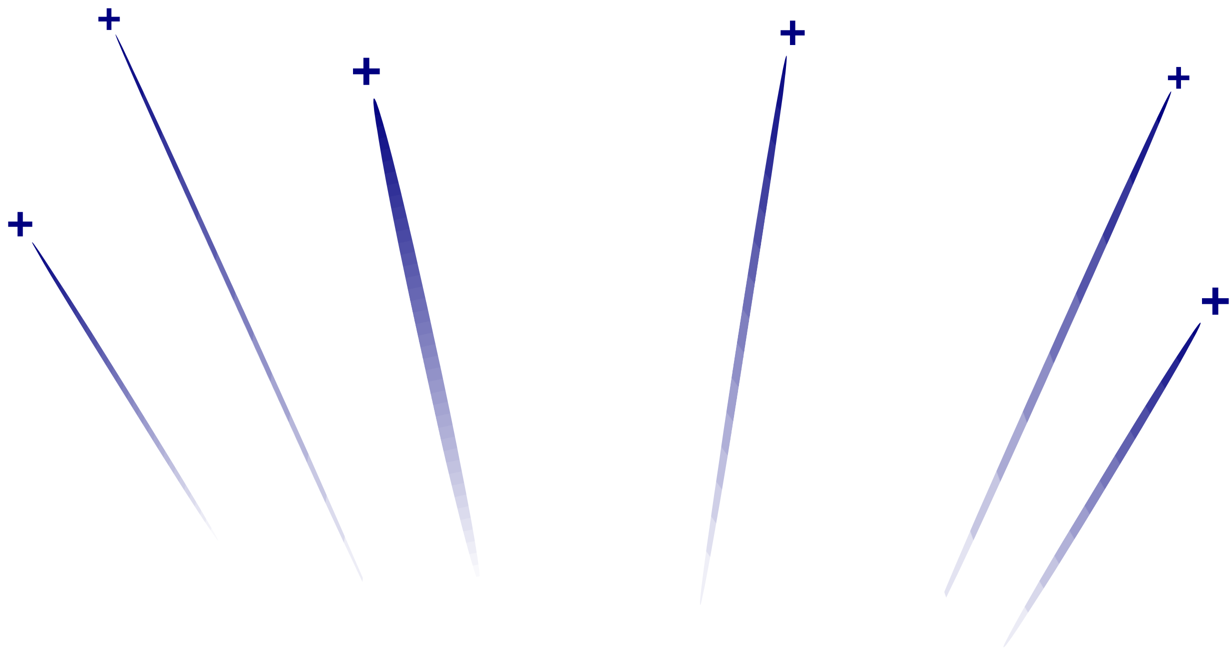
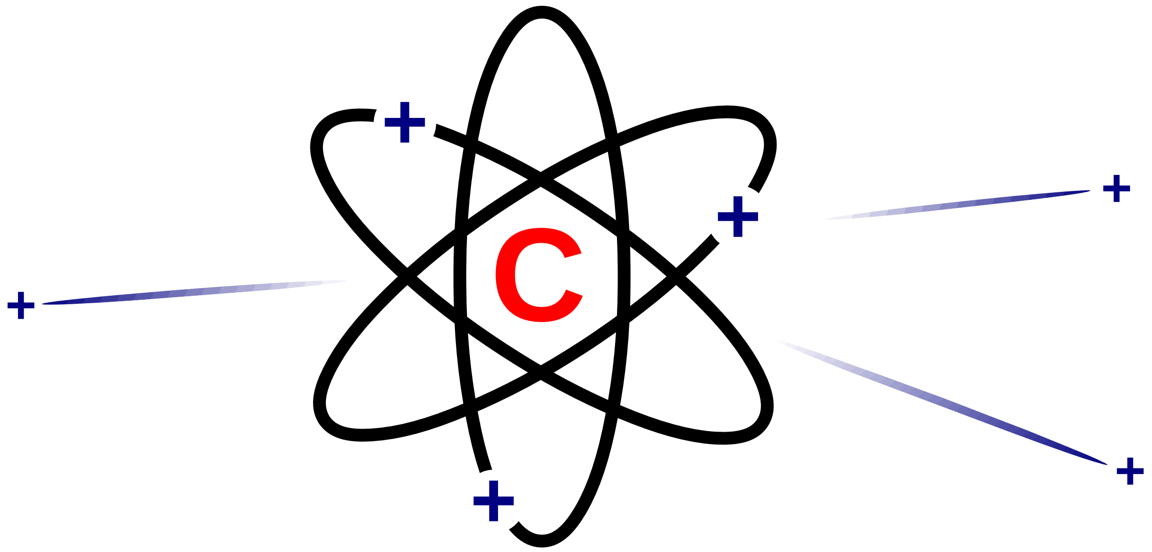
Martin Šechný



Programovanie v C/C++



2016

verzia 1.15

poďakovanie — *V tomto učebnom texte som zosumarizoval množstvo poznatkov z rôznych zdrojov, skúsenosti učiteľov viacerých škôl a hlavne informácie od komunity ľudí pracujúcich so slobodným a otvoreným softvérom.*

podporovatelia — *Radoslav Feďo* — *http://www.elekit.net*

http://www.eii.sk

názov — *Programovanie v C/C++*

URL — *http://www.shenk.sk/skola/programovanie/Programovanie-C.pdf*

HTML verzia — *http://www.shenk.sk/skola/programovanie/*

Obsah

1 Úvod

Nudná opakujúca sa práca? Zišlo by sa automatizovanie úloh, komunikácie? Chýba v počítači aplikácia na konkrétny účel? Keby sme čakali, že to urobí niekto iný, bolo by to príliš jednoduché. My chceme viac. Len takto napreduje veda a technika.

Obrázok 1: Európska iniciatíva pre podporu programovania, http://www.allyouneediscode.eu

Programovanie je nutná zručnosť v 21. storočí. Nie len pre informatikov. Každý potrebuje vedieť programovať. Lebo to, čo stroj môže urobiť pre nás, je riadené programom. Človek aj v tomto len kopíruje prírodu. Všetko živé je riadené kódom DNA:

Obrázok 2: DNA kód

Podobne, ako živý organizmus rozumie DNA kódu a človek rozumie jazyku, ktorým hovorí, tak počítačový program je napísaný v kóde určenom pre počítač – v programovacom jazyku. Existuje veľa jazykov. Pred tým, než si vyberieme vhodný jazyk, potrebujeme danú úlohu algoritmizovať.

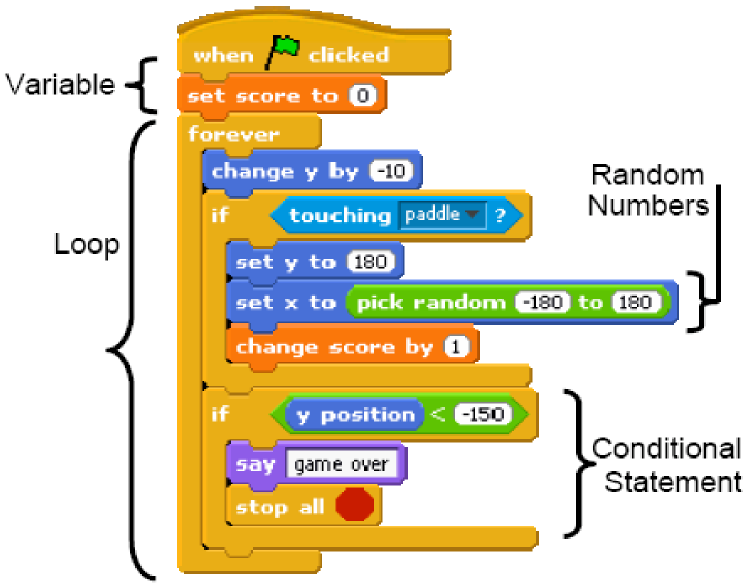
Algoritmus je jednoznačný postup na riešenie úlohy. Môže byť vyjadrený slovne, videom, obrázkom (diagramom), alebo programovacím jazykom.

Program je algoritmus zapísaný v konkrétnom programovacom jazyku.

Tento učebný text je pre technikov-informatikov, je určený pre tých, čo už zvládli základy programovania v nejakom jednoduchom jazyku. Aj keď úvodné príklady v jazyku C sú triviálne, technické programovanie je celkovo náročné. Programovanie v jazykoch C/C++ sa opiera v prvom rade o ľahké základy algoritmizácie a programovania v jazyku pre začiatočníkov:

Obrázok 3: Scratch

Scratch[1] je jednoduchý, ľahko zvládnuteľný programovací jazyk s prostredím pre deti alebo úplných začiatočníkov. Algoritmus sa vytvára grafickým ukladaním príkazov:

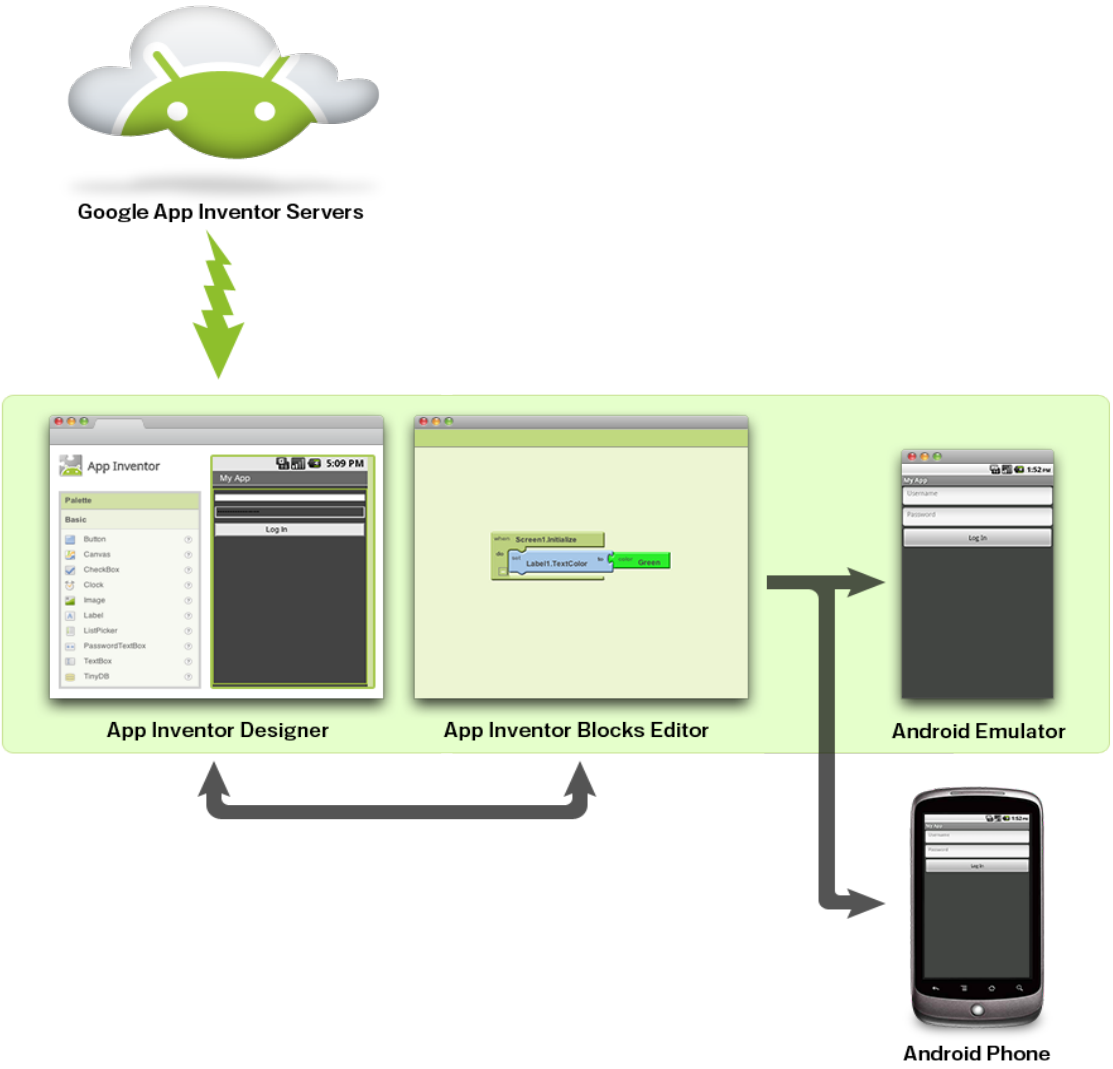


Obrázok 4: Scratch – algoritmus

1 *Scratch*, https://scratch.mit.edu, https://sk.wikipedia.org/wiki/Scratch, https://en.wikipedia.org/wiki/Scratch_%28programming_language%29, http://scratched.gse.harvard.edu/, http://scratch.ie

Scratch CUP je súťaž v programovaní v jazyku *Scratch* pre žiakov ZŠ.[2] Na webe sa dá nájsť množstvo súťaží pre programátorov v rôznych jazykoch.

Ďalším vhodným programovacím jazykom a prostredím pre začínajúcich programátorov je *Google/MIT App Inventor for Android.*[3]

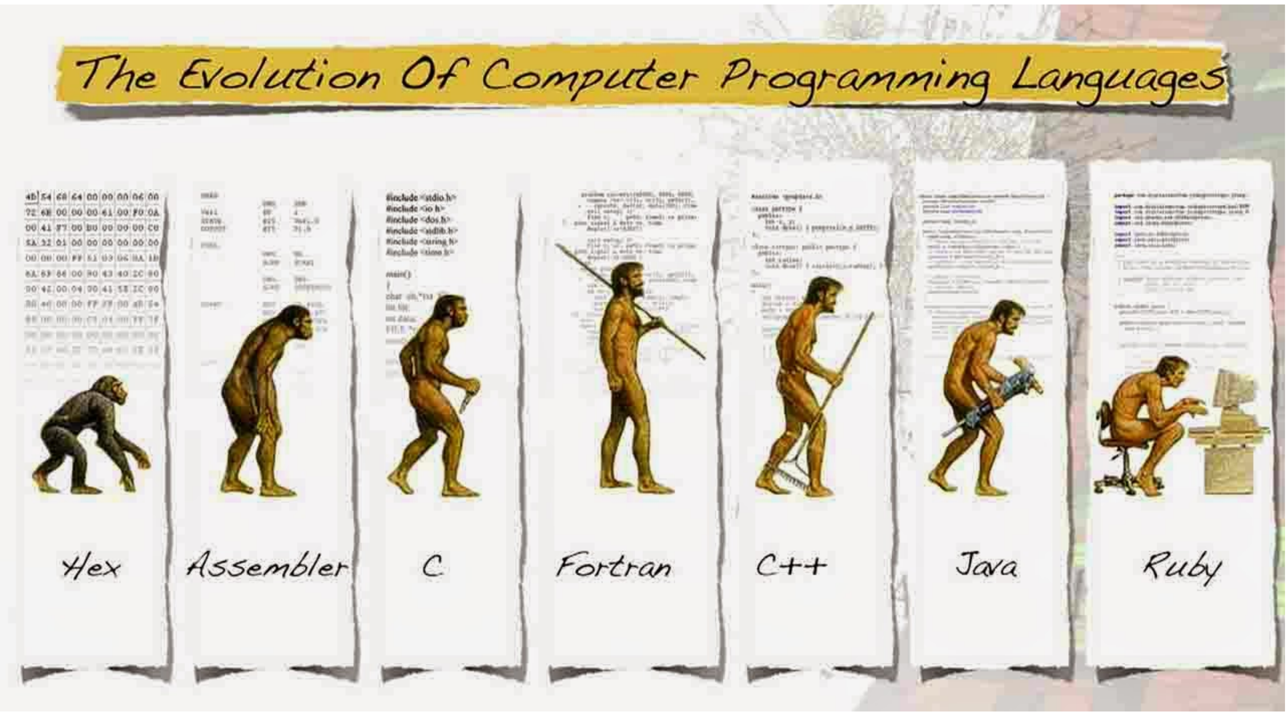


Obrázok 5: App Inventor

Nebudeme sa ďalej venovať programovacím jazykom *Scratch* alebo *App Inventor*.

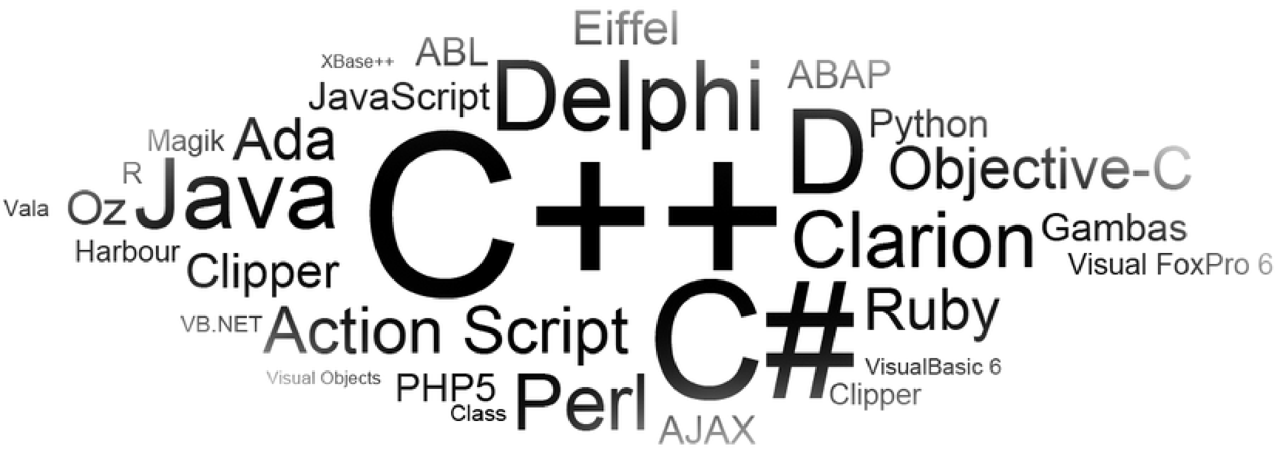
Programovanie v jazykoch C/C++ sa okrem základov algoritmizácie a programovania opiera aj o pochopenie fungovania počítača (hardvéru a operačného systému), pretože jazyk C bol pôvodne určený na programovanie operačných systémov. To robí z jazyka C nízko-úrovňový programovací jazyk. Časté využívanie knižníc a moderných konštrukcií z jazyka C robí tiež vyšší programovací jazyk pre programovanie aplikácií. Objektový návrh vo vyššom jazyku C++ je vhodný pre tvorbu rôznych aplikácií. Základom programovania v jazykoch C/C++ je dobré zvládnutie syntaxe oboch jazykov, preto v tomto učebnom texte začneme triviálnymi príkladmi na precvičenie základných konštrukcií v jazyku C a potom v jazyku C++. Neskôr pridáme objektový návrh programu. Príklady grafických aplikácií v jazykoch C a C++ sú náročnejšie na čítanie a pochopenie zdrojového kódu, preto sú na konci kapitol.

2 *Scratch CUP*, http://edi.fmph.uniba.sk/~tomcsanyiova/ScratchCup/
3 *Google/MIT App Inventor for Android*, http://appinventor.mit.edu/explore/



Obrázok 6: Evolúcia programovacích jazykov s humorom

Existuje veľa programovacích jazykov. Jazyk C je typicky procedurálny a funkcionálny jazyk, má bližšie k hardvéru a jazyk C++ je typicky objektový jazyk, má bližšie k aplikáciám.



Obrázok 7: Objektové programovacie jazyky

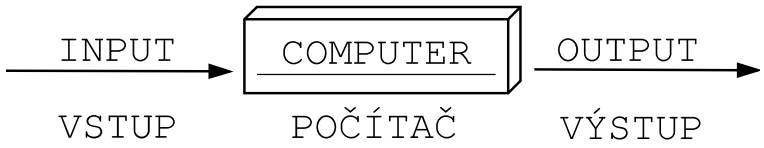
Znalosť programovania je cenená. Dobrý programátor si vie nájsť dobré uplatnenie.

Obrázok 8: Nauč sa a zarábaj

2 Počítač

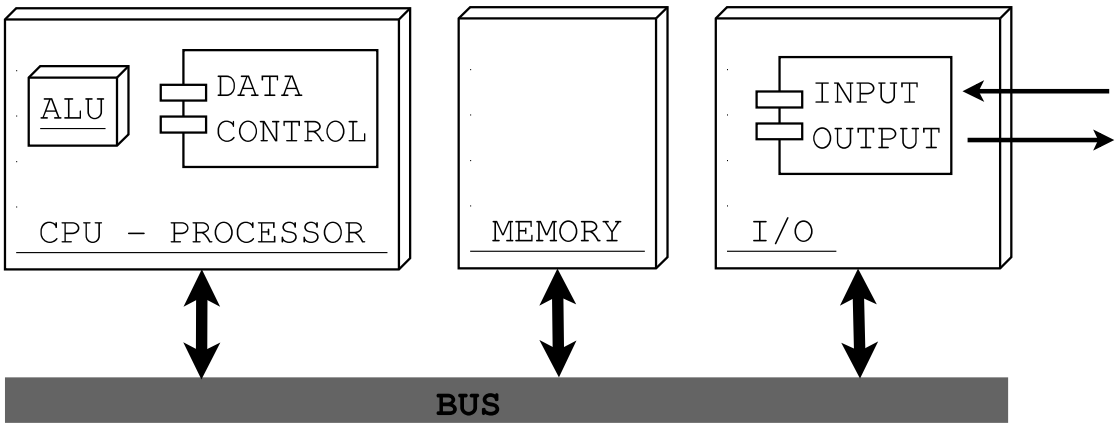
Najprv potrebujeme poznať základné pojmy pre pochopenie fungovania počítača:

- ✔ **Údaj, dáta** (*data*) – fakty, ktoré je možné zaznamenať do pamäte.
- ✔ **Informácia** (*information*) – údaj, ktorý má zmysel, význam, hodnotu pre používateľa, alebo ktorého pravdivosť možno overiť. Jednotka informácie je 1 bit (b), 1 bajt (B) = 8 b.
- ✔ **Znalosť** (*knowledge*) – zložená informácia z viacerých elementárnych informácií, informácia s kontextom, súbor informácií s hodnotou väčšou, ako súčet hodnôt samostatných informácií.
- ✔ **Počítač** (*computer*) – (elektronický, analógový, digitálny) stroj s vlastnou pamäťou, spracúvajúci vstupy podľa programu, odovzdávajúci výstupy.



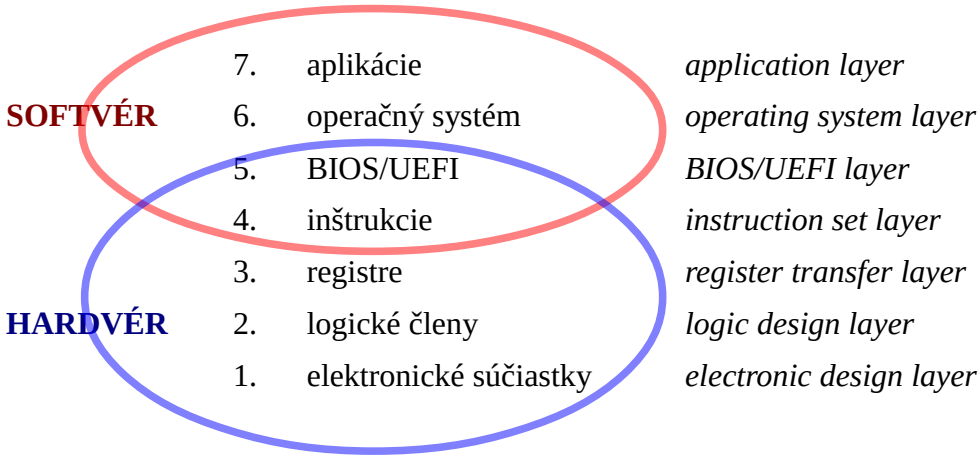
Obrázok 9: Počítač

- ✔ **Hardvér** (*hardware*) – fyzické súčasti počítača.



Obrázok 10: Moderná bloková architektúra počítača

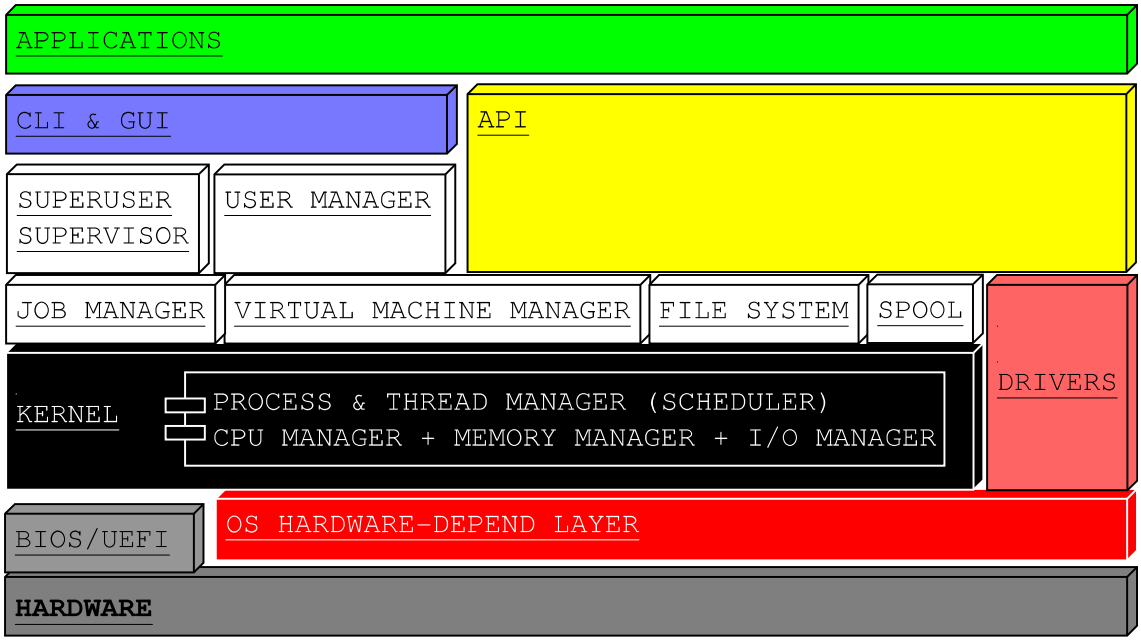
- ✔ **Firmvér** (*firmware*) – program obsluhujúci daný hardvér, je hardvérovo závislý, určený len pre daný typ hardvéru, v počítači sa označuje ako BIOS/UEFI.
- ✔ **Softvér** (*software*) – programové súčasti počítača nahraté v niektorej fyzickej pamäti.



Obrázok 11: Abstraktný hierarchický model počítača

✔ **Softvér** (*software*) delíme na 2 hlavné skupiny:

- aplikačný (aplikácie)
- systémový (BIOS/UEFI, operačný systém, príkazový interpreter, prekladač prg. jazyka)



Obrázok 12: Bloková štruktúra operačného systému

✔ **OS, operačný systém** (*Operating System*) – základný program v počítači, nutný na to, aby používateľ mohol s počítačom pracovať. Operačný systém sprístupňuje používateľovi hardvér a softvér – spravuje procesor, pamäť, vstupné a výstupné zariadenia, riadi procesy, používateľov, oprávnenia, zabezpečuje ukladanie súborov na disk, inštalovanie a spúšťanie aplikácií. Jadro operačného systému je nutný základ systému, ostatné časti sú voliteľné moduly.

✔ **Príkazový interpreter** (*command interpreter, shell, user interface*) – program, ktorý vykonáva príkazy zadané používateľom. Môže byť ich viacero k jednému operačnému systému. Niektoré sú textové, konzolové a niektoré sú grafické.

CLI (*Command Line Interface, console*) – textový, konzolový príkazový interpreter.

GUI (*Graphical User Interface*) – grafický príkazový interpreter, používateľské prostredie.

Všeobecne sa pre používateľské prostredie používajú tieto skratky:

UI (*User Interface*) – technické označenie, môže byť CLI alebo GUI

UX (*User eXperience*) – marketingové označenie, kde sa zdôrazňuje „zážitok“ používateľa

Ďalšie pojmy potrebné pre programovanie:

✔ **Programovanie** (*programming*) – vytvorenie postupu na riešenie problému, výsledkom je program (softvér).

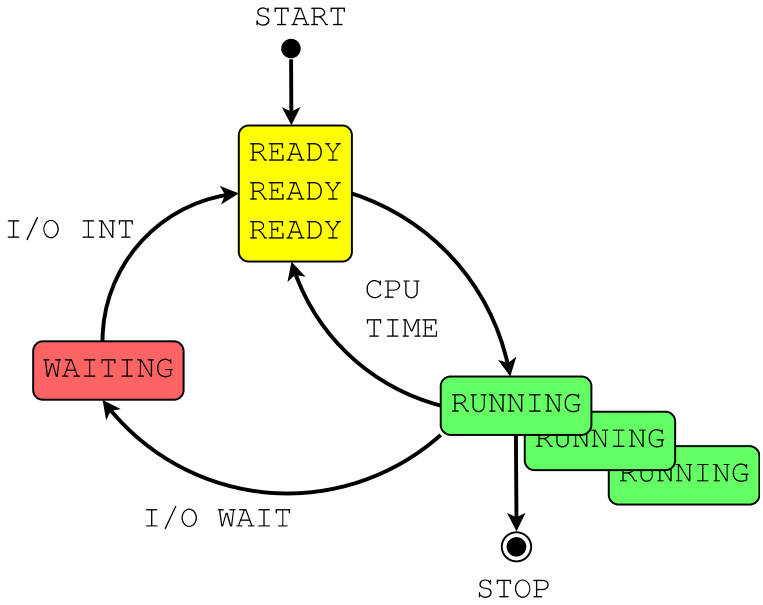
✔ **Prekladač programovacieho jazyka** (*compiler*) – program prekladajúci zdrojový kód v programovacom jazyku do strojového kódu, ktorému rozumie počítač.

✔ **Aplikácia** (*application, app*) – program určený pre používateľa, napr. kancelársky balík, webový prehliadač, ekonomický softvér, hra.

✔ **API** (*Application Program Interface*) – knižnica pre priamy prístup aplikácie k operačnému systému bez používateľa. Knižnice zaberajú značnú časť diskovej kapacity obsadenej operačným systémom.

- ✔ **Algoritmus** (*algorithm*) – jednoznačný postup na riešenie úlohy. Môže byť vyjadrený slovne, videom, obrázkom (diagramom), alebo programovacím jazykom.
- ✔ **Program** (*program*) – algoritmus zapísaný v programovacom jazyku, zdrojový kód.
- ✔ **Proces** (*process*) – program bežiaci v počítači, uložený v hlavnej pamäti.

Jadro operačného systému (*kernel*) prideľuje procesom čas na procesore (v milisekundách), strieda ich a používateľovi sa to javí, akoby všetky procesy bežali súčasne:



Obrázok 13: Stavový diagram procesov

- ✔ **Proces** (*process*) – bežiaci program na jednom CPU (na jednom procesore, ak ich nie je viac).
- ✔ **Vlákno** (*thread*) – časť procesu bežiaca na jednom CPU (na jednom jadre procesora alebo na jednom z viacerých jedno-jadrových procesorov).

Proces alebo vlákno môže požadovať od CPU vykonanie operácií alebo inštrukcií s viacerými úrovňami privilégií, napr. práca s I/O, práca s pamäťou, zmena priority pre plánovač atď. Kvôli zvýšeniu spoľahlivosti a bezpečnosti operačného systému je každému procesu alebo vláknu pridelený CPU režim (*CPU mode*), najčastejšie len z dvoch – privilegovaný režim (*kernel mode*), chránený režim *(user mode)*. Kernel vždy beží v plnom privilegovanom režime a všetky ostatné procesy a vlákna, ktoré nepotrebujú privilégiá, bežia v chránenom režime. Režim je kontrolovaný priamo procesorom, chráni tým operačný systém.

Špecifikácia IEEE **POSIX**[4] (*Portable Operating System Interface for UNIX*) zaručuje kompatibilitu operačných systémov medzi sebou, kompatibilitu komunikácie aplikácií s operačným systémom cez API (*Application Program Interface*), požiadavky na spracovanie v reálnom čase.

Vďaka štandardizácii programovacích jazykov **C/C++** (ANSI/ISO C[5], ISO/EIC C++[6]) máme k dispozícii univerzálny, používaný a dobre dokumentovaný spôsob, ako programovať operačné systémy. Práve jazyky C/C++ sú určené a používajú sa na programovanie operačných systémov.

4 POSIX, http://en.wikipedia.org/wiki/POSIX
5 C, http://en.wikipedia.org/wiki/C_language
6 C++, http://en.wikipedia.org/wiki/C++

3 Programovacie jazyky

Ľudská reč je dorozumievací jazyk slovom a písmom. Takýto jazyk je bohatý, ale nepresný, nejednoznačný. Človek používa aj frázy, metafory, v komunikácii odovzdáva okrem faktov emócie. Ľudská reč nie je vhodná na programovanie. Dnes už počítač vie komunikovať v ľudskej reči s obmedzeným slovníkom. Napriek mnohým rokom vývoja je umelá inteligencia a empatia ešte len na začiatku. Ubehne veľa rokov, kým bežný počítač bude schopný komunikovať s človekom v prirodzenej ľudskej reči tak, aby človek nerozoznal, že s ním komunikuje stroj.[7]

3.1 Čo je jazyk?

Informatika potrebuje zadefinovať jazyk ako jednoznačný a presný prostriedok pre zápis dát, zápis algoritmu a komunikáciu.

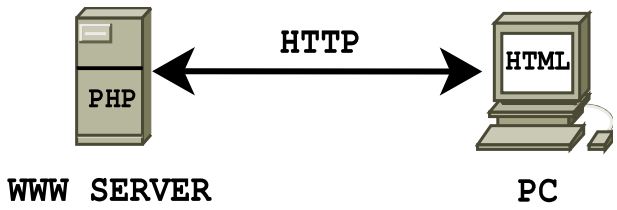
- **Formálny jazyk**[8] je množina slov konečnej dĺžky, zložených zo symbolov abecedy.
- **Abeceda**[9] je konečná množina symbolov (znakov).
- **Gramatika**[10] je konečná množina pravidiel pre tvorbu jazyka zo symbolov abecedy.

Neskôr v príkladoch sa budeme zaoberať analyzovaním formálneho jazyka, univerzálnym zápisom pre hľadaný výraz (regulárny výraz). Podrobný výklad je v knihe *Gramatiky a jazyky* [1].

Účely formálneho jazyka v informatike môžeme rozdeliť do 3 kategórií. Každá kategória má iné vlastnosti. Preto sa musíme učiť viacero rôznych jazykov:

- **Dátový jazyk** (*data definition language, data description language*)[11] – jazyk na zápis údajov. Príklady: XML, HTML, SVG, CSV, SQL. Zapisované dáta môžu mať určený dátový typ.

  Dátový typ (*data type*)[12] je spoločná vlastnosť skupiny údajov – jednotná množina hodnôt a jednotná množina operácií – napríklad celé číslo, reálne číslo, písmeno, text, dátum.
- **Komunikačný jazyk**, **protokol** (*protocol*)[13] – súbor pravidiel na komunikáciu medzi programami, počítačmi. Príklady: HTTP, FTP, SMTP, SSH, TCP, UDP, IP, Ethernet, WiFi.
- **Programovací jazyk** (*programming language*)[14] – množina príkazov (kľúčových slov), operandov a operácií, pre zapísanie algoritmu. Príklady: PHP, Python, C, C++, Javascript.



Obrázok 14:
Programovací jazyk, komunikačný jazyk, dátový jazyk

7 Turingov test umelej inteligencie, https://sk.wikipedia.org/wiki/Turingov_test
8 Formálny jazyk, https://sk.wikipedia.org/wiki/Form%C3%A1lny_jazyk
9 Abeceda, https://sk.wikipedia.org/wiki/Abeceda_%28te%C3%B3ria_automatov%29
10 Gramatika, https://sk.wikipedia.org/wiki/Gramatika_%28informatika%29
11 *Data language*, https://en.wikipedia.org/wiki/Data_definition_language
12 Dátový typ, https://sk.wikipedia.org/wiki/D%C3%A1tov%C3%BD_typ
13 Protokol, https://sk.wikipedia.org/wiki/Komunika%C4%8Dn%C3%BD_protokol
14 Programovací jazyk, https://sk.wikipedia.org/wiki/Programovac%C3%AD_jazyk

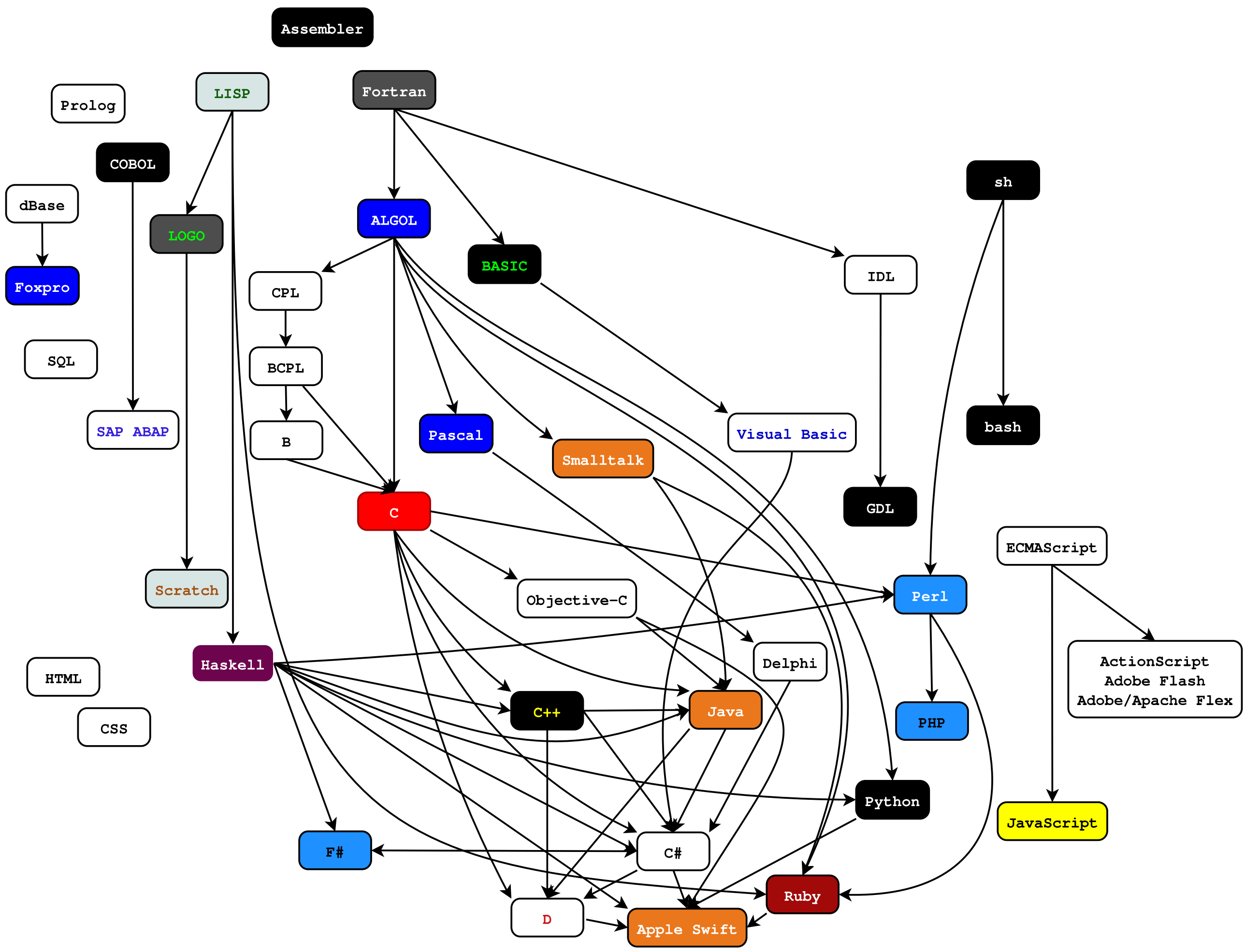
Úložisko http://sourceforge.net sústreďuje veľké množstvo slobodných softvérových projektov. Analyzovalo sa 360 000 z nich v rokoch 2003 – 2012, výsledky sú v článku „*How do OSS projects change in number and size? A large-scale analysis to test a model of project growth*“ [2]. Programátori použili spolu 106 programovacích jazykov. V jazyku C naďalej narastá absolútny počet projektov, aj keď jazyk C stráca percentuálny podiel v prospech skupiny príbuzných jazykov. Aj nás zaujíma táto skupina populárnych jazykov: C, C++, C#, Python, PHP, Java, Javascript.

3.2 História programovacích jazykov C/C++

Najstaršie digitálne počítače v rokoch 1936 – 1955[15] boli programované priamo hardvérovo, alebo v binárnom kóde, alebo v assembleri[16], neskôr vo vyššom programovacom jazyku[17]. Programoval sa počítač ako celok jedným programom. Nerozlišoval sa operačný systém a aplikácia. Od roku 1955 sa programujú aplikácie oddelene od operačného systému. Na počiatku však bola žena:

1842 – 1843	Augusta Ada Byron King, Countess of Lovelace[18], napísala prvý počítačový program pre prvý programovateľný mechanický počítač *Analytical Engine* (Charles Babbage)
1943 – 1951	**Assembler** je prvým programovacím jazykom pre konkrétny elektronický počítač
1953 – 1957	vývoj programovacieho jazyka **Fortran**[19] pre sálový počítač IBM 704
1955 – 1956	**prvý operačný systém** GM-NAA I/O[20] pre sálové počítače IBM 701 – IBM 704
1958	vývoj programovacieho jazyka ALGOL[21]
1966 – 1969	vývoj programovacích jazykov BCPL, B[22] pre projekt MIT Multics
1969 – 1985	operačný systém MIT Multics[23] pre sálové počítače GE, Honeywell
1969 – 1973	vývoj programovacieho jazyka C (*AT&T Bell Labs*, *Ritchie*)[24]
1969	operačný systém Unics pre DEC PDP-7 (*AT&T Bell Labs*)[25] premenovaný na UNIX
1970 – 1971	operačný systém **AT&T UNIX** pre počítač DEC PDP-11 napísaný v assembleri
1972 – 1973	prepísanie kódu operačného systému AT&T UNIX do jazyka C
1978	špecifikácia jazyka C *Kernighan&Ritchie*, neskôr štandardizovaná ako ANSI C
1983	štandardizácia programovacieho jazyka **ANSI C**, vývoj jazyka C++
1983	**GNU**[26] – projekt pre slobodný softvér, neskôr slobodný operačný systém
1988	špecifikácia IEEE **POSIX** (*Portable Operating System Interface for UNIX*)
1989 – 2011	štandardizácia programovacieho jazyka **ANSI/ISO C** (C89, C90, C95, C99, C11)[27]
1998 – 2011	štandardizácia programovacieho jazyka **ISO/IEC C++** (C++98, C++03, C++11)[28]
2008	**štandardizácia UNIX API** podľa ISO/IEC, IEEE POSIX, *The Open Group*
2014	nový komplexný štandard ISO/IEC C++ (C++14)[29]
2017	predpokladaný ďalší štandard ISO/IEC C++ (C++17)[30]

15 História počítačov a operačných systémov, http://sk.wikipedia.org/wiki/Dejiny_počítačov, http://en.wikipedia.org/wiki/History_of_operating_systems#The_mainframe_era
16 Assembler, http://sk.wikipedia.org/wiki/Asembler, http://en.wikipedia.org/wiki/Assembly_language#Assembler
17 História programovacích jazykov, https://en.wikipedia.org/wiki/Timeline_of_programming_languages
18 Augusta Ada Byron King, Countess of Lovelace, http://en.wikipedia.org/wiki/Ada_Lovelace
19 História programovacieho jazyka Fortran, https://en.wikipedia.org/wiki/Fortran#History
20 *General Motors & North American Aviation Input/Output system*, http://en.wikipedia.org/wiki/GM-NAA_I/O
21 História programovacieho jazyka ALGOL, https://en.wikipedia.org/wiki/ALGOL#History
22 História programovacích jazykov BCPL, B, https://en.wikipedia.org/wiki/BCPL, https://en.wikipedia.org/wiki/B_%28programming_language%29
23 Multics, http://en.wikipedia.org/wiki/Multics
24 Programovací jazyk C, http://en.wikipedia.org/wiki/C_(programming_language)
25 Unics, UNIX, http://en.wikipedia.org/wiki/Unix
26 GNU, http://www.gnu.org
27 Štandardizácia ANSI/ISO C, http://en.wikipedia.org/wiki/ANSI_C
28 Štandardizácia ISO/IEC C++, http://en.wikipedia.org/wiki/C%2B%2B
29 Štandard ISO/IEC C++14, https://en.wikipedia.org/wiki/C%2B%2B14
30 Predpokladaný štandard ISO/IEC C++17, https://en.wikipedia.org/wiki/C%2B%2B17



Obrázok 15: Vývoj programovacích jazykov C/C++ a iných súvisiacich jazykov

Assembler je najstarší programovací jazyk, je nízko-úrovňový, hardvérovo závislý od konkrétneho procesora. **Fortran** stojí na začiatku vývoja procedurálnych programovacích jazykov – jazyk, ktorým sa riadi postupnosť krokov v programe. Jazyk **C** je univerzálnym procedurálnym jazykom, použitým na programovanie operačného systému UNIX. V obrázku nižšie v strede je skupina objektových programovacích jazykov. Napravo sú interpretované (skriptovacie) jazyky. Naľavo LISP stojí na začiatku vývoja funkcionálnych jazykov, pričom Scratch je jednoduchý detský programovací jazyk, pre používateľa sa javí ako procedurálny. COBOL a dnes ABAP sú vyššie programovacie jazyky pre programovanie manažérskych aplikácií. Prolog slúži na programovanie výrokovej logiky. Na ľavom okraji sú databázové a webové jazyky. Existuje však veľké množstvo ďalších jazykov, niekoľko stoviek, toľko by sa do obrázku nevošlo...

Programovacie jazyky **C/C++** sú univerzálne, efektívne, štandardizované a moderné aj napriek ich vzniku pred mnohými rokmi, vďaka pokračujúcej štandardizácii nových verzií. Jazyk C je typický procedurálny a funkcionálny jazyk pre systémové programovanie a jazyk C++ je typický objektový jazyk pre programovanie aplikácií. Preto sú tieto jazyky najlepšou voľbou pre začínajúcich technických informatikov. Viaceré novšie populárne programovacie jazyky sú odvodené od jazykov C/C++, použitie novšieho jazyka si vyžiada minimálne úsilie pre toho, kto už pozná jazyky C/C++.

Python, PHP, Javascript sú dnes zrejme najlepšou alternatívou pre všetkých začínajúcich programátorov, ktorým prekáža technická zložitosť C/C++ a hľadajú čo najviac podobný jazyk.

Java je univerzálny programovací jazyk, nezávislý od hardvéru, lebo beží na virtuálnom stroji *Java Virtual Machine*. Problémom je licenčná politika firmy Oracle, ktorá vyústila do viacerých čiastočne nekompatibilných verzií (openJDK, Oracle Java JRE/JDK, IBM JDK, MS C#).

3.3 Modely programovania

Algoritmus, teda postup riešenia úlohy, môže byť zameraný na riadenie postupnosti krokov programu, alebo na riadenie toku dát v programe od vstupov po výstup:

- **riadiaci model** (*control flow*) – algoritmus riadi postupnosť krokov programu, dáta sa prispôsobujú v každom kroku, pri vstupe, pri spracovaní, pri výstupe – väčšina programovacích jazykov je určená pre tento model
- **dátový model** (*data flow*) – algoritmus riadi tok dát a ich spracovanie, postupnosť krokov programu sa prispôsobí dátam, vstup sa očakáva pri potrebe spracovania, výstup je na konci algoritmu, program je obvykle za behu interpretovaný a závisí na konkrétnom vývojovom prostredí (napr. NI LabVIEW[31], Logic Simulator, SQL server)

Algoritmus pre riadiaci model môže obsahovať:

- operátory (konštanty, premenné, procedúry, funkcie, činnosti) so správnymi dátovými typmi
- podmienky – vetvenie (if)
- cykly (for, while, do-while)
- skoky (jump, goto, gosub, call, return)

3.4 Lineárne programovanie

Program má 1 algoritmus, všetky kroky sú usporiadané lineárne za sebou (sekvenčne). **Assembler** je typický lineárny programovací jazyk.

3.5 Štruktúrované programovanie

Program je tvorený hlavným (riadiacim) algoritmom a ďalšími podprogramami, ktoré sú volané z hlavného programu:

- procedurálne f(x) môže obsahovať vstupy, nevracia návratovú hodnotu po vykonaní
- funkcionálne y=f(x) môže obsahovať vstupy, vracia návratovú hodnotu po vykonaní
- objektové program delíme na moduly podľa dát (objekt je dátová abstrakcia)
- aspektové program delíme na moduly naprieč štruktúre funkcií, objektov, dát

V štruktúrovanom programovaní sa nemajú používať globálne premenné a skoky, lebo sa tým kazí štruktúra programu a môžu vzniknúť zbytočné chyby.

Lineárne a procedurálne programovanie je vytváranie algoritmu čisto riadiaceho modelu. Objektové programovanie je kombináciou programovania riadiaceho algoritmu a dátového modelu, preto sa v modernom softvérovom inžinierstve považuje za najlepšiu metodológiu.

3.6 Vizuálne programovanie

- *icon* – grafické skladanie aplikácie z prototypov vo vývojovom prostredí, programovanie je ľahšie, vhodné pre deti[32], začiatočníkov, laikov, alebo programovanie na webe, v mobile[33]
- *canvas/form* – grafické skladanie aplikácie z grafických prvkov vo vývojovom prostredí, s algoritmom definovaným v programovacom jazyku alebo dialógmi vo vývojovom prostredí
- *diagram* – grafické znázornenie algoritmu diagramom (obvykle UML), prekladač preloží diagramy do zdrojového kódu vo vybranom programovacom jazyku, potom preloží do spustiteľného kódu, je to moderné automatizované navrhovanie softvéru

---

31 *National Instruments LabVIEW*, http://www.ni.com/labview/
32 *Scratch*, https://scratch.mit.edu, https://en.wikipedia.org/wiki/Scratch_%28programming_language%29
33 *Google/MIT App Inventor for Android*, http://appinventor.mit.edu/explore/

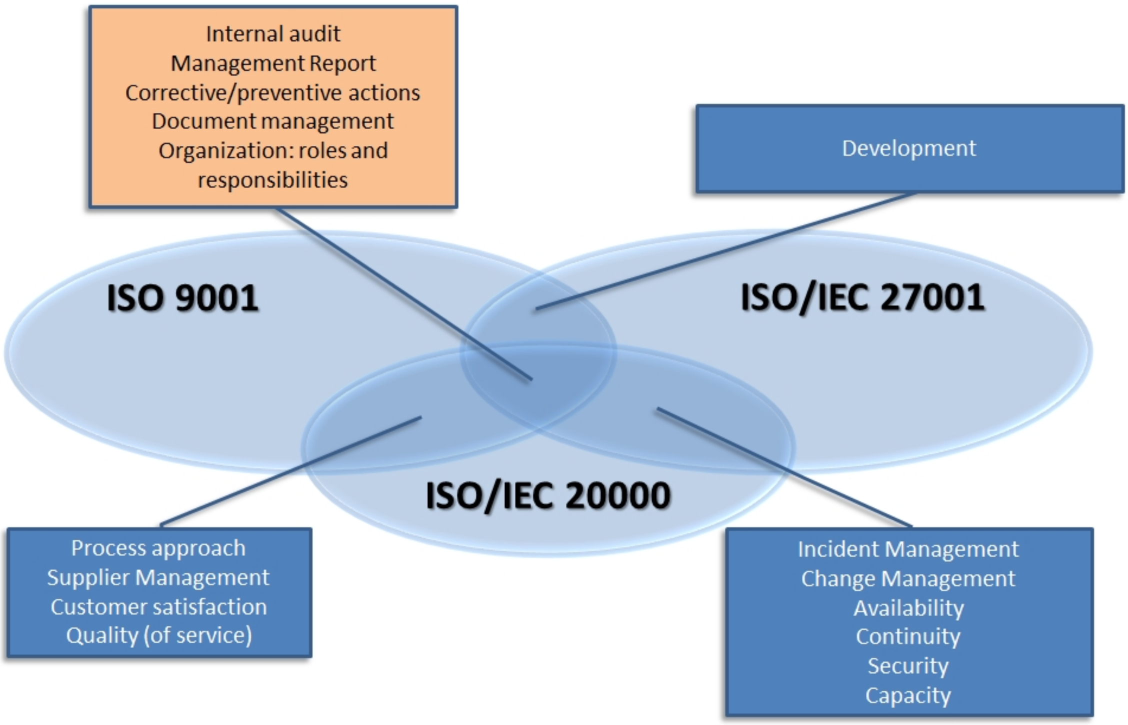
4 Softvérové inžinierstvo a manažment

História programovania sa prejavila aj v slovnom označení práce programátora. Najprv to bolo **programovanie počítača**, pretože programy sa vyrábali pre konkrétny počítač, neboli prenositeľné na iné počítače a neboli použiteľné na iné úlohy. Niektorí programátori boli lepší ako ostatní, vytvorili efektívnejšie algoritmy, krajšie programy. Preto sa práca lepších programátorov neskôr označovala ako **programátorské umenie**. Dnes je programovanie predmetom štúdia od základnej školy až po prax. Sú rozpracované viaceré modely a metódy programovania, vzniklo množstvo programovacích jazykov. Z programovania sa vyvinul odbor **softvérové inžinierstvo**.

Hlavným cieľom softvérového inžinierstva je tvorba kvalitného softvéru. **Kvalita softvéru** je súbor požadovaných vlastností: funkčnosť, jednoduchosť, modularita, univerzálnosť, efektívnosť, stabilita, bezpečnosť, integrita, kompatibilita, štandardizovanosť, čitateľnosť kódu a podobne.

IT firmy by mali fungovať profesionálne, mali by mať profesionálneho manažéra. Každá oblasť práce IT manažéra by mala byť popísaná pravidlami – vnútropodnikovými smernicami. Existuje niekoľko noriem, kde sú tieto pravidlá štandardizované, najmä:

- ISO 9000 (*Quality management systems standards*)[34] – skupina štandardov pre riadenie kvality výroby, služieb, procesov, riadenia firmy
- ISO/IEC 20000 (*International Service Management Standard for IT*)[35] – štandard správy IT
- ISO/IEC 27000 (*Information Security Management Systems standards*)[36] – skupina štandardov popisujúcich riadenie bezpečnosti IT



Obrázok 16: ISO 9001, ISO/IEC 20000, ISO 27001

V obrázku je naznačené, že programovanie (*development*) spadá do prieniku noriem pre riadenie kvality produktov, procesov, riadenia firmy (ISO 9001) a noriem pre riadenie bezpečnosti IT (ISO/IEC 27001). Na základe toho sú vypracované pravidlá pre riadenie IT projektov.

34 ISO 9000, http://sk.wikipedia.org/wiki/ISO_9000, http://en.wikipedia.org/wiki/ISO_9000
35 ISO/IEC 20000, http://en.wikipedia.org/wiki/ISO/IEC_20000
36 ISO/IEC 27000, http://en.wikipedia.org/wiki/ISO/IEC_27000-series, http://www.27000.org, http://www.csirt.gov.sk/informacna-bezpecnost/standardy-a-legislativa/isoiec-27000-814.html

4.1 Riadenie IT projektov

IT projekt vo firme môže byť riadený bez pripravených pravidiel, čo však prináša viacero rizík, napríklad nepredvídané situácie, preťaženie pracovníkov, vymýšľanie vymysleného.

Pracovné pozície v IT projektoch môžeme v zásade rozdeliť do 3 úrovní:

- manažér
- analytik, architekt
- programátor, kóder, tester

Manažér nemusí byť špecialista v IT. Podstatné je, že má všeobecný prehľad v IT a detailne pozná možnosti, skúsenosti svojho tímu, svojej firmy. Analytik, architekt – to je špecialista s VŠ vzdelaním, musí byť dobrý programátor, tímový hráč, s dobrými komunikačnými schopnosťami. Programátor, kóder – je špecialista vo svojej oblasti, v konkrétnom programovacom jazyku alebo vývojovom prostredí, vzdelanie nie je podstatné. Tester je špecialista na testovanie softvéru. Požiadavky na testera sú podobné ako na programátora. Požadovaná prax v pracovných inzerátoch pre analytika, architekta je najčastejšie do 3 rokov. Požadovaná prax pre programátora je rôzna, často od 0 rokov. Dobrý programátor je mladý programátor, pretože je rýchly, ľahko sa učí, vydrží veľa pracovať a je lacnejší v porovnaní so staršími, skúsenejšími IT špecialistami. [3] Je vhodné využiť osvedčené štandardy alebo metodiky manažérstva pre riadenie projektov:

PRINCE2 (*PRojects IN Controlled Environment*)[37] – metodika projektového riadenia, zahŕňajúca zdôvodnenie projektu, zadefinovanie rolí, zodpovedností, právomocí, ďalej firemné procesy, témy, techniky, etapy, produktové plánovanie.

IPMA (*International Project Management Association*)[38] – organizácia pre projektové riadenie.

PMI (*Project Management Institute*)[39] – organizácia pre štandardy projektového riadenia.

APM (*Association for Project Management*)[40] – organizácia pre projektové riadenie.

PMA (*Project Management Advisor*)[41] – poradca pre riadenie projektov.

Lean management[42] – metodika riadenia výroby pomocou obmedzenia plytvania zdrojmi a časom.

Lean Six Sigma[43] – metodika riadenia obmedzovaním strát (čas, zásoby, doprava, čakanie, nadprodukcia, spracovanie, nepodarky, zručnosti a schopnosti).

Kanban[44] – metodika riadenia výroby pomocou obmedzenia prestojov a skladov (Toyota).

Agile programming[45] – metodika riadenia softvérových projektov v tímoch s adaptívnym plánovaním s cieľom rýchlej dodávky softvérového produktu zákazníkovi, najmä s objektovým návrhom.

Scrum[46] – agilná metodika riadenia softvérových produktov s presne zadefinovanými rolami.

DevOps[47] (*Development and Operations*) – (pôvodne agilná) metodika automatizovanej komunikácie a spolupráce medzi vývojármi a ostatnými IT špecialistami, vrátane testovania kvality (QA).

Agilná metodika vychádza z kritiky ťažkopádnej hierarchickej organizačnej štruktúry riadenia (*top-down waterfall*), kde manažéri bez dostatočných IT znalostí rozhodujú o projekte, pričom

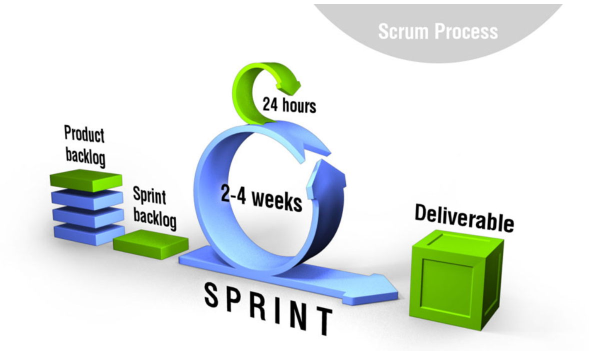
37 PRINCE2, http://sk.wikipedia.org/wiki/PRINCE2, http://prince-2.sk/index.php
38 IPMA, http://en.wikipedia.org/wiki/International_Project_Management_Association, http://ipma.ch
39 PMI, http://www.pmi.sk, http://www.pmi.org
40 APM, http://en.wikipedia.org/wiki/Association_for_Project_Management, http://www.apm.org.uk
41 PMA, http://www.pma.doit.wisc.edu
42 *Lean management*, https://en.wikipedia.org/wiki/Lean_manufacturing
43 *Lean Six Sigma*, https://en.wikipedia.org/wiki/Lean_Six_Sigma
44 *Kanban*, https://en.wikipedia.org/wiki/Kanban
45 *Agile software development*, https://en.wikipedia.org/wiki/Agile_software_development
46 *Scrum*, https://en.wikipedia.org/wiki/Scrum_%28software_development%29
47 *DevOps*, https://en.wikipedia.org/wiki/DevOps

programátori realizujú projekt bez zasahovania do riadenia projektu. To vedie k prekročeniu termínu, prekročeniu rozpočtu, únave z dlhej práce, nenaplneniu očakávaní zákazníka.

Agilná metodika delí projekt na kratšie časové úseky a na časti v zodpovednosti jednotlivých programátorov, uprednostňuje rýchlejšie dodanie produktu a častejšiu komunikáciu so zákazníkom. Konkrétna interpretácia agilnej metodiky môže byť rôzna. *Scrum* formalizuje agilnú metodiku striktne. Základné hodnotové princípy agilnej metodiky sú formulované ako ***Agile manifesto***[48]:

- jednotlivci a vzťahy v programátorskom tíme sú dôležitejšie ako procesy a nástroje
- funkčný softvérový produkt je dôležitejší ako úplná dokumentácia
- spolupráca so zákazníkom je dôležitejšia ako špecifikácia od zákazníka na začiatku projektu
- odozva na zmenu požiadaviek je dôležitejšia ako naplnenie plánu

Agilná metodika je vhodná pre tímy začínajúcich mladých programátorov, menej pre skúsených programátorov, lebo predpokladá vyrovnané zručnosti a schopnosti členov tímu. Je vhodná na zvládnutie dočasných krízových situácií, lebo uprednostňuje rýchlu reakciu, spoluprácu a potláča individuálne odlišnosti členov tímu. Nie je vhodná pre dlhodobé fungovanie skúsených programátorov, takým môže znižovať výkonnosť. Nedostatky agilnej metodiky a špeciálne metodiky *Scrum*: atomizovanie projektu na príliš malé časti, programátori strácajú pohľad na celok, riadenie projektu hrané ako divadlo, strata času na zbytočných a príliš častých stretnutiach tímu (až denne), frustrácia z nekončiaceho sa kolobehu termínov (obyčajne 2-týždňový cyklus vývoja), vyhorení programátori. Ak členovia tímu nemajú dostatočne dobré komunikačné zručnosti, začnú si prekážať a nevedia spolupracovať. Agilná metodika vyžaduje od tímu, aby všetci členovia tímu boli viditeľní a videní okolím, čo u časti tvorivých ľudí znižuje výkon (podobne ako otvorené kancelárske priestory).



Obrázok 17: Scrum

Metodika *Scrum* formalizuje roly v tíme. Vlastník produktu (*product owner*) je manažérska pozícia reprezentujúca požiadavky zákazníka, definuje prípady použitia produktu z pohľadu používateľa (*user stories*). *Poznámka: spôsob formulovania požiadaviek sa dobre zapisuje pomocou jazyka UML a vedie k objektovému návrhu, čo je vysvetlené v neskoršej kapitole.* Šéf tímu (*Scrum master*) je manažérsko-technická pozícia. Šéf tímu je pre členov tímu kouč, nie nadriadený. Šéf tímu je sprostredkovateľom technickej komunikácie medzi tímom a zákazníkom, nie projektový šéf. Je to manažérske divadlo? Členovia tímu si sami riadia projekt – programátori sú nútení robiť manažérsku prácu namiesto manažérov. Metodika *Scrum* je sociálny model nefunkčnej organizácie s nejasne definovanou organizačnou štruktúrou a nanútenou rovnosťou v tíme za cenu poklesu priemernej úrovne členov tímu (to pripomína komunizmus). Názvy rolí v tíme musíme chápať obrazne, nie doslovne, lebo to by odporovalo legislatívnym definíciám podľa autorského zákona.[49]

Či už sa rozhodneme pre tú alebo onú metodiku riadenia IT projektov, vyberme si to, čo je dobré a vyhnime sa tomu, čo je zlé. Agilná metodika pre študentské projekty môže mať význam na vysokej škole v niektorých situáciách, kde sa predpokladá, že študenti sú samostatní. Pri mladších

48 *Agile manifesto*, http://agilemanifesto.org, https://en.wikipedia.org/wiki/Agile_Manifesto
49 Autorský zákon, zákon 185/2015 Z.z., https://www.slov-lex.sk/pravne-predpisy/SK/ZZ/2015/185/20160101

študentoch je efektivita agilnej metodiky slabšia, lebo zodpovednosť za vzdelávanie je prenesené z učiteľa na študenta a metodika je ťažká na zvládnutie.[50]

Všeobecný postup v riadení softvérových projektov je daný obvyklými činnosťami, ktoré sa vykonávajú jednorázovo, alebo aj opakovane v prípade ďalších verzií softvéru:

- ⇩ analýza problému u objednávateľa – špecifikácia požiadaviek
- ⇩ programátorská analýza problému
- ⇩ návrh algoritmu, diagramy, popis prvkov
- ⇩ výber modelu programovania a metódy programovania
- ⇩ výber programovacieho jazyka, operačného systému, hardvéru, licenčných podmienok
- ⇩ implementácia v programovacom jazyku a súčasne komentáre, technická dokumentácia
- ⇩ testovanie
- ⇩ dokončenie dokumentácie (technická dokumentácia, používateľská príručka, licencia)
- ⇩ nasadenie do prevádzky
- ⇩ servis
- ⇦ opakovanie cyklu pre novú verziu

Na základe zvolenej projektovej metódy si navrhneme časový plán a ekonomickú kalkuláciu.

4.2 Prototyp

Pre názornú predstavu vývojárov a zákazníka je dobré urobiť prototyp aplikácie na začiatku vývoja aplikácie. Prototyp má mať želaný vzhľad a základné funkcie pre používateľa. Vytvorenie prototypu by malo byť rýchle a ľahké. Preto sa prototyp robí obyčajne ako jednoduchá webová aplikácia pomocou jazykov HTML, CSS, Javascript.

4.3 Verzia softvéru

Vývoj softvéru je málokedy jednorázovou činnosťou. Aby sa udržal poriadok v nasledujúcich etapách vývoja softvéru, číslujú sa jeho verzie. Ak sa už dodaný/zverejnený softvér neskôr zmení, tak sa vyrobí nová verzia. Verzia je číselné alebo slovné označenie. Zaužívané označenia sú:

alfa	vývojová etapa
beta	testovacia etapa
0.x	vývojová/beta verzia s podverziami
1.x	prvá finálna verzia s podverziami

Verzionovací systém spravuje verzie súboru, balíka, projektu. Je vhodný pre prácu v tíme:

CVS[51]	*concurrent version system*
SVN[52]	*Apache Subversion*
git[53]	*version server (github)*
mercurial[54]	distribuovaný verzionovací systém
Gource[55]	vizualizácia verzií softvéru
code_swarm[56]	vizualizácia verzií softvéru (animácia verzií zdrojového kódu v čase)
DejaCode[57]	manažment použitých licencií v zdrojovom kóde

50 *eduScrum*, http://agile.sk/wp-content/uploads/2014/12/The-eduScrum-Guide-SK-December-2013-version-1.1.pdf
51 CVS, https://en.wikipedia.org/wiki/Concurrent_Versions_System, http://www.nongnu.org/cvs/
52 SVN, https://en.wikipedia.org/wiki/Apache_Subversion, https://subversion.apache.org
53 *git*, https://git-scm.com, https://sk.wikipedia.org/wiki/Git_%28softv%C3%A9r%29, https://en.wikipedia.org/wiki/Git_%28software%29
54 *mercurial*, https://www.mercurial-scm.org, https://en.wikipedia.org/wiki/Mercurial
55 *Gource*, http://gource.io
56 *code_swarm*, http://vis.cs.ucdavis.edu/~ogawa/codeswarm/
57 *DejaCode*, http://www.dejacode.com

4.4 Programátorské nástroje

Programátor potrebuje textový editor na písanie zdrojového kódu a rôzne ďalšie nástroje, ktoré môžu byť samostatné, alebo združené do vývojového balíka. Používajú sa tri označenia:

- ✔ **CASE** (*Computer Aided Software Engineering*)
- ✔ **SDK** (*Software Development Kit*)
- ✔ **IDE** (*Integrated Development Environment*)

Vývojový balík obsahuje tieto súčasti:

- *editor* textový editor, farebne rozlišuje syntax jazyka, dopĺňa príkazy, čísluje riadky
- *manual* príručka programovacieho jazyka
- *preprocessor* spracuje zdrojový kód, pripraví pre preklad, testuje a simuluje beh programu
- *includes* knižnice API (.h) (*header files*)
- *libs* knižnice API (.o, .so, .obj, .a, .dll) (*compiled object files*)
- *compiler* prekladač zdrojového kódu do spustiteľného strojového kódu (vytvorí objekt)
- *linker* spájač objektov do celkového spustiteľného programu
- *assembler* nízkoúrovňový, hardvérovo závislý prekladač do strojového kódu
- *loader* spúšťač preloženého spustiteľného kódu, obyčajne súčasť operačného systému
- *interpreter* vykonáva interpretovaný program
- *debugger* testuje, krokuje, hľadá chyby, ukladá a zobrazuje stav pamäte
- *utilities* ďalšie pomocné programy

Programátorské nástroje pre jazyky C/C++:

cc, gcc	*C Compiler, GNU C Compiler* (UNIX/Linux)
c++, g++	*C++ compiler, GNU C++ compiler* (UNIX/Linux)
make	*GNU make utility for compiling* (UNIX/Linux)
mingw	*minimalist GNU for Windows* (gcc & SDK)
ld	*GNU linker*
gdb	*GNU DeBugger*
asm	*assembler*

Prehľad nástrojov a jazykov GNU je v knihe *Programovací jazyky GNU* [4].

4.5 Preklad a interpretácia

Zdrojový kód sa transformuje na program prekladom alebo interpretovaním. Preto potrebujeme prekladač (*compiler*) alebo interpreter:

- **zdrojový kód preložený** do spustiteľného kódu (*executable .exe*)
- **zdrojový kód interpretovaný** priamo (*script .sh, batch .bat*)

Preložený zdrojový kód do spustiteľného kódu je spustený ako samostatný proces. Interpretovaný program beží pod interpreterom, interpreter je spustený ako proces.

source code → *compiler* → *object* → *linker* → *executable program* → *loader* → *process*

source code → *interpreter (process)* → *program*

Programovacie jazyky C a C++ sú typické prekladané jazyky. Ale to neznamená, že nemôže existovať interpreter pre takýto jazyk. Interpreter je vhodný pre začiatočníkov, lebo zjednodušuje vývojové prostredie, programátor nie je zaťažovaný technickými detailami. Interpreter je vhodný aj pre použitie jazyka tam, kde nie je podstatná optimalizácia a rýchlosť vykonávaného programu.

CINT (*C INTerpreter*)[58] – interpreter programovacích jazykov C a C++, naprogramovaný v jazyku C++, je súčasťou softvérového balíka ROOT, používaného vo vede a výskume (CERN).

58 CINT, https://root.cern.ch/cint

4.6 Licencia

Program je autorské dielo a na jeho použitie existujú predpisy. Programovanie je zárobková činnosť programátora. Programátor, teda autor, rozhoduje o spôsobe použitia svojho programu.

Licencia daná licenčnou zmluvou je právo od autora diela pre používateľa diela na použitie diela. Licencie sú definované v autorskom zákone[59] a konkrétne znenie je formulované autorom, alebo organizáciou zastupujúcou autorov. Bežné typy softvérov a softvérových licencií sú tieto:

- ✔ **Anonymný softvér (*public domain*)** – autor je neznámy, voľné kopírovanie a používanie.
- ✔ **Bezplatný softvér (*freeware*)** – voľné a bezplatné kopírovanie a používanie.
- ✔ **Otvorený zdrojový kód (*open source*)** – voľné kopírovanie a editovanie zdrojového kódu.
- ✔ **Slobodný softvér (*free software = freeware & open source*)**
- ✔ **Príspevkový softvér (*shareware*)** – najprv vyskúšaj, potom prispej peniazmi a používaj.
- ✔ **Demonštračná-ukážková verzia (*demo version*)** – má obmedzené funkcie na voľné použitie.
- ✔ **Skúšobná verzia (*trial version*)** – verzia obmedzená obvykle počtom dní používania.
- ✔ **Platený softvér (*proprietary software*)** – platené použitie je definované v licenčnej zmluve.
- ✔ **EULA**[60] (*End User License Agreement*) – licenčná zmluva s koncovým používateľom.
- ✔ **OEM**[61] (*Original Equipment [from] Manufacturer*) – obchodný vzťah výrobcu komponentu a výrobcu finálneho produktu zloženého z komponentov.

Princípy slobody pre slobodný softvér podľa *Free Software Foundation*[62] (*Richard Stallman*):

0. právo spúšťať softvér na akýkoľvek účel
1. prístupnosť otvoreného zdrojového kódu – právo študovať kód, modifikovať, prispôsobovať
2. právo kopírovať softvér a šíriť kópie softvéru
3. právo zverejniť a šíriť modifikovaný softvér

Podľa týchto princípov boli vytvorené viaceré verejné licencie pre softvérové autorské diela:

- ✔ **GNU GPL**[63] (*GNU General Public License*) – štandardizovaná verejná licencia pre softvér; odvodený softvér musí zachovať pôvodnú licenciu.
- ✔ **GNU LGPL**[64] (*GNU Lesser General Public License*) – voľnejšia licencia, umožňuje spojenie otvoreného a uzavretého zdrojového kódu pomocou knižnice.
- ✔ ***Apache License***[65] – čiastočne kompatibilná licencia s GNU GPL.
- ✔ ***BSD License***[66] – stručná a málo reštriktívna verejná licencia, pôvodne pre BSD UNIX.
- ✔ ***MIT License***[67] – stručná a málo reštriktívna verejná licencia.

---

59 Autorský zákon: zákon 185/2015 Z.z., https://www.slov-lex.sk/pravne-predpisy/SK/ZZ/2015/185/20160101
60 EULA, http://en.wikipedia.org/wiki/End-user_license_agreement
61 OEM, http://en.wikipedia.org/wiki/Original_equipment_manufacturer
62 Slobodný softvér, http://www.gnu.org/philosophy/free-sw.html, https://en.wikipedia.org/wiki/The_Free_Software_Definition, https://en.wikipedia.org/wiki/Free_software, http://fsfe.org/about/basics/freesoftware.html, http://www.fsf.org/about/what-is-free-software
63 GNU GPL v. 3, http://www.gpl.sk, http://www.gnu.org/copyleft/gpl.html
64 GNU LGPL v. 3, http://www.gnu.org/licenses/lgpl-3.0.html,
65 *Apache License v. 2*, http://www.apache.org/licenses/LICENSE-2.0, https://en.wikipedia.org/wiki/Apache_License
66 *BSD License*, https://en.wikipedia.org/wiki/BSD_licenses, http://opensource.org/licenses/BSD-2-Clause, http://opensource.org/licenses/BSD-3-Clause, https://www.freebsd.org/copyright/freebsd-license.html
67 *MIT License*, https://en.wikipedia.org/wiki/MIT_License, http://opensource.org/licenses/MIT

OSI (*Open Source Initiative*)[68] – nezisková organizácia propagujúca myšlienku tvorby softvéru s otvoreným zdrojovým kódom. Zaužívané skratky pre softvér s otvoreným zdrojovým kódom:

OSS (*Open Source Software*) – otvorený softvér
FOSS (*Free and Open Source Software*) – slobodný softvér (bezplatný a otvorený)
FLOSS (*Free/Libre and Open Source Software*) – bezplatný, slobodný, otvorený

Softvér je špeciálnym prípadom autorského diela podľa autorského zákona. Čo pre iné diela?

- ✔ **CC**[69] (*Creative Commons*) – štandardizovaná verejná licencia vyhradzujúca pre autora niektoré autorské práva, ostatné práva necháva voľné pre používateľa za predpokladu, že nezasahujú do práv autora.
- ✔ **GNU FDL**[70] (*GNU Free Documentation License*, *GFDL*) – štandardizovaná verejná licencia pre literárne diela s podobnými pravidlami ako CC-BY-SA a GNU GPL.

Wikipedia používa dvojitú licenciu CC-BY-SA a GFDL.[71] Písanie študentských prác, učebných textov a iných literárnych diel s použitím Wikipedie má prísne pravidlá. Citácia je korektné uvedenie zdroja, pričom prevezme sa len malá časť textu zo zdroja, napríklad definícia, veta, krátky odsek. Odvodené dielo obsahuje dlhší prevzatý text, napríklad odsek, stranu, celý dokument. Dôležité je pozrieť sa na licenciu zdroja. Licencie CC-BY-SA, GNU FDL, GNU GPL vyžadujú, aby odvodené dielo malo rovnakú licenciu, autor odvodeného diela ju musí uviesť.

Charta práv pre WWW – návrh základných princípov (Tim Berners-Lee, 2014)[72]:

Slobody pre používateľské dáta:

1. právo rozhodovať o svojom súkromí
2. prenositeľnosť dát
3. právo rozhodovať o tvorbe a editovaní

Slobody pre softvérové produkty a služby:

4. sloboda pre softvér – slobodný softvér
5. právo podieľať sa na vývoji softvéru aj pre ostatných
6. samostatnosť používateľa

Slobody verejnej siete:

7. univerzálny prístup k sieti, k sieťovým službám
8. sloboda informácií
9. sieťová neutralita
10. otvorenosť sieťovej infraštruktúry pre všetkých

68 OSI, https://en.wikipedia.org/wiki/Open_Source_Initiative, http://opensource.org/
69 CC (*Creative Commons*) v. 4, http://sk.creativecommons.org
70 GFDL (*GNU Free Documentation License*), https://sk.wikipedia.org/wiki/GNU_Free_Documentation_License, https://en.wikipedia.org/wiki/GNU_Free_Documentation_License, http://www.gnu.org/licenses/#FDL
71 *Wikipedia license*, https://en.wikipedia.org/wiki/Wikipedia:GFDL, https://en.wikipedia.org/wiki/GNU_Free_Documentation_License
72 *Digital rights and freedoms: A framework for surveying users and analyzing policies*, http://arxiv.org/abs/1410.2379, http://arxiv.org/pdf/1410.2379v2.pdf

Zaužívané označenia pre použitie autorského práva, patentového práva a obchodných značiek:

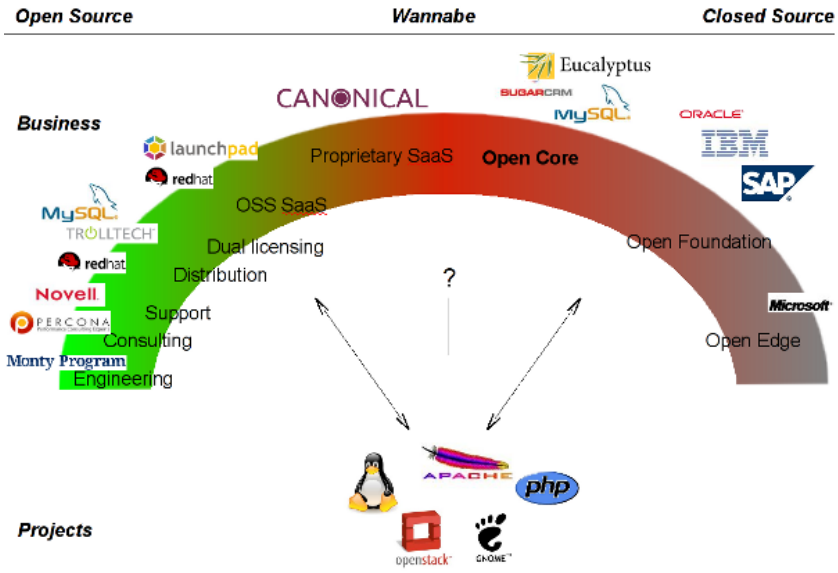
- ✔ ™ (*Trade Mark*) – obchodná značka.
- ✔ ® (*Registered trademark*) – registrovaná obchodná značka, má právnu ochranu.
- ✔ © (*Copyright*) – autorské právo rozhodovať o použití svojho diela (aj o kopírovaní).

Kedy sa použije autorské právo a kedy patentové právo?

- ✔ **Autorské právo (*copyright*)** – pravidlá pre ochranu literárneho diela, umeleckého diela, zdrojového kódu v programovacom jazyku, vyjadrenia myšlienky a pravidlá pre licencie na použitie diela. Licencie, ktoré chránia autora len v malom rozsahu a umožňujú voľné použitie diela, sa zvyknú označovať ako ***copyleft***, napr. CC, GNU FDL, GNU GPL.
- ✔ **Patentové právo (*patent law*)** – pravidlá pre ochranu myšlienky, inovácie.

Softvéroví piráti obyčajne útočia na platené licencie, nemali by útočiť celkovo na autorské právo. Patentové právo sa zvykne zneužívať právnymi spormi na obmedzovanie konkurencie a inovácií. Preto je lepšie obmedziť možnosť patentovať softvérové riešenia (softvérové patenty). Myšlienka sa dá vyjadriť aj inými prostriedkami. Komunikačný jazyk, programovací jazyk nie je možné chrániť ani autorským ani patentovým právom, lebo ide len o prostriedok vyjadrenia myšlienky, čiže reč.

Aké sú možnosti podnikania pri softvéri s otvoreným zdrojovým kódom? Rôzne typy licencií umožňujú široké spektrum podmienok používania softvéru. Každá firma si môže zvoliť svoj model podnikania.[73]



Obrázok 18: Business model

73 *Open source software business models,* https://en.wikipedia.org/wiki/Business_models_for_open-source_software

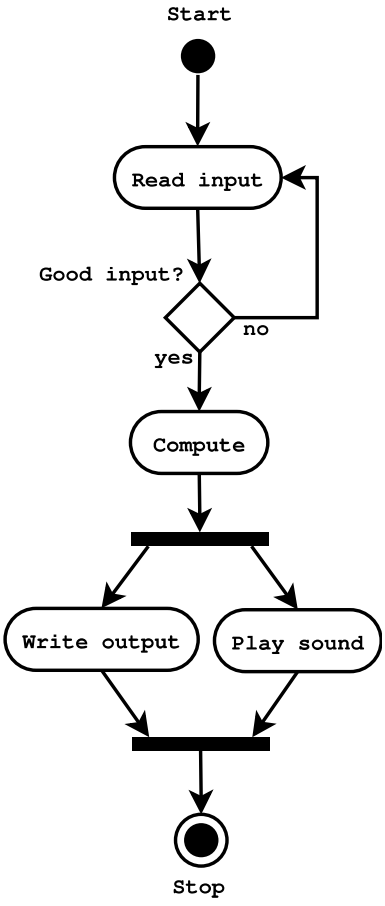
5 Algoritmizácia

5.1 UML

Modelovanie, kompletný grafický popis algoritmu, je súčasťou činností programátora. Na tento účel sa používa jazyk UML 2.5 (*Unified Modeling Language*)[74] – jednotný modelovací jazyk. Algoritmus sa opíše diagramami z pohľadu skladby programu aj z pohľadu behu programu:

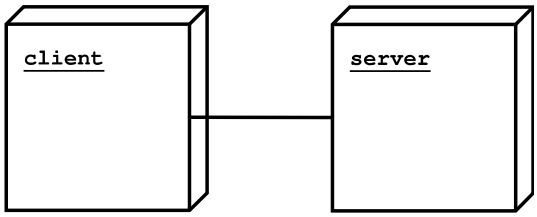
- **statický pohľad** (štruktúra programu, programové komponenty):
  - blokové diagramy (rozmiestnenie, balíčky, komponenty, zložená štruktúra)
  - diagramy tried a objektov
- **dynamický pohľad** (správanie sa programu v čase):
  - diagram prípadov použitia
  - diagram aktivity (diagram riadenia, diagram toku dát)
  - stavový diagram
  - diagramy interakcie (interakcie, sekvencie, časovanie, komunikácia)

Základné typy UML diagramov si názorne ukážeme na príkladoch. Prvé dva sú zvýraznené žltým.



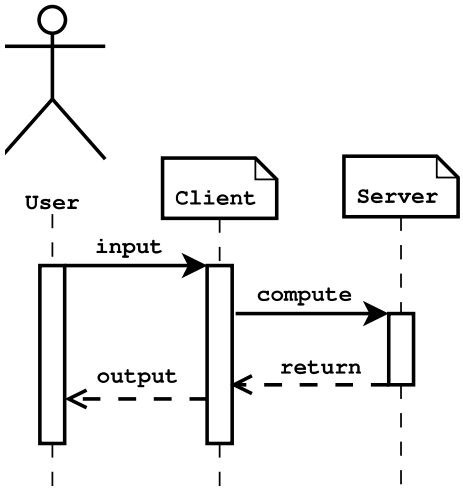
Obrázok 20: Diagram aktivity (activity diagram)

Na začiatok postačí dvojica jednoduchých diagramov, jeden pre dynamický, druhý pre statický pohľad. Spolu tvoria popis jedného programu, preto musia súvisieť. Ak súvis nie je zrejmý, doplníme ďalší typ diagramu. Zatiaľ nie je jasné, ako súvisí diagram aktivity s blokovým diagramom – ktorá činnosť sa vykonáva v ktorom bloku.



Obrázok 19: Blokový diagram: diagram rozmiestnenia (deployment diagram)

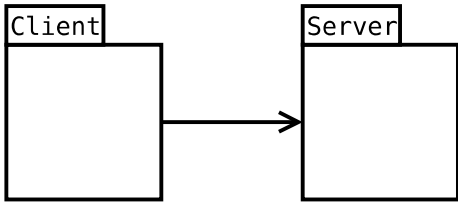
Pridaný tretí diagram (sekvenčný) vysvetľuje postupnosť činností, aj rozmiestnenie činností do blokov. Čas plynie zhora nadol. Bloky programu sú v stĺpcoch vedľa seba, vrátane používateľa.



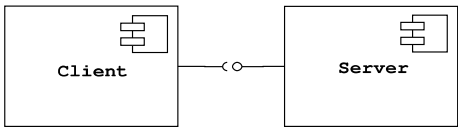
Obrázok 21: Sekvenčný diagram (sequence diagram)

74 UML, http://sk.wikipedia.org/wiki/Unified_Modeling_Language, http://en.wikipedia.org/wiki/Unified_Modeling_Language, http://www.uml.org, https://sites.google.com/site/valasek/oss/uml-diagramy

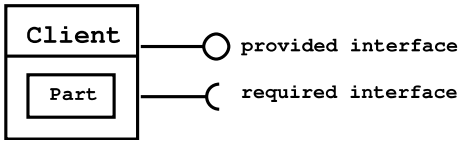
Objektový návrh programu vyžaduje doplniť ďalšie typy diagramov pre statický pohľad (objektové balíčky, komponenty, triedy, objekty):



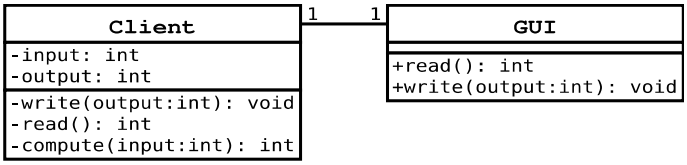
Obrázok 22: Diagram balíčkov (package diagram)



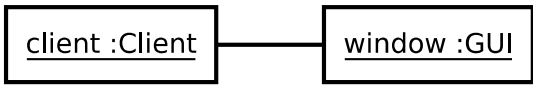
Obrázok 23: Diagram komponentov (component diagram)



Obrázok 24: Diagram zloženej štruktúry (composite structure diagram)

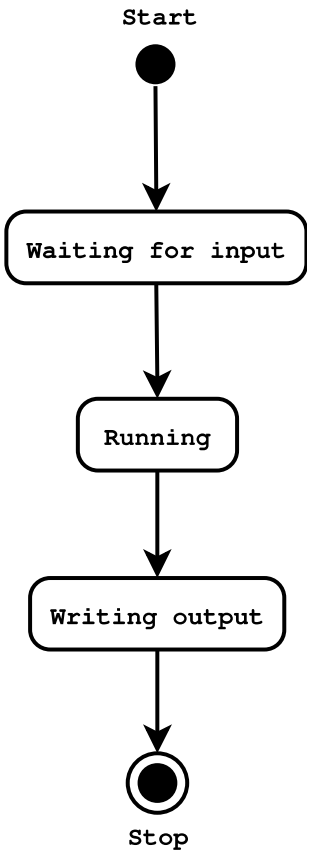


Obrázok 25: Diagram tried (class diagram)

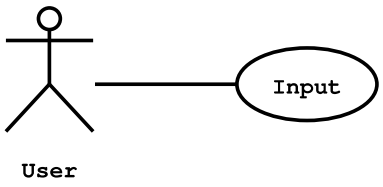


Obrázok 26: Diagram objektov (object diagram)

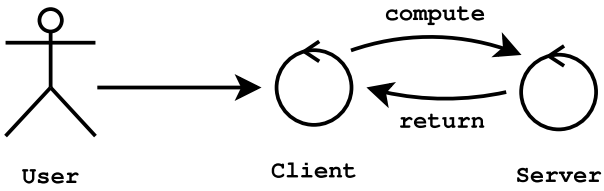
Zvyšné typy diagramov pre dynamický pohľad:



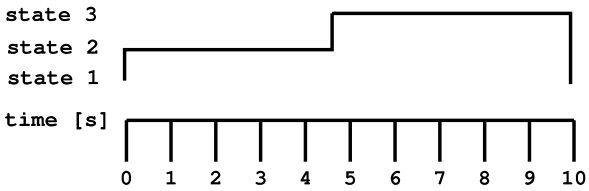
Obrázok 27: Stavový diagram (state diagram)



Obrázok 28: Diagram prípadov použitia (use-case diagram)



Obrázok 29: Diagram komunikácie (communications diagram)



Obrázok 30: Diagram časovania (timing diagram)

5.2 Nástroje pre kreslenie UML diagramov

Existuje veľa nástrojov pre UML jazyk. Niektoré vedia kresliť diagramy, niektoré generovať zdrojový kód z diagramov a niektoré generovať diagramy zo zdrojového kódu.[75] Napríklad:

- Dia (+ dia2code, cpp2dia)
- Modelio
- UMLet
- BOUML
- Rational Rose (IBM)
- Visual Pardigm for UML
- MS Visio

Uvedené diagramy boli nakreslené v slobodnom programe Dia.

75 UML nástroje, https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

6 Vývojové prostredie pre C/C++

Programovať sa dá v princípe na hocijakom počítači. Veľmi záleží na operačnom systéme, lebo ten poskytuje základné systémové nástroje pre programovanie.

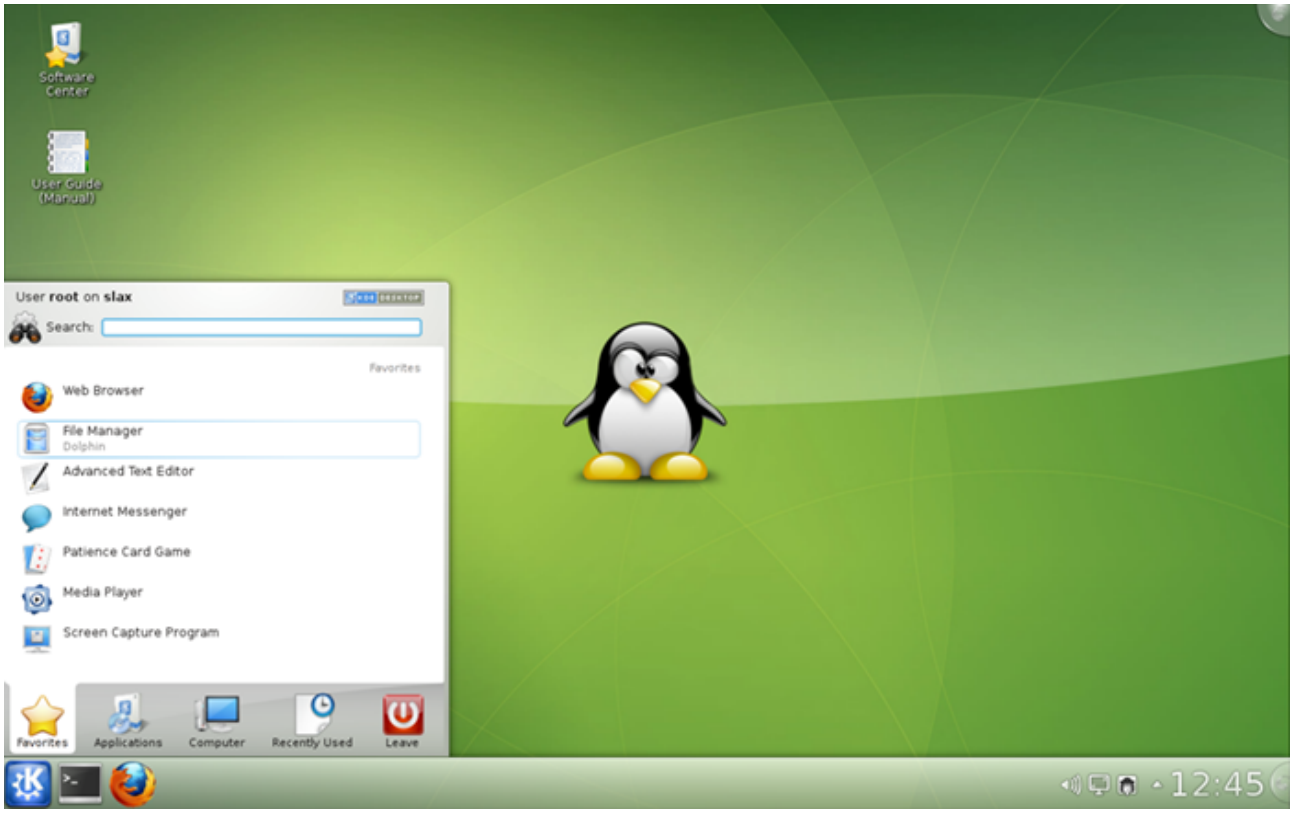
Operačný systém MS Windows nie je vhodný na univerzálne programovanie, nemá potrebné systémové nástroje a knižnice pre jazyky C/C++. Je nutné inštalovať niektorý balík s vývojovým prostredím a systémovými nástrojmi. Integrácia viacerých vývojových prostredí a knižníc, najmä grafických knižníc, je problematická. Programovanie aplikácií pre MS Windows 8/10 s prostredím Metro/ModernUI má špecifický postup, lebo GUI pre aplikáciu je programovaný v jazyku HTML 5, oddelene od algoritmu.

6.1 Operačný systém GNU/Linux

Tento operačný systém má viacero výhod pre programátora: je slobodný, modulárny, spoľahlivý, bezpečný, efektívny, funguje aj na staršom hardvéri, má dostupné veľké množstvo nástrojov a knižníc pre programovanie vo viacerých jazykoch. Integrácia systémových nástrojov, viacerých vývojových prostredí a knižníc je pri vhodných distribúciách bezproblémová.

Programovanie v operačnom systéme GNU/Linux si vyžaduje základné zručnosti pri práci s týmto systémom – správa používateľov, práca so súbormi a oprávneniami, správa procesov.

Slax (`http://www.slax.org`) je distribúcia malá, jednoduchá, pripravená pre spúšťanie zo živého média. Slax 7.0.8 ISO 218 MB (MiB) je pre napálenie na CD, alebo pre pripojenie vo virtuálnom stroji ako virtuálne CD. Súbor ZIP 218 MB (MiB) je určený pre rozbalenie na USB disk. Už obsahuje všetko potrebné pre programovanie zhruba polovice príkladov v tomto učebnom texte. Postačí 512 MB (MiB) RAM a pevný disk žiadny. Chýbajúce balíky sa dajú doinštalovať.



Obrázok 31: Slax

Výber distribúcií GNU/Linux je široký. Každý si nájde takú, ktorá mu bude vyhovovať.

Fungovanie operačných systémov a práca s operačným systémom GNU/Linux je podrobne popísaná v samostatnom učebnom texte *Operačné systémy (GNU/Linux)*.[76]

Niekoľko potrebných príkazov pre prácu so súbormi v príkazovom riadku:

man *<text>*	zobraz pomoc
touch *<file>*	vytvor prázdny súbor
> *<file>*	vytvor prázdny súbor alebo prepíš existujúci
ls	*list* – vypíš obsah aktuálneho priečinka
ls *<dir>*	vypíš obsah daného priečinka (názvy súborov)
cd *<dir>*	*change directory* – zmeň aktuálny priečinok
cp *<file1> <file2>*	*copy file* – kopíruj súbor 1 na súbor 2
rm *<file>*	*remove file* – zmaž súbor
cat *<file>*	vypíš textový súbor na obrazovku celý
more *<file>*	vypíš textový súbor na obrazovku po stranách (fungujú klávesy ENTER, SPACE, B, Q)

GNU/Linux má viacero textových editorov pre spracovanie textu:

ed, sed, vi, vim, nano, pico, joe, emacs, gedit, xedit…

Alebo editor programu *Midnight commander*[77] (Slax ho má nainštalovaný, Ubuntu nie):

mc

Budeme potrebovať nastavenie oprávnení na súbory:

chmod *<options> <file>*	*change file mode bits* – zmeň oprávnenia na súbor:
chmod u+x *<file>*	pridaj oprávnenie na spustenie pre vlastníka (*user*)
chmod 750 *<file>*	oprávnenia budú: **rwxr-x---** (číselne v bitoch)

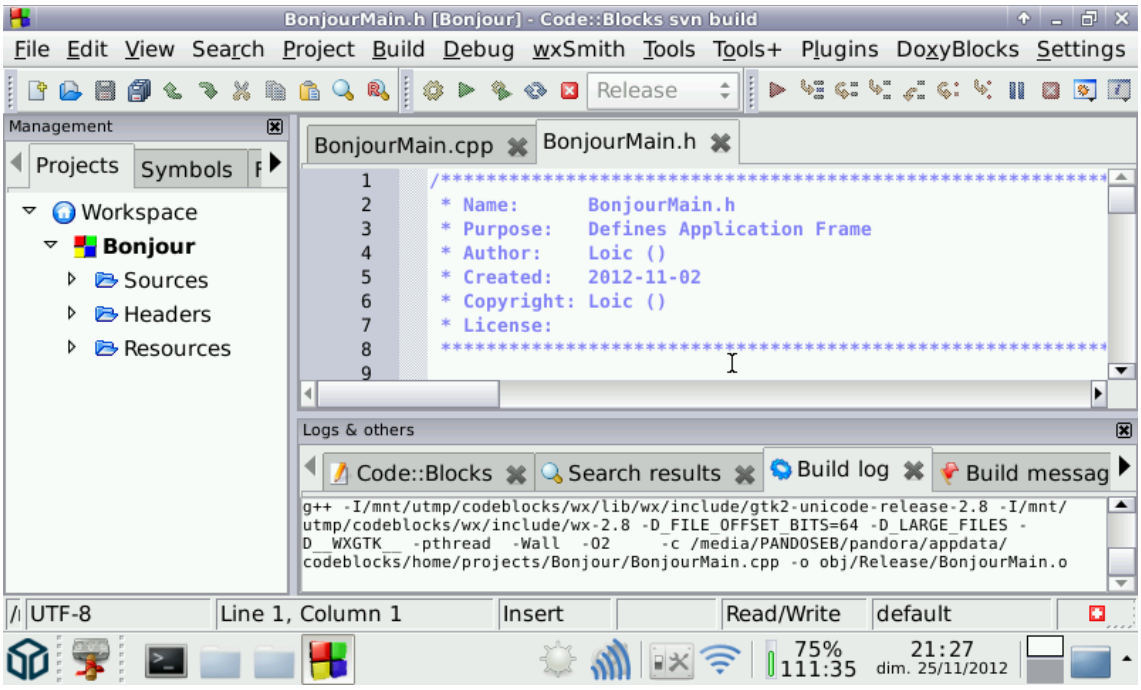
Príkazy pre preklad a spustenie programu:

cc *[options] <source.c>*	*cc (C Compiler)* – prelož
gcc *[options] <source.c>*	*gcc (GNU C Compiler)* – prelož
g++ *[options] <source.cpp>*	*g++ (GNU C++ Compiler)* – prelož
make	prekladaj podľa súboru *Makefile*
. *<script>*	spusti skript (za bodkou je medzera)
./*<program>*	spusti program alebo skript (bez medzery)
./a.out	spusti program (gcc vytvorí takýto súbor)

76 Operačné systémy (GNU/Linux), http://www.shenk.sk/skola/informatika/operacne-systemy-gnu-linux.pdf
77 *Midnight Commander (mc)*, http://en.wikipedia.org/wiki/Midnight_Commander

6.2 Code::Blocks

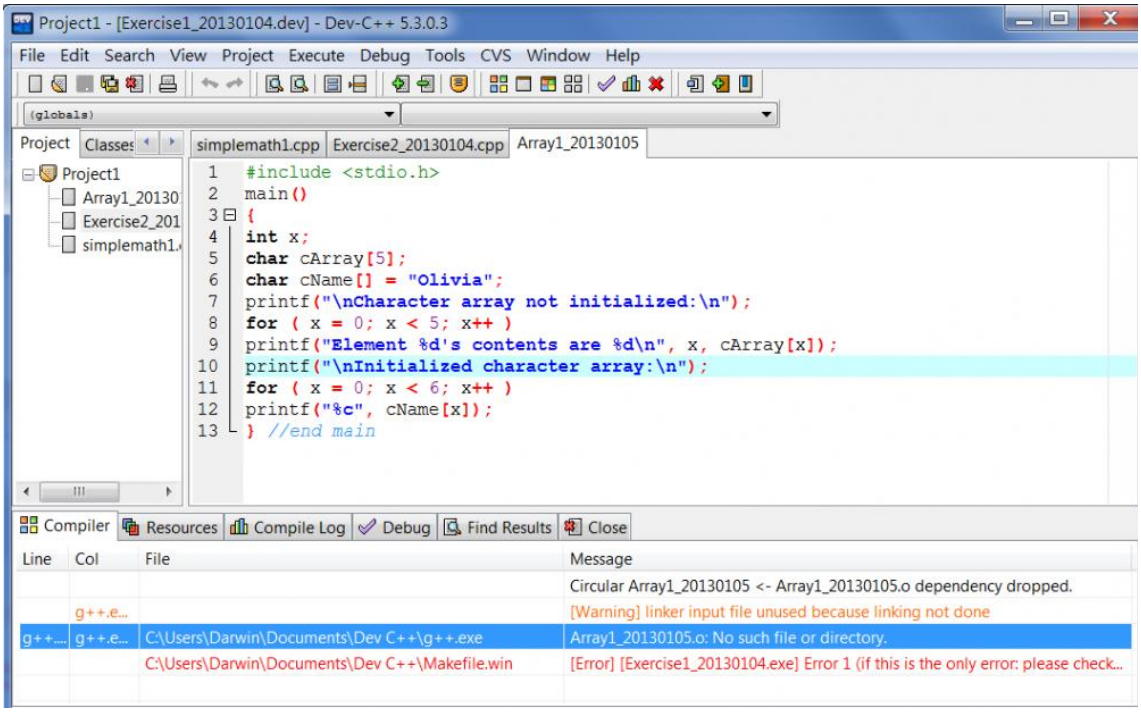
Tento slobodný softvér je dobrou voľbou vývojového prostredia pre začínajúcich aj pokročilých programátorov. Funguje vo viacerých operačných systémoch.[78]



Obrázok 32: Code::Blocks

6.3 Dev-C++

Alternatívne slobodné vývojové prostredie pre operačný systém MS Windows. Sú dostupné viaceré verzie: wxDev-C++, Orwell Dev-C++, Bloodshed Dev-C++.[79]



Obrázok 33: Dev-C++

78 Code::Blocks, https://en.wikipedia.org/wiki/Code::Blocks, http://www.codeblocks.org/, http://sourceforge.net/projects/codeblocks/

79 Dev-C++, http://wxdsgn.sourceforge.net/, http://orwelldevcpp.blogspot.sk/, http://www.bloodshed.net/devcpp.html

7 Jazyk C

V tejto veľkej kapitole sa naučíme programovať v štandardnom jazyku ANSI/ISO C (C11) pomocou série krátkych okomentovaných príkladov. Poradie príkladov je zámerné, s postupne narastajúcimi znalosťami a zručnosťami. Tieto príklady sú určené začínajúcim programátorom. Programovací jazyk C je v princípe jednoduchý. Expert v jazyku C musí sám napísať stovky takýchto programov, alebo pracovať na väčšom projekte dostatočne dlhý čas. Jazyk C je univerzálny a má široké možnosti, existuje veľmi veľké množstvo knižníc. Expert pozná knižnice, vyhľadá potrebný algoritmus a použije ho. Informovaný laik by povedal, že všetko už bolo vymyslené, treba to len nájsť a použiť. Samozrejme, vývoj ide stále dopredu a čo ešte nebolo vymyslené do dnes, bude vymyslené zajtra.

Pomoc možno hľadať v širokej plejáde odbornej literatúry, základy napríklad v stručnej, ale už zastaralej príručke *Programujeme v jazyku C* [5]. Nasledujúce očíslované súbory sú dostupné na stiahnutie aj v HTML verzii tohto učebného textu.[80]

7.1 Minimálny zdrojový kód

- Úloha: Napíšte minimálny zdrojový kód, ktorý sa dá preložiť. Program nemusí vykonávať žiadnu činnosť. Podstatný je najmenší počet znakov v zdrojovom súbore.

Použijeme ľubovoľný textový editor. Programátorský textový editor by mal vždy používať písmo v kontrastných farbách, typu `monospace/fixed`, napr. `Courier`, ktoré má rovnakú šírku každého znaku. Zdrojový kód je takto lepšie čitateľný. Vytvoríme prázdny súbor, napíšeme kód, uložíme súbor s názvom `01-minimal.c`. Názvy súborov sú v tomto texte číslované podľa kapitoly, aby ostali na disku usporiadané. Názov súboru by mal byť čo najkratší, nemá obsahovať národné znaky (diakritiku), ani medzery. To preto, aby sme zaručili, že súbor bude použiteľný na každom počítači, s každým operačným systémom. Program (zdrojový kód) v štandardnom jazyku ANSI/ISO C má byť nezávislý od hardvéru, operačného systému, vývojového prostredia.

`01-minimal.c`

```
main(){}
```

Program začína kľúčovým slovom `main`, je to hlavná funkcia. V okrúhlych zátvorkách `()` sú vstupné argumenty do funkcie, teda programu. V zložených zátvorkách `{}` je algoritmus funkcie, teda kód programu. Program nemá vstupy, ani nevykonáva žiadnu činnosť, nemá algoritmus.

Preklad zdrojového súboru urobíme vo zvolenom vývojovom prostredí, alebo v príkazovom riadku:

```
gcc 01-minimal.c
```

Prekladač vyrobí spustiteľný súbor `a.out`, pretože sme nezadali meno výstupného súboru. Inak:

```
gcc -o 01-minimal 01-minimal.c
```

Spustiteľný súbor `01-minimal` nemusí mať príponu (`.exe`), lebo v operačných systémoch GNU/Linux a UNIX je rozhodujúci typ súboru, ktorý je rozpoznaný operačným systémom pri čítaní začiatku súboru, rozlišuje sa architektúra systému (ELF 32-bit LSB / ELF 64-bit LSB). Spustiteľný súbor musí mať nastavené oprávnenie na spúšťanie (`rwx`). Spustíme program v príkazovom riadku:

```
./01-minimal
```

Vyskúšajme preklad s parametrom pre hlásenie varovaní a pokračujme v nasledujúcom príklade:

```
gcc -Wall -std=c11 01-minimal.c
```

80 Programovanie v C/C++ (HTML verzia), http://www.shenk.sk/skola/programovanie/

7.2 Korektný zdrojový kód

Aj napriek tomu, že predošlý zdrojový kód sa preložil (bez chýb), varovania nás upozorňujú, že zdrojový kód nie je v súlade s aktuálnym štandardom (napr. C11). Zvyknime si na štandard.

Každý program má pri ukončení odovzdať (*return*) operačnému systému návratovú hodnotu, čo je celé číslo. Ak program končí bez chyby, má odovzdať číslo `0`. Návratová hodnota musí mať priradený správny dátový typ. Celé číslo – **`int`** (*integer*).

Obyčajne sa príkaz v jazyku C končí bodkočiarkou. Preto nie je nutné pridávať medzeru alebo zalomenie riadku. Niektoré jazyky potrebujú zalomenie riadku (napr. Python).

V jazyku C musia byť medzerou oddelené od seba kľúčové slová, názvy operátorov, aby boli jednoznačné. Nadbytočné medzery, tabulátory, zalomenia riadkov sa ignorujú. Prekladač číta súbor po znakoch od začiatku po koniec a rozpoznáva jazyk. Človek-programátor vníma zdrojový kód aj vzhľadovo. Medzery, tabulátory, zalomenia riadkov človeku pomáhajú v čitateľnosti zdrojového kódu. Niektoré programovacie jazyky (napr. Python) používajú medzery, tabulátory, zalomenia riadkov ako povinné znaky pre identifikovanie jednotlivých príkazov, blokov príkazov. Preto si zvyknime písať dobre čitateľný kód. Patrí to ku dobrému hodnoteniu programátora.

Odporúča sa na koniec súboru so zdrojovým kódom pridať zalomenie riadku, teda prázdny riadok. Niektoré programátorské nástroje, prekladače, operačné systémy vyžadujú, aby sa každý riadok končil znakom ukončenia riadku (EOL, *end of line*), čo je klávesa ENTER.

`02-main.c`

```
int main() {
  return 0;
}
```

7.3 Komentár

Čitateľnosť a zrozumiteľnosť zdrojového kódu sa podstatne zvýši pridaním komentárov. Komentár (poznámka) je informačný text, ktorý je určený len programátorovi a prekladač ho ignoruje. Komentár má vysvetľovať názvy operátorov, algoritmus a pomôcť programátorovi pokračovať v práci neskôr, keď už si nepamätá, ako pri programovaní postupoval.

```
// jednoriadkovy komentar


/*
   viacriadkovy komentar
*/
```

Jazyk C je rozpoznávaný ako formálny jazyk s presne určenou abecedou. Zdrojový kód má byť písaný anglickou klávesnicou v kódovaní *7-bit ASCII*.[81] To znamená, že bez národných znakov (diakritiky). Ak potrebujeme do programu vložiť text s národnými znakmi, odporúča sa kódovanie *Unicode UTF-8*.[82] Takýto text by mal byť v osobitnom pripojenom súbore mimo jazyka C.

81 *ASCII*, https://sk.wikipedia.org/wiki/ASCII, http://www.asciitable.com/
82 *Unicode UTF-8*, https://sk.wikipedia.org/wiki/UTF-8, https://en.wikipedia.org/wiki/UTF-8

Na začiatku každého súboru so zdrojovým kódom má byť v komentári **informačná hlavička**. Programátor do hlavičky napíše názov programu, stručný popis programu, mená všetkých autorov, dátum alebo rok vytvorenia súboru, číslo verzie, licenciu a iné dôležité poznámky. Odporúča sa písať komentáre v angličtine, lebo angličtina je medzinárodný jazyk programátorov. Ak názov súboru obsahuje slovo, malo by byť tiež anglické. Takto sa zvýši univerzálnosť a znovu-použiteľnosť zdrojového kódu.

`03-comment.c`

```
/*
  03-comment.c

  Programming in C
  Comment

  (C) 2015, Martin Sechny <martin.sechny@shenk.sk>

  This is a free software,
  you can use it under the terms of the GNU GPL 3.0 license.
  <http://www.gnu.org/licenses/>

  In this example we describe the use of comments.
  Program does nothing.
*/

int main() {             // main function, start of program
  return 0;
}
```

Licencia GNU GPL[83] by mala byť priložená v textovom súbore ku zdrojovému kódu, aj ku preloženému programu, ak je dodávaný ako samostatný balík. Textový súbor s licenciou máva rôzne názvy, napr. `COPYING`, `COPYRIGHT`, `Copyright`, `LICENSE`, `License`, `README`. Informácia o licencii v hlavičke zdrojového kódu môže byť formulovaná stručne, dlhšie, aj v plnom znení. Odporúčané príklady použitia sú na konci textu licencie GNU GPL.

Kvôli čitateľnosti tohto učebného textu nebudem v ďalších príkladoch uvádzať informačnú hlavičku. Ale zdrojové súbory v HTML verzii tohto učebného textu sú kompletné.[84]

83 GNU GPL, http://www.gnu.org/licenses/, http://www.gnu.org/licenses/gpl-3.0.txt
84 Programovanie v C/C++ (HTML verzia), http://www.shenk.sk/skola/programovanie/

7.4 Knižnica

Ak chceme v zdrojovom kóde použiť meno dátového typu, konštanty, premennej, alebo funkcie, musí byť toto meno najprv deklarované – oznámené. Deklarácia musí byť umiestnená v kóde vždy pred použitím. Ak chceme použiť meno, ktoré nie je deklarované v danom súbore, ale v inom, musíme ten iný súbor pripojiť na začiatok zdrojového kódu.

`04-lib.c`

```
#include <stdio.h>                  // library, header file

int main() {
  printf("Ahoj\n");                 // output to screen
  return 0;
}
```

Funkcia **`printf`** nie je deklarovaná v tomto zdrojovom kóde, prekladač zahlási chybu, lebo nepozná takú funkciu. Táto funkcia je definovaná (a deklarovaná) v knižnici **`stdio.h`**. Knižnicu pripojíme na začiatok zdrojového kódu. Prekladač spojí všetky pripájané súbory a prekladá akoby jeden dlhý zdrojový kód. Aby bol preklad rýchly, knižnice sú na disku už dopredu preložené:

- *include* knižnica API (.h) *(header files*)
- *lib* knižnica API (.o, .so, .obj, .a, .dll) *(compiled object files*)

source code → *compiler* → *object* → *linker* → *executable program* → *loader* → *process*

Ujalo sa označovať zdrojový súbor knižnice ako hlavičkový súbor, pretože sa pripája na začiatku. Pripájaný súbor musí byť umiestnený v tom istom priečinku, ako prekladaný súbor, alebo sa uvedie s názvom súboru aj cesta, alebo ide o nainštalovanú systémovú knižnicu, ktorú vie operačný systém vyhľadať automaticky.

Poznámka: Niektoré vývojové prostredia uľahčujú prácu programátorovi tak, že nevyžadujú pripojenie štandardných systémových knižníc v napísanom zdrojovom kóde. To však nie je vždy dobrá pomoc, lebo zdrojový kód nemusí byť preložiteľný inokedy, v inom vývojovom prostredí, alebo na inom počítači. Niektoré programovacie jazyky sa viac spoliehajú na vývojové prostredie s prekladačom, preto nie je nutné pripájať knižnice.

Programátor ku svojej práci potrebuje príručku so zoznamom a popisom funkcií a knižníc. Existujú papierové knihy, elektronické dokumenty (HTML, PDF), alebo príručky vo vývojovom prostredí, alebo príkazy v príkazovom riadku operačného systému. Základný prehľad štandardných systémových knižníc pre jazyk C sa dá nájsť na webe pod menom: *GNU libc manual*. Zobrazme si pomoc ku funkcii **`printf`** v príkazovom riadku, alebo vyhľadajme na webe:

```
man printf
```

Funkcia alebo príkaz **`printf`** existuje viackrát, všimnime si v prvom riadku manuálu číslo. Manuál pre programátora jazyka C má číslo 3. Skúsme znova:

```
man 3 printf
```

Manuál je písaný obvykle tak, že najprv je informácia o knižnici, potom syntax príkazu, vysvetlenie príkazu, príklady, chyby, podobné príkazy. Zoznam všetkých funkcií v knižnici **`stdio.h`**:

```
man stdio.h
```

7.5 Premenná a dátový typ

Konštanta je veličina, ktorá má nemennú (konštantnú) hodnotu, alebo nemenný dátový typ. Premenná je veličina, ktorej hodnota aj dátový typ sa môže meniť. Konštanta sa dá chápať ako špeciálny prípad premennej. V jazyku C každá premenná má mať priradený správny dátový typ. Deklarácia premennej je jednorázové oznámenie mena premennej s priradeným dátovým typom. Definícia alebo inicializácia premennej je prvé vloženie (priradenie) hodnoty.

```
typ meno;                    deklarácia
typ meno=hodnota;            deklarácia a definícia (inicializácia)
meno=hodnota;                priradenie hodnoty
```

Dátový typ definuje použiteľné operácie s premennou a veľkosť potrebného pamäťového miesta pre uloženie hodnoty premennej. Je zrejmé, že množina možných hodnôt v počítači musí byť konečná, ohraničená, lebo fyzická pamäť počítača je konečná. Každé pamäťové miesto má svoju adresu pridelenú operačným systémom. Pamäťové miesto môže byť pridelené staticky – pri spustení programu na celú dobu behu programu, alebo dynamicky – príkazom počas behu programu. Výrez hlavnej pamäte:

V jazyku C sa často používajú špeciálne symboly: ***** pre hodnotu, **&** pre adresu.

Základné dátové typy sú: prázdny (*void*), celé číslo (*int*), reálne číslo (*float*), znak (*char*). Prázdny dátový typ sa použije vtedy, keď je potrebná len adresa, ale nie hodnota. Dátový typ *int* má veľkosť v B závislú od architektúry počítača (operačného systému) z historických dôvodov, čo je komplikácia pri písaní univerzálneho zdrojového kódu. Podľa veľkosti v B je daný rozsah hodnôt.[85]

Tabuľka 1: Prehľad veľkostí pre dátový typ int

system	***short int***	***int***	***long int***	***long long int***
16 b	2 B = 16 b	2 B = 16 b	4 B = 32 b	8 B = 64 b
32 b	2 B = 16 b	4 B = 32 b	4 B = 32 b	8 B = 64 b
64 b	2 B = 16 b	4 B = 32 b	8 B = 64 b	8 B = 64 b

Existujú aj dátové typy pre celé čísla s pevne danou veľkosťou pamäťového miesta (napr. *int64_t*). Nové dátové typy sa podľa štandardu C99/C11 majú pomenovať tak, aby končili znakmi: *_t*

Štandard POSIX[86] zabezpečuje kompatibilitu pri programovaní pomocou API, najmä použitím týchto štandardných systémových knižníc:

```
unistd.h                     POSIX verzia
limits.h                     konštanty, dátové typy
```

NIST[87] poskytuje testovanie kompatibility so štandardom POSIX.

85 Dátový typ *int*, https://en.wikipedia.org/wiki/Integer_%28computer_science%29, https://en.wikipedia.org/wiki/C_data_types
86 POSIX (*Portable Operating System Interface for UNIX*), http://en.wikipedia.org/wiki/POSIX
87 NIST (*National Institute of Standards and Technology, USA*), http://www.nist.gov/

`05-variable.c`

7.6 Konzola

7.7 Podmienka a vetvenie

7.8 Cyklus

7.9 Pole

7.10 Matica

7.11 Piškvorky

7.12 Usporiadanie (*bubble sort*)

Zaujímavé príklady a návody na tvorbu algoritmov sú v knihe *Perly programovania* [6].

7.13 Minimálny zdrojový kód

Vstupné a výstupné hodnoty programu

7.14 Funkcia

7.15 Faktoriál

7.16 Menu

7.17 Súbor

7.18 Súbor a triedenie

7.19 Ukazovateľ

7.20 Ukazovateľ a pole

7.21 Ukazovateľ a funkcia

7.22 Usporiadanie (*quick sort*)

7.23 Štruktúra

7.24 Textový reťazec

7.25 Formálny jazyk, regulárny výraz, gramatika

7.26 Heslo

7.27 Oneskorenie

7.28 Náhodné číslo

7.29 Matematika

7.30 Dynamická pamäť

7.31 Sieťová komunikácia

7.32 Proces

7.33 Pripojenie zdrojového súboru

7.34 Skript pre preklad (*Makefile*)

7.35 Projekt

7.36 Grafická aplikácia (GTK+)

7.37 Grafická aplikácia (Windows)

8 Jazyk C++

Obsiahla učebnica *Objektovo-orientovaná tvorba systémov a jazyk C++* [7].

8.1 Zdrojový kód

8.2 Knižnica

8.3 Konzola

8.4 Textový reťazec

8.5 Súbor

8.6 Dynamická pamäť

8.7 Trieda

8.8 Objekt

8.9 Dedenie

8.10 Geometria

8.11 Grafická aplikácia (GTK+)

8.12 Grafická aplikácia (Qt)

8.13 Grafická aplikácia (wx)

8.14 Semafor

9 Návrhové vzory

10 Optimalizácia

Výpočtová zložitosť – časová a pamäťová – ohodnocuje algoritmus, či optimálne využíva procesor a pamäť. Sú to protichodné požiadavky.

11 Paralelné programovanie

11.1 openMP

11.2 MPI

12 Automatické generovanie kódu a dokumentácie

Dokumentácia sa dá aspoň čiastočne vygenerovať zo zdrojového kódu s pomocou (aj iných):

doxygen	C/C++ a viacero ďalších jazykov
javadoc	Java

Diagramy pre dokumentáciu sa dajú tiež vygenerovať zo zdrojového kódu pomocou:

UML CASE	nástroje na kreslenie UML diagramov
cpp2dia	C/C++ zdrojový kód skonvertuje do UML diagramov pre Dia

13 Záver

14 Metodika

Prehľad súčasného stavu vo vyučovaní programovania v EÚ je v správe *Computing our future: Computer programming and coding – Priorities, school curricula and initiatives across Europe*[88].

88 *Computing our future: Computer programming and coding – Priorities, school curricula and initiatives across Europe*, http://www.eun.org/publications/detail?publicationID=481, http://www.eun.org/c/document_library/get_file?uuid=521cb928-6ec4-4a86-b522-9d8fd5cf60ce&groupId=43887

15 Literatúra

[1] Ľ. Molnár, M. Češka, B. Melichar: Gramatiky a jazyky, alfa Bratislava, SNTL Praha, 1987, 063-557-87
[2] Frank Schweitzer, Vahan Nanumyan, Claudio J. Tessone, Xi Xia: How do OSS projects change in number and size? A large-scale analysis to test a model of project growth, 2014, Advances in Complex Systems, 17, 1550008, arXiv:1506.06924v1 [cs.SE], http://arxiv.org/abs/1506.06924
[3] Faheem Ahmed, Piers Campbell, Azam Beg, Luiz Fernando Capretz: Soft Skills Requirements in Software Architecture’s Job: An Exploratory Study (What Soft Skills Software Architect Should Have? A Reflection from Software Industry), 2015, , , http://arxiv.org/abs/1508.07283, http://arxiv.org/pdf/1508.07283v1.pdf
[4] Miroslav Dressler: Programovací jazyky GNU, volně šiřitelná programátorská prostředí, Computer Press, 1998, ISBN 80-7226-070-7
[5] Peter Galan: Programujeme v jazyku C, FUTURE PRESS, 1991, ISBN 80-900402-2-5
[6] Jon Bentley: Perly programovania, Alfa Bratislava, 1992 (originál 1986), ISBN 80-05-01056-7
[7] Ľubor Šešera, Aleš Mičovský: Objektovo-orientovaná tvorba systémov a jazyk C++, PERFEKT Bratislava, 1994, ISBN 80-85261-66-9

16 Zdroje

Európska iniciatíva pre podporu programovania, http://www.allyouneediscode.eu, http://no1leftbehind.eu/wp-content/uploads/2015/04/Logo_AYNIC_Final__ECI_white_bg-04-1030x520.png 5

DNA kód, http://imagenes.montevideo.com.uy/imgnoticias/201410/_W620/472075.jpg 5

Scratch, http://1.bp.blogspot.com/-xUYfDZWTKW0/UI2PkSUYTBI/AAAAAAAAGgU/fa7SauYrLwg/s1600/fly-o-rama.png 6

Scratch – algoritmus, http://1.bp.blogspot.com/_5Njmh9cAWX0/SwWCINXZdhI/AAAAAAAAAJk/UXL6d0mXQuY/s1600/sample.png 6

App Inventor, http://coderdojonavan.com/wp-content/uploads/2013/03/appinventor-doc-diagram.png 7

Evolúcia programovacích jazykov s humorom, http://4.bp.blogspot.com/-2-8AyqdLJkk/VNuV_v-wxDI/AAAAAAAABLg/IFRu0UxnBPM/s1600/History%2Bodf%2BProgramming.jpg 8

Objektové programovacie jazyky, http://sa.whitneyhighfoundation.org/wp-content/uploads/2015/04/Programming.png 8

Nauč sa a zarábaj, http://www.missiontolearn.com/wp-content/uploads/2009/11/learn_earn-300x199.jpg 8

ISO 9001, ISO/IEC 20000, ISO 27001, http://1.bp.blogspot.com/-BMVXMrXcyGU/T469CFjSIVI/AAAAAAAAAe0/7jBHYkHJMlA/s1600/ISO9001-20000-27001+preview.jpg 16

Scrum, http://www.intellias.com/images/stories/scrum_circle_en_big.jpg 18

Business model, http://openlife.cc/system/files/BusinessModelsSpectrum-OpenCore.png 24

Slax, https://www.slax.org/images/ss1th.png 28

Code::Blocks, http://repo.openpandora.org/files/pnd/codeblocks6022/preview1.png 30

Dev-C++, http://cboard.cprogramming.com/attachments/c-programming/12401d1357379863-dev-cplusplus-dependency-dropped-initiation-array_20130105.jpg 30

17 Register