduchost používat, ale čím je člověk starší tím více vstupují do hry i další faktory v podobě zkušeností a intuicí. Proto se používá při testování více metod (Wechslerovy testy inteligence, Stanford-Binet Intelligence Scale či Ravenové progresivní matice.)

7.2. Počátky umělé inteligence

Za první významnější počátky počítačové umělé inteligence můžeme považovat její nasazení v NASA o to při jejím používání u raket a družic. O největší rozvoj v umělé inteligenci se však postarala armáda.

Rok 1941, poprvé byl použit elektronický počítač. Bylo to v Německu. Počítač zabíral obrovské místnosti a byl chlazen leteckými motory.

Počátky roku 1950 Norbert Wiener přišel na to, že vlastně všechna inteligentní rozhodnutí jsou založena na principu zpětné vazby (feedback).

Princip zpětné vazby použil už James Watt ve svém parním stroji, i když on asi netušil, že tento objev velmi ovlivní počátky vývoje UI.

Později v roce 1955, Newell a Simon vyvinuli The Logic Theorist, mnohými považovaný za první UI program. Tento program, který reprezentoval každý problém jako tree (stromový) model, se pokoušel vyřešit ho tak, že našel větev, jejíž výsledek byl s největší pravděpodobností ten správný.

V roce 1956 John McCarthy, který je považován za otce UI, zorganizoval konferenci pro všechny zajímající se o strojovou inteligenci. V roce 1958 John McCarthy uvedl svůj nový objev, jazyk LISP, který se užívá dodnes. LISP (LISt Processing - zpracování seznamů) se stal brzy jazykem UI vývojářů.

Koncem šedesátých let vznikl třeba program STUDENT, který dokázal řešit algebraické problémy, nebo program SIR, který rozuměl jednoduchým anglickým větám. Výsledkem těchto programů bylo zdokonalení v porozumění jazykům a logice.

Od 70. let se postupně dostávají na výsluní programy nazývané jako expertní systémy. Tyto programy jsou díky rozvoji výpočetního výkonu dnešních počítačů nasazovány dnes i v normálním životě (třeba vaše komunikace s virtuálním operátorem, či lékařské poradenské systémy).

Rozvoj fotografických čipů kamer dal další směr umělé inteligenci a její nasazení jednak ve výrobních linkách, řízení křižovatek, ostraha objektů a podobně. [53]

7.2.1. Definice umělé inteligence

Je množství definic, co je to vlastně umělá inteligence. Základ asi položil Alan Turing otázkou „mohou stroje myslet“, kterou převedl z oblastí filozofických spekulací na exaktnější úroveň. Vytvořil základy pro test počítacího stroje a jeho zařazení, zda se jedná již o umělou inteligenci.

Ve své původní podobě vychází Turingův test z tzv. imitační hry, ve které, jde o to odlišit dva lidi podle pohlaví. Pozorovatel, na jehož pohlaví nezáleží, má proti sobě např. ženu a muže, který předstírá, že je žena.

Trojice lidí spolu nepřijde do fyzického kontaktu, sedí v oddělených místnostech a zprostředkovatel mezi nimi přenáší popsané lístky. Turingův test spočívá v tom, že imitátorem člověka by byl počítač. Vyvstává tedy otázka, zda-li dokáže počítač simulovat chování ženy stejně dobře jako muž.

Za myslící podle něj prohlásíme PC nebo robotický výtvor tehdy, když jeho chování nebudeme schopni rozeznat od chování člověka. Je to test, kterého se účastní tři lidé v oddělených místnostech a je tam žena a muž, který předstírá, že je žena. Komunikují spolu jen lístečky, žádný fyzický kontakt. A jde o to odlišit dva lidi podle pohlaví. V Turingově testu by člověka nahradil PC. Prozatím vždy neúspěšné.

Rozvoj fotografických čipů kamer dal další směr umělé inteligenci a její nasazení jednak ve výrobních linkách, řízení křižovatek, ostraha objektů a podobně. [53]

Umělá inteligence je věda o vytváření strojů nebo systému, které budou při řešení určitého úkolu užívat takového postupu, který-kdyby to dělal člověk-bychom považovali za projev jeho inteligence (1967 Minsky.).

UI se zabývá tím, jak počítačově řešit úlohy, které dnes zvládají lidé lépe. (1991 E. Richová). Nevýhoda této definice že nezahrnuje úlohy, které neumí řešit jak člověk tak PC.

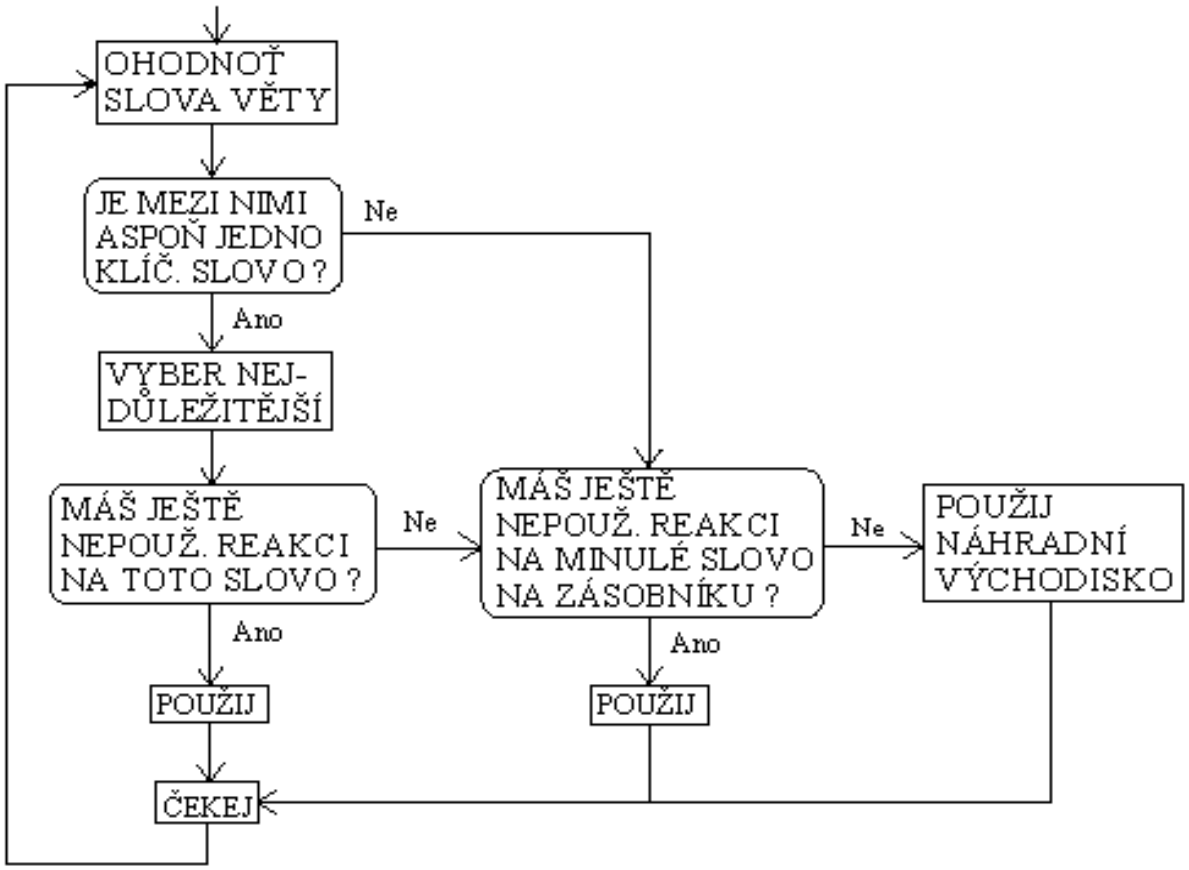
Alternativní definice

UI je označení uměle vytvořeného jevu, který dostatečně přesvědčivě připomíná přirozený fenomén lidské inteligence. [53]

UI označuje tu oblast poznávání skutečnosti, která se zaobírá hledáním hranic a možností symbolické, znakové reprezentace poznatků a procesů jejich nabývání, udržování a využívání. [53]

UI se zabývá problematikou postupů zpracování poznatků – osvojováním a způsobem použití poznatků při řešení problémů. [53]

Do praxe byl nasazen částečně úspěšný program ELIZA, kdy člověk komunikující s tímto programem může uvěřit, že komunikuje s člověkem (ELIZA imitoval psychiatra).



Obrázek 55 Vývojový diagram Elizy [54]

OTÁZKY

1. Co je to Turingův test a v čem spočívá?
2. Co to je vlastně umělá inteligence?
3. Jak souvisí vývoj umělé inteligence s obecnou historií informatiky?
4. Vyjmenujte minimálně 5 úloh, které podle vás vyžadují „inteligenci“.
5. Navrhněte, co by měl mít dům obdařený umělou inteligencí.

7.3. Expertní systémy

Expertní systémy jsou počítačové programy, simulující rozhodovací činnost experta při řešení složitých úloh a využívající vhodně zakódovaných, explicitně vyjádřených znalostí, převzatých od experta, s cílem dosáhnout ve zvolené problémové oblasti kvality rozhodování na úrovni experta (Feigenbaum – 1988).

Další definice postuluje, že expertní systém je počítačový systém hledající řešení problému v rozsahu určitého souboru tvrzení nebo jistého seskupení znalostí, které byly formulované experty pro danou specifickou oblast.

Expertní systémy tvoří znalostní inženýři, kteří společně s odborníky v daném oboru studují chování reálného systému. Snaží se formulovat pravidla a zákonitosti, aby je mohli nadeklarovat v podobě pravidel, kterými se bude řídit budovaný systém.

7.3.1. Charakteristické rysy expertních systémů

Oddělení znalostí a mechanismu pro jejich využívání. Znalosti experta jsou uloženy v bázi znalostí odděleně od inferenčního mechanismu. To umožňuje vytvářet problémově nezávislé (prázdné) expertní systémy (expert system shells), kde jeden inferenční mechanismus může pracovat s různými bázemi znalostí. Předem je dána pouze strategie využívání znalostí z této báze – řídící mechanismus neboli inferenční mechanismus.

Neurčitost (nejistota) v bázi znalostí a neurčitost (nejistota) v datech. Způsob zpracování znalostí a dat v expertním systému musí mít některé rysy podobné způsoby uvažování experta. Expert pracuje velmi často s nejistými znalostmi a s nejistými daty (např. se subjektivním popisem potíží pacienta). I expertní systém musí být schopen využívat nejistých znalostí, tj. znalostí s přidělenou mírou důvěry v jejich platnost – nejistota v bázi znalostí. Zde se pak objevují pojmy jako „často“ „většinou“, které je potřeba kvantifikovat (např. v nějaké škále od „určitě ano“ přes „nevím“ až k „určitě ne“).

Dialogový režim a báze dat. Expertní systémy jsou nejčastěji konstruovány jako tzv. konzultační systémy. Uživatel komunikuje se systémem způsobem „dotaz systému odpověď uživatele“ obdobně jako s lidským expertem.

7.3.2. Typy architektur expertních systémů

Diagnostické expertní systémy provádějí výběr dat s cílem určit, která z možností z předem stanovených možností nejlépe odpovídá reálným údajům vstupujících do systému.

Toto může třeba nastat při vyhodnocování povrchu, po kterém půjde robot. Přísun dat z čidel o drsnosti povrchu, náklonu povrchu, případném protivětru nám dává na výběr z předem definovaných možností vybírat určité varianty a ty potom kombinovat s jinými. Jádrem tohoto systému je řídící mechanismus, který využitím báze znalostí a báze dat po každé příchozí informaci s čidel upřesňuje aktuální chování robota.

Plánovací expertní systémy. Jsou jimi obvykle řešeny takové úlohy, kdy je znám cíl řešení a počáteční stav a systém má využitím dat o konkrétním řešeném případu nalézt posloupnost kroků (operátorů), kterými lze cíle dosáhnout. Podstatnou částí takovýchto expertních systémů je generátor možných řešení, který automaticky kombinuje posloupnost kroků. Lze ukázat, že s rostoucím počtem operátorů, a především s nutným počtem kroků řešení, roste velmi rychle počet kombinací při vytváření posloupnosti kroků. [40]

Výsledkem činnosti plánovacího systému v případě pohybu robota, je návrh případných možných řešení, které jsou nějak ohodnoceny. Algoritmus potom vybere to nejlépe ohodnocené navrhované řešení (pruběhu řešení úlohy se dynamicky mění) a robot jej potom vykoná.

Hybridní systémy se vyznačují kombinovanou architekturou, poněvadž částečně využívají principů diagnostických, částečně plánovacích systémů. K hybridním systémům řadíme například inteligentní výukové systémy či systémy monitorovací. Tyto systémy mají největší využití v robotice.

Prázdné expertní systémy jsou expertní systémy (diagnostické, plánovací) bez báze znalostí a bez báze dat. Doplněním báze znalostí k prázdnému systému se systém teprve orientuje na řešení příslušné problematiky. Dodáním báze dat je pak vždy řešení konkrétního případu. [53]

7.4. Implementace UI

Umělá inteligence se s rozvojem výpočetní techniky stává součástí mnoha odvětví ve společnosti. Velké množství aplikací umělé inteligence je již nasazeno do komerčního prostředí, kde jsou často považovány za samozřejmé, aniž by si lidé uvědomili, že se vlastně jedná o umělou inteligenci. Nikoho nepřekvapí hlášení na nádražích, které je pomocí automatu. Mobilní operátoři již ve velké míře využívají pokročilé systémy umělé inteligence schopné na základě položené otázky vrátit relevantní odpověď v reálném čase.

Společnost IBM předvedla v roce 2011 pokročilý systém umělé inteligence WATSON. Tento systém se zúčastnil americké vědomostní soutěže, kde zvítězil nad svými lidskými soutěžícími.

Watson je postaven na technologii DeepQA, která umožňuje vytvářet hypotézy, shromažďovat velké množství faktů, jejich analýzu a provádět ohodnocení. Dále využívá technologie pro zpracování jazyka, reprezentace znalostí a jejich vyvozování, rozpoznávání a syntézu řeči, vyhledávání informací nebo strojové učení.

K tomu, aby Watson dosahoval nízké odezvy, je využíváno 750 serverů, kde každý server má 8 procesorových jader, každé o frekvenci 3,5 GHz a čtyřech vláknech. Operační paměť je 16 TB.

Jedna z největších amerických bank Citibank plánuje využití technologie Watson, a to v kontextu managementu řízení rizik. Banka předpokládá, že tím, že Watson bude indexovat finanční zdroje jako informace o trzích a Facebook, bude pak schopen rozhodnout, zda konkrétní investice banky bude příliš riziková a před jejím provedením ji tak včas varovat.

Ve zdravotnictví se také předpokládá využití technologie Watson, které bude spočívat v asistenci lékařům při určení správné diagnózy. Watson je schopen zatím pouze číst text, není schopen analyzovat digitální snímky, tudíž není schopen rozeznat tumory na snímcích apod. Předpokládá se ale, že společnost IBM implementuje technologie pro rozpoznávání obrazu tak, aby byl Watson schopen číst snímky, analyzovat EKG atd. [56]

7.4.1. Počítačové vidění

Analýza obrazu, neboli počítačového vidění, jsou jednou z nejdynamičtěji se rozvíjejících disciplín ve světě informačních technologií.

Počítačové vidění je disciplína, která se snaží technickými prostředky aspoň částečně napodobit lidské vidění. Pro počítačové vidění je typická snaha porozumět obecné trojrozměrné scéně, např. takové, jakou zahlédnete při pohledu z okna do zahrady. Postupy počítačového vidění jsou značně složité, s těžištěm v interpretaci obrazových dat, která jsou

nejčastěji reprezentována symbolicky. Jádrem pokročilejších postupů jsou znalostní systémy a techniky umělé inteligence. Této části počítačového vidění se říká vyšší úroveň.

Druhou částí je nižší úroveň. Cílem nižší úrovně je analyzovat vstupní dvojrozměrná obrazová data číselného charakteru a najít kvalitativní symbolickou informaci potřebnou pro vyšší úroveň.

Předmětem zpracování a případné rozpoznání obrazu je obrazová informace o reálném světě, která do počítače vstupuje nejčastěji televizní kamerou. Počítačové vidění řeší úlohu vytvoření explicitního popisu fyzikálních objektů v obraze. Postup zpracování a rozpoznávání obrazu se daří rozložit do posloupnosti základních kroků:

Snímání a digitalizace a uložení obrazu v počítači. Při snímání se převádějí vstupní optické veličiny na elektrický signál spojitý v čase i úrovni. Vstupní informací může být jas (z TV kamery, scanneru), intenzita rentgenového záření, ultrazvuk, tepelné záření aj. Snímat se může v jednom nebo více spektrálních pásmech. Pro barevné snímání stačí tři spektrální složky – červená, zelená a modrá.

Předzpracování obrazu. Cílem je potlačit šum a zkreslení vzniklé při digitalizaci a přenosu dat. Jindy se předzpracování snaží zvýraznit určité rysy obrazu podstatné pro další zpracování. Příkladem může být hledání hran v obraze, tj. obrazových bodů (pixelů) s vysokými hodnotami velikosti gradientu obrazové funkce.

Segmentace obrazu na objekty. Asi nejtěžší krok postupu zpracování je segmentace, která dovolí v obraze najít objekty. Za objekty se považují ty části obrazu, které nás z hlediska dalšího zpracování zajímají. Při segmentaci se tedy zhusta využívá znalosti interpretace obrazu (sémantika).

Popis nalezených objektů. Lze je popsat buď kvantitativně pomocí souboru číselných charakteristik, anebo kvalitativně pomocí relací mezi objekty. Způsob popisu objektů je ovlivněn tím, na co se popis bude používat. Za krajně jednoduchý popis objektů lze považovat např. stanovení velikosti (plochy) objektů, tj. počet jemu odpovídajících obrazových bodů v obraze.

Porozumění obsahu obrazu (klasifikace objektů). Ve velmi jednoduchém případě můžeme za porozumění považovat klasifikaci objektů v obraze podle jejich velikosti. V obecném případě představuje porozumění interpretaci obrazových dat, o kterých se předem nic nepředpokládá. Porozumění obrazu je potom založeno na znalosti, cílech, tvorbě plánu k jejich dosažení a využití zpětných vazeb mezi různými úrovněmi zpracován. [57]

V dnešní době je zcela běžné, že kamery v zabezpečovací technice dokáží rozpoznat pohybujícího člověka, a to ještě v přesně vymezeném prostoru. Taktéž rozpoznávání značek aut na křižovatkách pomocí kamer nikoho z nás již nepřekvapí.

7.4.2. Zpracování přirozeného jazyka

Počítačové zpracování přirozeného jazyka představuje velkou výzvu a perspektivní zaměření výzkumu a vývoje v celé řadě praktických činností člověka s informacemi. Může jít například o:

Databáze textů – textové záznamy je třeba třídit, vyhledávat v nich a to pokud možno s ohledem na obsah těchto textů v přirozeném jazyce.

Překlad textů – úplná náhrada překladatele počítačem nebo různé úrovně podpory překladatele počítačem.

Báze znalostí – úloha automatického učení z textů, neboli automatický převod textových informací do formalizované podoby, ve které by se s nimi dalo snadno logiky manipulovat.

Textové editory – minimálním požadavkem dnes je automatizovaná korektura aspoň na úrovni pravopisu (překlepů), automatická kontrola gramatické a stylové správnosti napsaných textů.

Automatický převod mluvené řeči na text, způsob komunikace mezi člověkem a strojem (např. ovládání operačního systému, získávání informací z databáze, od expertního systému, objednávka, rezervace). [58]

Rozpoznávání hlasu a řeči se stává realitou již v mobilních telefonech. Jsme schopni pomocí telefonu překládat mluvená slova mezi různými jazyky. Bohužel díky složitosti lingvistických převodů zde většinou čeština zatím chybí.

Obrázek 56 Samsung Galaxy S4 Active [59]

7.4.3. OCR

OCR je technologie převodu textu uloženého v bitmapovém formátu do formátu textového. OCR je speciálním případem vektorizace (Optical Character Recognizing), tedy rozpoznávání písma. Text uložený v bitmapě není chápán jako text, je to jen sada tmavých a světlých bodů v obrázku. OCR program tedy musí identifikovat v bitmapě různé tvary a porovnat je s předlohou a rozhodnout jaké písmenko ten který shluk představuje. Situace je navíc zkomplikována tím, že texty bývají napsány v různých fontech a dokumenty bývají často nekvalitní.

I do těchto oblastí již taktéž pronikly mobilní telefony a tablety.

ČAS POTŘEBNÝKE STUDIU 30 minut.

CÍL: Seznámení s čidly a vybavovacími mechanizmy používanými u robotů.

8. ROBOTI A JEJICH PROGRAMOVÁNÍ

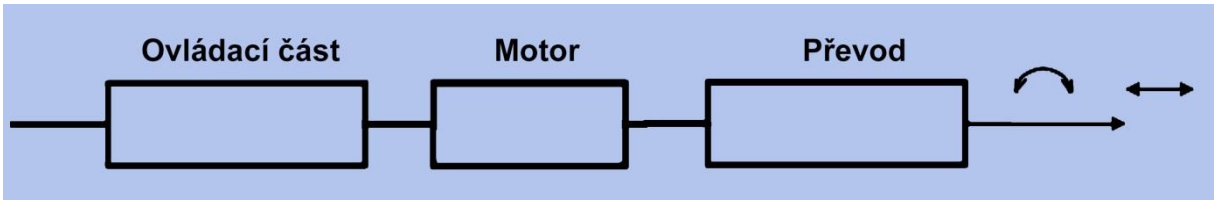
Robotické mechanické zařízení nebo robot jako kybernetický systém obsahuje většinou tři podsystémy. Jedná se o pohybové systémy, snímací systémy a zpracování informací (kognitivní systémy).

8.1. Kognitivní systémy

Kognitivní proces, neboli proces vnímání a racionálního myšlení. Zde patří, všechny řídící systémy robota. Jde jak o programové vybavení (software), které realizuje funkci umělé inteligence, tak o vlastní zařízení (hardware). Některé programové vybavení je autonomní a přímo ve snímacích senzorech již dochází k vyhodnocování, jde tedy o inteligentní senzory.

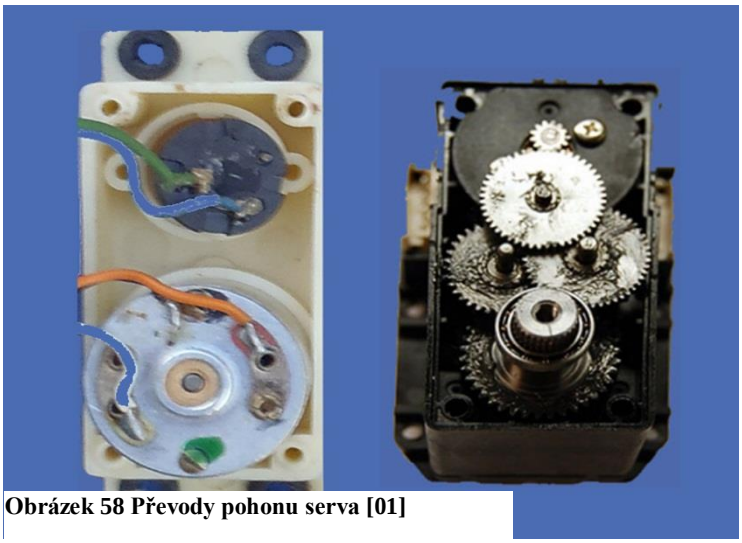
8.2. Motorické systémy a vybavovací mechanismy

Do této kategorie patří veškeré mechanické části robota, včetně pohonů a převodů. Funkcí pohonu je přeměna elektrické energie na mechanický pohyb. Pohony bývají tvořeny elektromotory. Pohyb je většinou realizován pomocí mechanických spojovacích prvků (převodů či táhel) na pohyblivou část mechanické jednotky robota (kolo, paže robota).



Obrázek 57 Obecné schéma pohonu

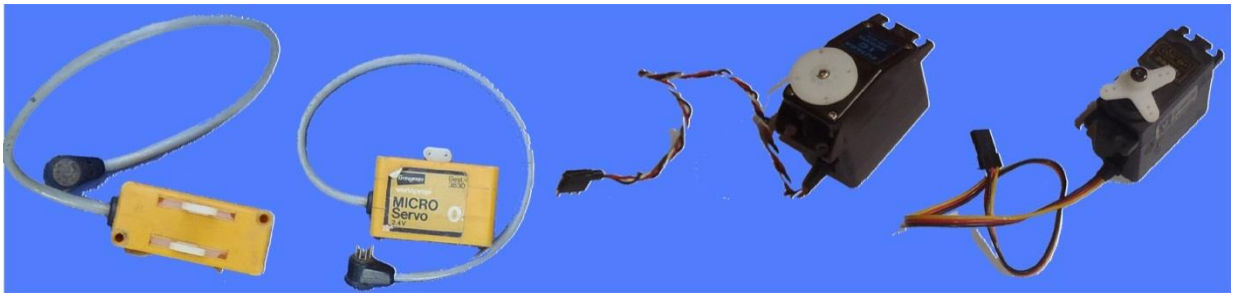
Ovládací část přivádí elektrickou energii do motoru v odpovídající polaritě. V této části je elektronika, která snímá výstupní pohyb mechanické jednotky a parametry motoru. Motory se používají střídavé, stejnosměrné a krokové. Převod může být rotační, nebo přímočarý.



Obrázek 58 Převody pohonu serva [01]

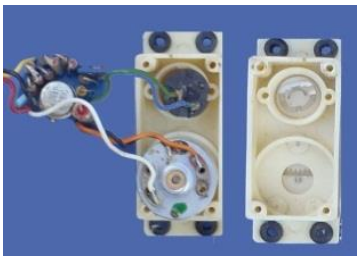
8.2.1. Vybavovací prvky používané u robotu

V prvopočátcích se používaly vybavovací mechanismy převzaté od modelářů. Jednalo se o vybavovací serva, která se otáčela, nebo měla lineární pohyb. Od vybavovacích servopohonů (serv) nebyla žádná zpětná informace o jejich poloze.



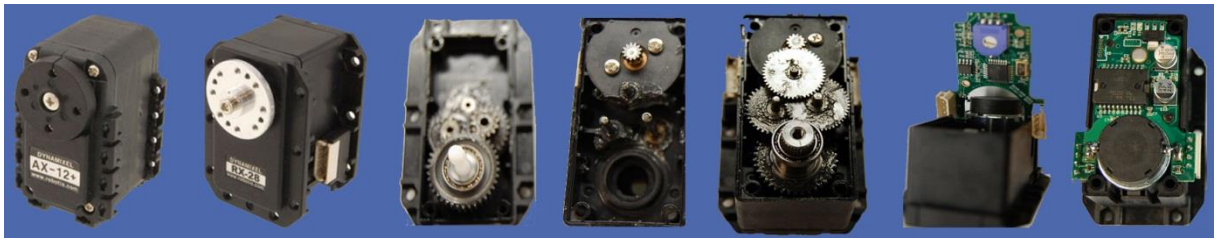
Obrázek 59 Modelářská serva (mechanické vybavovací mechanizmy)

K těmto servům, se ještě dodatečně montovaly snímače polohy (jeden snímač natočení obsahuje každé servo, ale je využit jen pro jeho vnitřní funkci). Tyto serva (elektro mechanické vybavovací prvky) neměly možnost žádného programování.



Obrázek 60 Rozložené servo

Dnešní vybavovací mechanizmy již v sobě mají většinou samostatný mikroprocesor, který nejen ovládá pohybový mechanismus, ale má v sobě i snímače poloh, výkonu, přetížení, krouticích momentů a komunikuje po sériové sběrnici s řídící jednotkou. Těmto jednotkám většinou uživatel může jen nahrát od výrobce nový firmware, který si výrobce chrání.



Obrázek 61 Robotis serva

POJMY K ZAPAMATOVÁNÍ

Mikrokontrolér, paměť flasch.

Von Neumannova architektura a Harvardské schéma.

INTERNETOVÉ ZDROJE DOPORUČENÉ K NAHLÉDNUTÍ

http://videa.seznam.cz/?q=BIOLOID&fulltext#utm_source=search.seznam.cz&utm_medium=hint&utm_term=BIOLOID&utm_content=videa

http://www.hizook.com/blog/2010/03/14/robotis-dynamixel-servos-overview-applications-tear-down-and-open-source-software

8.3. Ovladače

K ovládání robotů se dnes používají již jen bezdrátové ovladače pracující v infračervené oblasti nebo rádiové oblasti spektra. V prvopočátcích se používaly drátové ovladače. Z důvodu dostupnosti se potom rozšířilo používat pro ovládání modelářských radiostanic pracující v pásmu 27 Mhz, 35 Mhz, 40 Mhz, 2,4 GHz.

Obrázek 62 Modelářské vysílačky pacující v pásmu 27 MHz, 40MHz, 2,4 GHz [01]

V dnešní mobilní době samozřejmě komunikujeme pomocí technologie Bluetooth, Wi-Fi, infra skoro se vším. Vyjímkou tedy nejsou ani roboti. Pod pojmem robot si ale dnes již můžeme představit nejen zařízení, co chodí, ale i samo létá a vykonává naplánované činnosti. Tyto létající robotické zařízení, nejen že mají vestavěnou kameru, ale mají někdy i úchopové manipulátory, které jim umožňují provádět mechanické pohyby. Takže vlastně kosmické technologie se vrací zpátky na zem i v podobě hraček.

Robotické modely (třeba Wi-Fi kvadrokoptéra Parrot AR. Drone) se v dnešní době ovládají nejen specializovanými ovladači, ale s potřebným programovým vybavením si s ním poradí mobil či tablet pracující s operačním systémem Android.

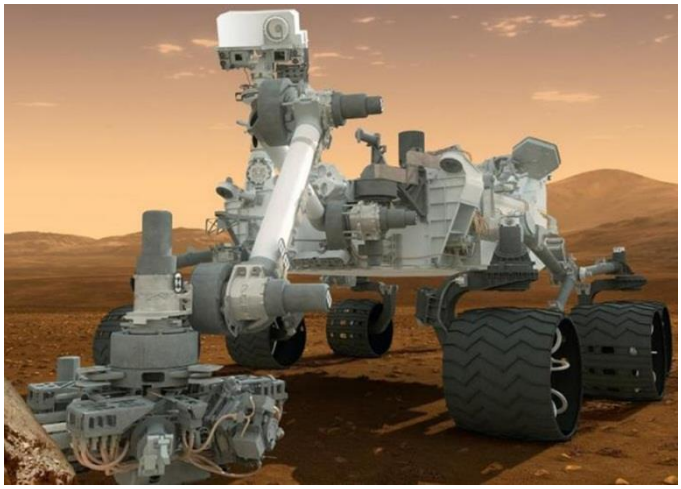
Obrázek 63 Parrot Ar. Drone 2.0(kvadrokoptéra ovládaná pomocí iPhone/iPad/iPod Touch/Androidem) [64]

Robotická zařízení mají svoji naprogramovanou inteligenci, ale dají se taktéž ovládat pomocí kabelových přípojek (vodičů) připojených k počítačům přes nějaké rozhraní (RS232, RS435, USB).

Obrázek 64 Ovládaní Robotis., přes PC, a bezdrátový ovladač [1]

Některé typy ovladačů se programují přímo. Ty potom při vyvolání jedné funkce vyšlou sérii příkazů přímo z vysílače.

Pro laickou veřejnost jsou nejznámější robotičtí představitelé, pohybující se vozítka posílané na vzdálené planety na průzkum, kde vykonávají autonomně určené činnosti. Tyto vozítka se musí rozhodovat samostatně, bez okamžité obsluhy člověka, jelikož řídící a komunikační signály putují k zemi několik minut, což prakticky vylučuje okamžitou obsluh těchto robotických zařízení.



Obrázek 65 Vozítko Curiosity [65]

Na trhu se začíná díky výkonným čipům a stále se zlevňující elektronice objevovat množství robotických hraček a stavebnic. Záleží jen na výrobci, jak svůj výrobek koncipuje. Zda se jedná jen o mechanickou hračku, obdařenou částečnou elektronickou inteligencí, nebo jako zařízení schopné komunikovat s obsluhou a mít možnosti pro programování. Programování může být provedeno třeba jen pomocí dopředu zveřejněných symbolů daných jen pro dotyčný výrobek.

Obrázek 66 Humanoidní robot ROBOSAPIEN X [66]

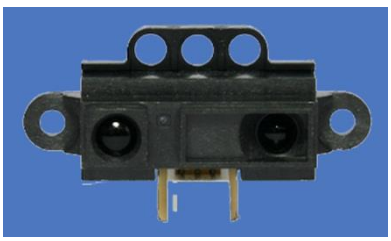
8.4. Snímací čidla používaná u robotů

Každé robotické zařízení se musí orientovat v prostředí a nějakým způsobem se v tomto prostředí pohybovat. Řídící jednotka (počítač) musí vydávat povelové signály pro mechanické projevy robotického zařízení. Pokud má mít ale toto zařízení nějakou inteligenci je nutné, aby řídící jednotka (procesor) dostával informace z okolí, v kterém se nalézá a současně pohybu. Podle těchto vstupních informací se teprve řídící jednotka rozhoduje, na co a jak má reagovat. Pro sběr informací z okolí používáme různé typy čidel. Stejně jak lidé používají oči, uši, hmat, čich, tak i roboti mají specializovaná čidla.

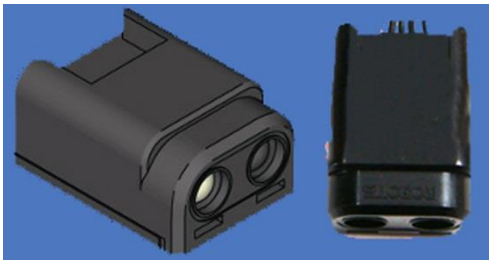
Obrázek 67 Humanoid Robotis Dynamic (Antropomorfní robot s inteligencí) [67]

8.4.1. Snímač infračerveného záření

Infra čidla mají v sobě jen foto diody, nebo fototranzistory pracující ve spektru infračerveného záření. Většinou jsou vestavěná společně v mechanické jednotce i s vysílací infračervenou diodou.



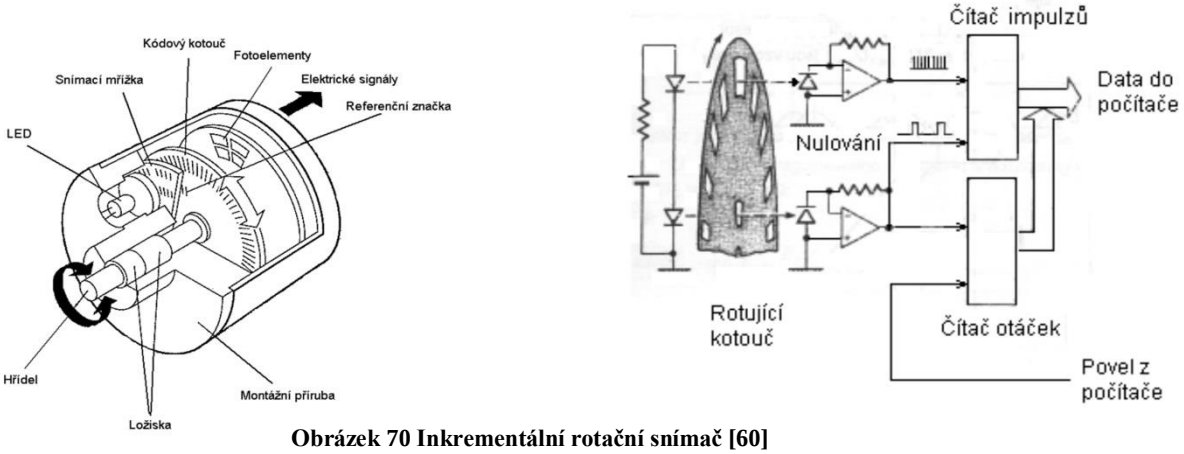
Obrázek 68 DMS Sensor pro měření vzdálenosti



Obrázek 69 IR Sensor ROBOTIS [1]

8.4.2. Inkrementální rotační snímač

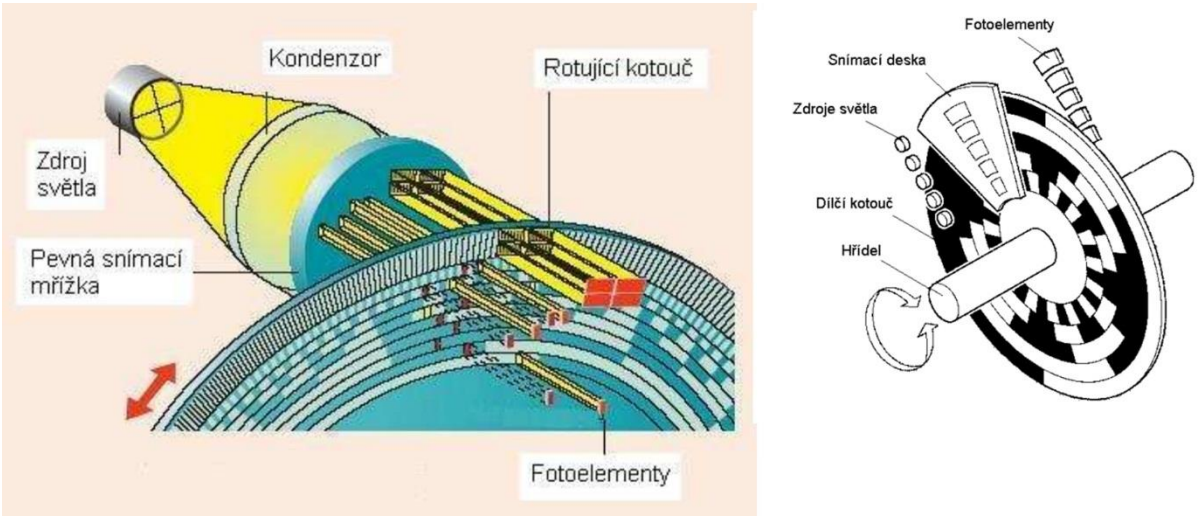
Princip těchto snímačů spočívá ve clonění světelného toku mezi zdrojem světla a fotocitlivými prvky. Pro zjištění informace o rychlosti otáčení stačí zjistit počet impulzů za určitý časový úsek. Pro zjištění směru otáčení je nutno použít rotující kotouč, který má dvě řady otvorů, které jsou vůči sobě posunuty o polovinu šířky otvoru. Pro zjištění úhlu natočení má rotující kotouč ještě jeden otvor, který je určen pro generování nulového impulzu. [60]



Obrázek 70 Inkrementální rotační snímač [60]

8.4.3. Snímače polohy

Absolutní snímač polohy jsou konstruovány na stejném optoelektrickém principu jako čidla inkrementální.

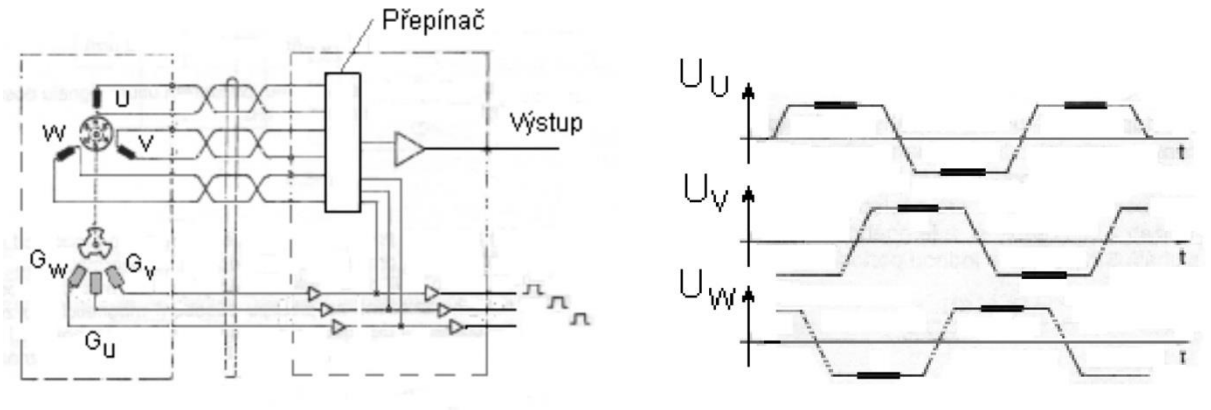


Obrázek 71 Konstrukční uspořádání absolutního snímače [61]

Tento snímač umožňuje zjistit úhel natočení rotoru bezprostředně po připojení napájení bez nutnosti předchozího pootočení. Jejich rotující kotouč má ale několik stop s otvory tvořícími určitý kód. Často se používá kód Grayův. Tento kód se při přechodu do sousední polohy mění pouze v jednom bitu, a proto je detekce a korekce chyb snadná. [61]

8.4.4. Snímače rychlosti

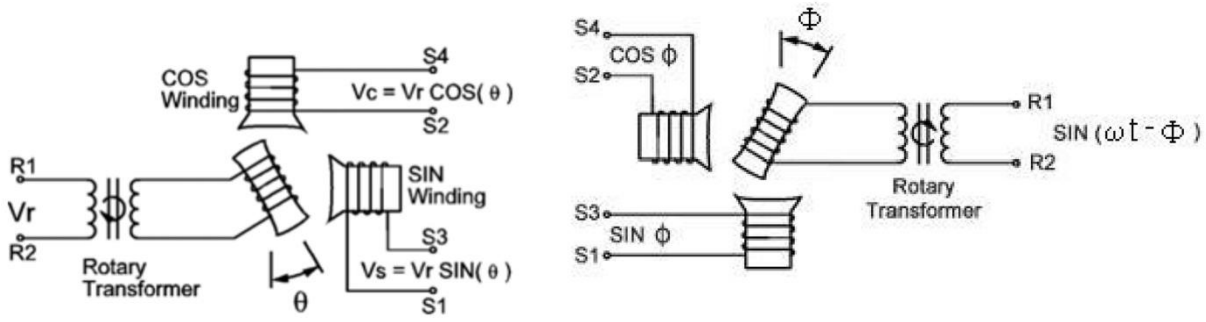
Pro snímání otáčivé rychlosti jsou určeny Tachodynama. Výstupní informace je ve formě analogového signálu. Konstrukčně se jedná se o synchronní x-pólový stroj, jehož rotor s permanentními magnety je uspořádán tak, aby výsledná magnetická indukce ve vzduchové mezeře měla téměř obdélníkový průběh. V důsledku toho se indukuje v trojfázovém vinutí lichoběžníkové napětí, časově posunuté vůči sobě a překrývající se. [62]



Obrázek 72 Elektronicky komutované tachodynamo [62]

8.4.5. Absolutní snímač úhlu natočení

Resolver je polohový transformátor používaný jako absolutní snímač úhlu natočení. Využívá změny vazby (vzájemné indukčnosti) mezi vinutími na rotoru a statoru. Resolver má dvojfázové vinutí na statoru a jednofázový rotor. Vinutí umístěná na statoru jsou vůči sobě o 90 stupňů natočená. Resolver může být napájen do rotoru nebo statoru. V případě použití dvoupólového resolveru získáme vyhodnocením jeho signálů snadno informaci o absolutní poloze, což je předností tohoto snímače. Používání analogového signálu spolu s menší přesností dvoupólového provedení, ale vymezuje jeho použití pro dynamicky méně náročné varianty servopohonů. [63]



Obrázek 73 Resolver [63]

9. POUŽITÉ GRAFICKÉ SYMBOLY A POKYNY KE STUDIU

ČAS KE STUDIU

Čas potřebný k prostudování látky. Čas je pouze orientační a slouží jako hrubé vodítko pro rozvržení studia kapitoly.

CÍL

Cíle, kterých lze dosáhnout prostudováním kapitoly – konkrétní dovednosti, znalosti.

POJMY K ZAPAMATOVÁNÍ

Pojmy, které si je potřeba zapamatovat.

VÝKLAD

Teoretický výklad studované látky, zavedení nových pojmů a jejich vysvětlení.

ŘEŠENÉ PŘÍKLADY

Podrobný postup při řešení příkladů.

SHRNUTÍ POJMŮ

Zopakování hlavních pojmů.

OTÁZKY

Několik teoretických otázek pro ověření zvládnutí kapitoly.

INTERNETOVÉ ZDROJE

Užitečné odkazy na internetu, k dotyčnému tématu.

10. POUŽITÉ ZDROJE

[01] Vlastní tvorba

[02] Freeware [online]. 2007 , 21.11.2010 [cit. 2012-01-10]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/129221-freeware>.

[03] Shareware [online]. 2006 , 24. 5. 2006 [cit. 2007-12-20]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/442623-shareware>.

[04] Adware [online]. 2007 , 27.12.2007 [cit. 2008-03-02]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/469786-adware>.

[05] GNU General Public License [online]. 2007 , 23.12.2007 [cit. 2008-01-05]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/493120-gpl>. Software

[06] Public domain [online]. 2007 , 3.11.2007 [cit. 2007-12-05]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/485385-public-domain>.

[07] Open source software [online]. 2007 , 25.7.2007 [cit. 2007-12-05]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/189262-open-source>.

[08] Svobodný software [online]. 2006 , 10.10.2007 [cit. 2007-12-10]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/12775-free-software>.

[09] BENEŠ, Libor. Vývojové diagramy - 6. díl: Jednoduchá kalkulačka. In: [online]. 1. vyd., 27. 7 .2011 [cit. 2013-06-17]. Dostupné z: http://programujte.com/clanek/2010060400-vyvojove-diagramy-6-dil/

[10] Vyvojovy diagram zarovka. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-12]. Dostupné z: http://cs.wikipedia.org/wiki/Soubor:Vyvojovy_diagram_zarovka.png

[11] Vyhledávací algoritmy: Sekvenční vyhledávání. In: [online]. 2.7.2007 [cit. 2013-04-12] Dostupné z: http://www.manualy.net/article.php?articleID=23

[12] TRETEROVÁ, ELIŠKA. Začínáme programovat v JAVĚ: modulový systém dalšího vzdělávání pracovníků škol a školských zařízení v moravskoslezském kraji. druhé (inovace distanční výukové opory). Ostrava: Ostravská univerzita v Ostravě, 2010, s. 8-12.

[13] TRETEROVÁ, ELIŠKA. Začínáme programovat v JAVĚ: modulový systém dalšího vzdělávání pracovníků škol a školských zařízení v moravskoslezském kraji.. Ostrava: Ostravská univerzita v Ostravě, 2010, s. 115.

[14] ŠVÍGLER, Jan. Algoritmizace a řešení problémů: Vliv typu úlohy na programování. 6.12.2008.

[15] Základní informace o licencích [online]. 2010 [cit. 2011-12-12]. Dostupný z WWW: <http://www.microsoft.com/cze/licence/ZakladniInformace/Licencni Smlouva.mspx>.

[16] Nákup softwaru s počítačem (OEM) [online]. 2012 [cit. 2011-12-15]. Dostupný z WWW: <http://www.microsoft.com/cze/licence/oem/default.mspx>.

[17] Windows Vista Business [online]. 2010 [cit. 2012-02-05]. Dostupný z WWW: <http://www.microsoft.com/cze/licence/images/oem/oem_coa.jpg>.

[18] Demo produkty ke stažení [online]. 2010 [cit. 2007-12-15]. Dostupný z WWW: <http://www.micos-sw.cz/Demo/Registrace/>.

[19] Co je to mikrokontrolér?: Co je to mikrokontrolér, k čemu slouží, kde se používá a jaká je jeho základní struktura…. *Co je to mikrokontrolér?* [online]. [cit. 2013-05-11]. Dostupné z: http://mikrokontrolery-pic.cz/zaciname/co-je-to-mikrokontroler/

[20] Mikrokontrolér: PIC. [online]. [cit. 2013-05-11]. Dostupné z: http://mikrokontrolery-pic.cz/wp-content/uploads/mikrokontrolery-PIC-16bit-PIC24-dsPIC.jpg

[21] Bioloid: Projekty robotů do škol. *Robotis: RoboPlus and C Language Solution* [online]. [cit. 2013-05-11]. Dostupné z: http://www.robotis.com/xe/BIOLOID_main_en

[22] PETERKA, Jiří. Archiv článků a přednášek Jiřího peterky: Von Neumannova architektura. [online]. 2011 [cit. 2013-05-11]. Dostupné z: http://www.earchiv.cz/a93/a321c120.php3

[23] TECH1: Von Neumannovské a Harvardské schéma počítače, popis, funkce. [online]. [cit. 2013-05-11]. Dostupné z: http://ai-fim-uhk.wikispaces.com/TECH1

[24] SPŠ A VOŠ PÍSEK, MediaWiki. MediaWiki SPŠ a VOŠ Písek:MediaWiki SPŠ a VOŠ Písek: Harvardská architektura. [online]. 27. 5. 2010. 2011 [cit. 2013-05-11].

[25] Bio + All + Droid = BIOLOID: Bioloid robot parts. [online]. [cit. 2013-06-24]. Dostupné z: http://www.robotis.com/xe/BIOLOID_main_en

[26] JAHODA, Václav. Průmyslová robotika: Animace v Inventoru I. In: *Autodeskclub* [online]. 28.5.2012 [cit. 2013-06-24]. Dostupné z: http://www.autodeskclub.cz/clanek/5876-prumyslova-robotikaanimace-v-inventoru-i

[27] Jak vzniká program: Programovací jazyk, zdrojový kód a překladač. [online]. [cit. 2013-06-21]. Dostupné z: http://www.sallyx.org/sally/c/c03.php

[28] Jak vzniká program: Přenositelnost. [online]. [cit. 2013-06-21]. Dostupné z: http://www.sallyx.org/sally/c/c03.php

[29] Úvod a historie C++. In: *Devbook: Programátorská sociální síť* [online]. [cit. 2013-06-21]. Dostupné z: http://www.devbook.cz/kurz-cpp-uvod-do-sveta-cpp

[30] Učebnici Assemleru: Terminologie. [online]. [cit. 2013-06-21]. Dostupné z: http://conmet.cz/assembler/uc02.htm

[31] Výuka assembleru: 1. Začínáme s assemblerem. ZEZULA, Ladislav. [online]. 2003 [cit. 2013-06-21]. Dostupné z: http://www.zezula.net/cz/teach/assembler_01.html

[32] ASSEMBLER: Asm obecně. [online]. [cit. 2013-06-21]. Dostupné z: http://k-prog.wz.cz/progjaz/assemb.php

[33] PROGRAMOVACÍ JAZYKY: Dělení programovacích jazyků. [online]. [cit. 2013-06-21]. Dostupné z: http://k-prog.wz.cz/progjaz/index.php

[34] Python Zdroj: https://cs.wikipedia.org/w/index.php?oldid=10307663 Přispěvatelé: -jkb-, Adam Zivner, Aleš Tošovský, Amper, BobM, Brumla, Che, Chrupoš, DaBler, Danny B., Frosty.CZ,

[35] *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-21]. Dostupné z: https://cs.wikipedia.org/wiki/Python

[36] PYTHON: Úvod k Pythonu. [online]. [cit. 2013-02-21]. Dostupné z: http://k-prog.wz.cz/python/index.php

[37] 3. díl - Úvod a historie C++: Jazyk C++ a objektově orientované programování. *Devbook: Prográmatorská sociální síť* [online]. [cit. 2013-06-21]. Dostupné z: http://www.devbook.cz/kurz-cpp-uvod-do-sveta-cpp

[38] PASCAL: Úvod do Pascalu. [online]. [cit. 2013-06-21]. Dostupné z: http://k-prog.wz.cz/pascal/index.php

[39] ŠTEFAN, Radim. *Programování: studijní obor: informační technologie ve vzdělávání*. Vyd. 1. Ostrava: Ostravská univerzita, Pedagogická fakulta, 2002, s. 12. Informační technologie ve vzdělávání. ISBN 80-7042-254-8.

[40] C++. In: [online]. [cit. 2013-06-25]. Dostupné z: http://www.bloodshed.net/images/devcpp5_scr.jpg

[41] NAGYOVÁ, Ingrid. *Základy programování: vývojové prostředí DELPHI*. Vyd. 1. Ostrava: Ostravská univerzita, Pedagogická fakulta, s. 3. Informační technologie ve vzdělávání.

[42] ŠTEFAN, Radim. *Autorské systémy: vývojové prostředí DELPHI : studijní obor: Informační technologie ve vzdělávání*. Vyd. 1. Ostrava: Ostravská univerzita, Pedagogická fakulta, 2002, s. 20. Informační technologie ve vzdělávání. ISBN 80-7042-253-x.

[43] 1. díl - Programování v jazyce C - Úvod. In: *Devbook: Prográmatorská sociální síť* [online]. [cit. 2013-06-21]. Dostupné z: http://www.devbook.cz/jazyk-c-uvod-tutorial

[44] Oracle: Java Embedded Technology Enables End-to-End Solutions. [online]. [cit. 2013-06-21]. Dostupné z: http://www.arm.com/community/software-enablement/oracle-foundation-page-22516.php

[45] JAVASCRIPT: Co je JavaScript. [online]. [cit. 2013-06-21]. Dostupné z: http://k-prog.wz.cz/javascr/index.php

[46] Smalltalk: Co je to Smalltalk. JANOUŠEK, Vladimír. [online]. [cit. 2013-06-26]. Dostupné z: http://www.comtalk.net/Squeak/15

[47] *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-06-26]. Dostupné z: http://cs.wikipedia.org/wiki/PHP

[48] HABIBALLA, Hashim. *Prolog: distanční studijní opora*. Ostrava: Ostravská univerzita, Pedagogická fakulta, 2003, s. 3. Informační technologie ve vzdělávání.

[49] HABIBALLA, Hashim. *Prolog: distanční studijní opora*. Ostrava: Ostravská univerzita, Pedagogická fakulta, 2003, s. 11. Informační technologie ve vzdělávání.

[50] Inteligence. [online]. [cit. 2013-05-06]. Dostupné z: http://www.inteligence.cz/

[51] KŘIVÝ, Ivan a Evžen KINDLER. *SIMULACE A MODELOVÁNÍ 1: Přírodovědecká fakulta*. OSTRAVSKÁ UNIVERZITA, 2003. Přírodovědecká fakulta.

[52] Robot: Robůtek. [online]. [cit. 2013-06-26]. Dostupné z: http://sti.discipline.ac-lille.fr/technologie-college

[53] HABIBALLA, Hashim. *UMĚLÁ INTELIGENCE: UČEBNÍ TEXTY OSTRAVSKÉ UNIVERZITY*. Ostravská Univerzita, 2004. Distanční studijní opora.

[54] Téma umělá inteligence: Vývojový diagram Elizy. [online]. [cit. 2013-06-26]. Dostupné z: http://www.phil.muni.cz/fil/sci-fi/osnova07.html

[55] KOSEK, Jiří. JavaScript: JavaScript za devět a půl minuty. [online]. 1998 [cit. 2013-06-26]. Dostupné z: http://www.kosek.cz/clanky/dhtml/skripty.html

[56] JIRÁSEK, Petr. PB016 Úvod do umělé inteligence: Aplikace umělé inteligence – Watson a Quill. [online]. 2012, s. 14 [cit. 2013-06-26]. Dostupné z: http://petrjirasek.cz/files/watson-referat.pdf

[57] OBLASTI VYUŽITÍ UMĚLÉ INTELIGENCE: Počítačové vidění. In: [online]. [cit. 2013-06-26]. Dostupné z: http://majdulenka.webzdarma.cz/UMI/videni.htm

[58] OBLASTI VYUŽITÍ UMĚLÉ INTELIGENCE: Porozumění přirozenému jazyku. In: [online]. [cit. 2013-06-26]. Dostupné z: http://majdulenka.webzdarma.cz/UMI/jazyk.htm

[59] Samsung Galaxy S4 Active. [online]. [cit. 2013-06-26]. Dostupné z: http://www.ubergizmo.com/wp-content/uploads/2013/06/att-galaxy-s4-active.jpg

[60] Snímače otáček a polohy: Inkrementální rotační snímač. [online]. [cit. 2013-06-26]. Dostupné z: http://www.mti.tul.cz/files/svm/Snimace_polohy.pdf

[61] Snímače otáček a polohy: Absolutní snímač polohy. [online]. [cit. 2013-06-26]. Dostupné z: http://www.mti.tul.cz/files/svm/Snimace_polohy.pdf

[62] Snímače otáček a polohy: Elektronicky komutované tachodynamo. [online]. [cit. 2013-06-26]. Dostupné z: http://www.mti.tul.cz/files/svm/Snimace_polohy.pdf

[63] Snímače otáček a polohy: Resorver. [online]. [cit. 2013-06-26]. Dostupné z: http://www.mti.tul.cz/files/svm/Snimace_polohy.pdf

[64] Parrot Ar.Drone 2.0. [online]. [cit. 2013-06-27]. Dostupné z: http://onedrone.com/store/image/data/Parrot/ardrone-2-buy-shop.jpg

[65] ČT zpravodajství -svět: NASA pošle na Mars další robotické vozítko. [online]. 5. 12. 2012 [cit. 2013-06-27]. Dostupné z: http://www.ceskatelevize.cz /ct24/svet/206230-nasa-posle-na-mars-dalsi-roboticke-vozitko/

[66] ROBOSAPIEN X: Kultovní humanoidní robot. [online]. 5. 12. 2012 [cit. 2013-06-27]. Dostupné z: http://www.wowwee.com/en/products/toys/robots/robotics/robosapiens /robosapien-x

[67] Humanoid Project: Darwin-OP (Dynamic Antropomorfní robot s inteligencí. *Robotis* [online]. [cit. 2013-06-27]. Dostupné z: http://www.robotis.com/xe/darwin_en

11. SEZNAM OBRÁZKŮ

12. SEZNAM VLOŽENÝCH TEXTOVÝCH POLÍ (TEXTŮ)

ROBOTI

VE ŠKOLE PRO PRAKTICKOU VÝUKU, MOTIVACI I ZÁBAVU

CZ.1.07/1.1.24/01.0066

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

ZÁKLADY PROGRAMOVÁNÍ

Mgr. Vladislav BEDNÁŘ