Učebnica jazyka C

Autor:
Lukáš Vanek

1. Úvod

1.1 História a charakteristika jazyka C

Programovací jazyk C vznikol na začiatku sedemdesiatych rokov v Bellových laboratóriách firmy AT&T. Prvým štandardom jazyka bola verzia jeho autorov – **Brian W. Kernighan** a **Denis M. Ritchie** – opísaná v knihe **The C programming Language**. Táto kniha vyšla v roku 1978 a okrem iného sa stala základnou učebnicou jazyka C. Opisovaný štandard jazyka C sa označuje ako K&R.

Dnešný štandard je tzv. ANSI C z roku 1988, ktorý vychádza z K&R. Jeho súčasťou je aj množina funkcií a hlavičkových súborov **.h**, ktorú musí každý kompilátor ANSI C obsahovať. Tejto norme by mala vyhovovať väčšina dnešných prekladačov. Výhodou ANSI C je, že program napísaný s využitím iba štandardných funkcií je takmer 100% prenosný na ľubovoľný počítač pod ľubovoľný operačný systém. Pokiaľ je nutná nejaká zmena, tak je minimálna.

Charakteristika jazyka C:
- je to univerzálny programovací jazyk nízkej úrovne (low level language)
- má veľmi úsporné vyjadrovanie, je štruktúrovaný, má veľký súbor operátorov
- nie je špecializovaný na jednu oblasť používania
- pre veľa úloh je efektívnejší a rýchlejší ako iné jazyky

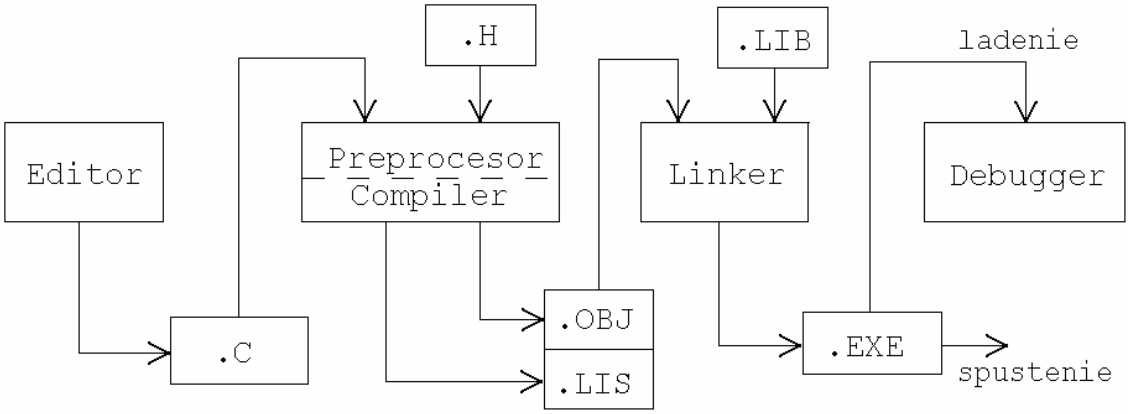
Jazyk C nie je rozsiahly, ale nepatrí pritom medzi jazyky vyznačujúce sa dobrou čitateľnosťou. Svojím pôvodom je zameraný na systémové programovanie. Na jednej strane je ho možné zaradiť do skupiny vyšších programovacích jazykov, na druhej strane je možné program napísaný v jazyku C preložiť do veľmi efektívneho strojového kódu (niekedy sa preto o ňom hovorí ako o štruktúrovanom assemblery). Jazyk C má pomerne málo jednoduchých pravidiel, pomocou ktorých je možné vytvárať a skladať jednotlivé úseky programov do väčších a väčších celkov.

Jazyk C patrí medzi tzv. **kompilačné jazyky**, v ktorých vytvorený zdrojový program musíme najskôr preložiť do podoby, ktorej rozumie procesor, a až potom môžeme program spustiť. Nejde teda o tzv. interpretačný jazyk, ktorý prevádza vždy preklad jedného riadku nášho zdrojového programu. Takýto jazyk prevádza postupnú interpretáciu zdrojového programu. Príkladom takéhoto jazyka je jazyk Basic. V jazyku C musíme po každej zmene zdrojového súboru previesť opakovaný preklad.

Jazyk C je implementovaný pre všetky typy procesorov a spolu so systémom UNIX predstavuje dnes jeden z hlavných trendov vo svete. Vznikol tiež celý rad rozšírení jazyka C najperspektívnejším sa zdá byť objektovo orientované rozšírenie jazyka pod označením C++ (označenie vychádza zo syntaxe jazyka C a znamená prechod na nasledujúci prvok), ktoré navrhol Bjarne Stroustrup. Komisie ANSI a ISO, ktoré pripravujú normu jazyka C++ (označuje sa ako ANSI/ISO C++). Táto norma bola oficiálne prijatá až okolo roku 1996. Kompilátory jazyka C++ dnes ponúkajú všetky významné firmy (Borland - **Borland C++**, Microsoft - **Microsoft Visual C++**). V súčasnosti sa kladie veľký dôraz na objektovo orientované programovanie (OOP), podporu databáz priamo cez ovládač v operačnom systéme ODBC (Open Database Connectivity), poprípade DAO (Data Access Object) a taktiež na podporu výmeny objektov medzi aplikáciami OLE (Object Link Embedding) alebo DDE (Dynamic Data Exchange).

1.2 Základné pojmy

1.2.1 Spôsob spracovania programu



- **Editor** – pomocou editora sa vytvára a upravuje zdrojový súbor (**.c, .cpp**)
- **Preprocesor** – je to súčasť prekladača, ktorá pred spracováva (upravuje) zdrojový súbor tak, aby mal prekladač ľahšiu prácu. Napr. vynecháva komentáre, zaisťuje správne vloženie hlavičkových súborov (**.h**), rozvoj makier, atď.
- **Compiler –** nazývaný tiež *prekladač* alebo *kompilátor*, prevádza preklad zdrojového súboru (spracovaného už preprocesorom) do relatívneho (objektového) kódu počítača – vzniká **.obj** súbor. Relatívny kód je takmer hotový program len adresy premenných alebo funkcií ešte nie sú vôbec známe. Vedľajší produkt prekladača je produkt o preklade (súbor **.lis**), v ktorom je uložená informácia o chybách nájdených prekladačom.
- **Linker** – zostavovací program pridelí relatívnemu kódu absolútne adresy a prevedie všetky odkazy na doteraz neznáme identifikátory. Výsledkom práce je priamo spustiteľný program (**.exe**)
- **Debugger** – ladiaci program, ktorý slúži pre ladenie alebo hľadanie chýb, ktoré nastávajú pri behu programu. Po nájdení chyby sa celý cyklus (editor, compiler, linker, Debugger) opakuje tak dlho, až si myslíme, že náš program žiadnu chybu neobsahuje.

1.2.2 Základné pojmy v jazyku C

Doteraz boli opisované základné pojmy z hľadiska spracovania celého programu. Teraz nás bude zaujímať iba to, čo spracováva prekladač s preprocesorom, teda zdrojový program v jazyku C.

- **Zdrojové a hlavičkové súbory** – zdrojový súbor **.C**, v ktorom je náš program, je pre jeho správnu funkciu väčšinou nutné doplniť *vložením súboru*. Najčastejšie vkladáme (include) tzv. hlavičkové (header -ové) súbory **.H**. Používajú sa preto, že program väčšinou volá knihovnu funkcie (I/O, ...).
- **Biele znaky** (white spaces) – zahrňuje znaky, ktoré sú síce veľmi dôležité, ale nie sú na obrazovke vidieť. Sú to tzv. oddeľovacie znaky ako medzera, tabulátora, nový riadok, nová strana, návrat na začiatok riadku, atď.
- **ASCII tabuľka** – popisuje kódy, ktoré sú pridelené jednotlivým znakom. Má rozsah od 0 do 255, ale bežne sa pracuje len s jej dolnou polovinou, teda so znakmi od 0 do 127. Horná polovica ASCII tabuľky je vyhradená pre znaky národných abecied a pre niektoré špeciálne znaky (napr. rámčeky, matematické symbolmi, ...).

Dolná časť ASCII tabuľky

riadiace znaky	0	(00h)	31	(1Fh)	neviditeľné
medzera	32	(20h)	' '		
pomocné znaky	33	(21h)	'!'	47	(2Fh)	'/'
číslice	48	(30h)	'0'	57	(39h)	'9'
pomocné znaky	58	(3Ah)	':'	64	(40h)	'@'
veľké písmená	65	(41h)	'A'	90	(5Ah)	'Z'
pomocné znaky	91	(5Bh)	'['	96	(60h)	'`'
malé písmena	97	(61h)	'a'	122	(7Ah)	'z'
pomocné znaky	123	(7Bh)	'{'	126	(7Eh)	'~'

Neviditeľné znaky sú napr.:
7 – Bell
8 – Backspace
9 – Tab
10 – LineFeed
13 – CarriageReturn (Enter)

- **Identifikátory** – jazyk C je case sensitive jazyk tj. rozlišuje malé a veľké písmena. V praxi to znamená, že: ram Ram RAM sú tri rôzne identifikátory!

Kľúčové slová jazyk C (napr. **if**, **while**, **register**, ...) musia byť napísané malými písmenami. Ak sú napísané veľkými alebo kombinácia malých a veľkých písmen, tak sa berú ako identifikátory.

- **Komentáre** – úlohou komentárov je sprehľadniť, niekedy na prvý pohľad dosť nepochopiteľný, program. Komentár slúži k tomu, aby sa vo vašom programe vyznali aj iní aj vy sám, keď sa k tomuto programu vrátite za nejakú dobu. Začiatok komentára sa začína znakmi **/*** a končí znakmi ***/**.

1)
```
/*
 * Výrazný
 * viacriadkový
 * komentár
 */
```

2)
```
/*
 * jednoriadkový popis funkcie
 */
```

3)
```
/* dlhší popis nasledujúcej logickej časti kódu,
   ktorý sa nezmestí na jeden riadok */
   if ((fr = fopen(str,"r")) == NULL) {}
```

4)
```
x = 3 * a + b;     /* popis príkazu */
```

1.3 Turbo C – popis, prostredie, menu, klávesy

1.3.1 Popis a prostredie

Integrované programátorské prostredie (IDE – Integrated Development Environment) prekladača Turbo C 1.0 (podobné prostredie je aj Borland C++ 3.1) obsahuje tieto funkcie:

- editor
- kompilátor (prekladač)
- linker
- diagnostický program
- debugger (ladiaci program)
- krokovací program
- nápoveď

Integrované programátorské prostredie umožňuje vytvoriť jeden alebo viacej zdrojových textov, uložiť ich do súboru, preložiť, zlinkovať a diagnostikovať vytvorený program, opravovať zdrojový súbor (napr. pri výskyte nejakej chyby pri preklade, pri výpočte alebo po zistení, že program neplní účel, ktorý sme zamýšľali) s možnosťou návratu k ľubovoľnej predchádzajúcej funkcií bez opustenia integrovaného programátorského prostredia. Pri úspešne zakončenej činnosti je potom na disku k dispozícií príslušné **.EXE** súbor, ktorý je naďalej možný spúšťať v prostredí kompatibilnom s MS DOSom, bez toho aby bola potreba mať k dispozícií integrované programátorské prostredie, zdrojový súbor, vytvorený **.OBJ** súbor (modul) programu alebo knihovnu.

Integrované programátorské prostredie je realizované súborom TC.EXE pri prekladači Turbo C, ktorý sa nachádza v adresári \TC\BIN. Pri prekladači Borland C++ je to súbor BC.EXE, ktorý sa nachádza v adresári \BORLANDC\BIN.

Napr.: C:\TC\BIN\TC.EXE alebo C:\BORLANDC\BIN\BC.EXE.

Integrované programátorské prostredie prekladača môžeme ovládať buď len prostredníctvom klávesnice alebo prostredníctvom klávesnice spolu s myšou. Ovládanie s použitím myši je však rýchlejšie, pohodlnejšie a komfortnejšie.

1.3.2 Menu

Teraz si trochu popíšeme niektoré položky v Menu, s ktorými sa bežne nestretávame pri používaní iných programoch, a ktoré môžeme využiť pri našom programovaní. Menu vyvoláme stlačením klávesy F10 alebo kliknutím pomocou myši na menu, ktoré sa nachádza v hornej časti obrazovky.

- **Systém (≡)**

Prvou položkou je položka System (≡), ktorý zahrňuje globálne možnosti integrovaného programátorského prostredia: **About**, **Clear desktop**, **Repaint desktop** a **Transfer** položky.

- **File**

Ďalšou položkou, ktorú budeme používať asi najviac, je položka File, ktorá je určená pre prácu s editovanými diskovými súbormi (čítanie z disku, uloženie na disk, vytvorenie nového súboru, ...). Ďalej ponúka niektoré operácie súvisiace s operačným systémom a tlačiarňou: **Open**, **New**, **Save**, **Save as**, **Save all**, **Change dir**, **Print**, **Get info**, **DOS shell** a **Quit**.

Save all – uloží súbory vo všetkých otvorených oknách

Change dir – táto voľba umožňuje bez opustenia integrovaného programátorského prostredia interaktívnym spôsobom zmeniť aktívny adresár DOSu, prípadne iba zistiť štruktúru adresárov

Get info – zobrazí dialógový panel s informáciami o aktívnom súbore a kontexte

- **Edit**

Táto voľba má veľmi úzky vzťah s úpravou textu, umožňuje prenosy častí textu medzi oknami (**Cut**, **Copy**, **Paste**, **Copy example** a **Show clipboard**), rušenie časti textu (**Clear**) a k návratu predchádzajúcich stavov (**Undo** a **Redo**).

Copy example – text príkladu zobrazený v aktívnom okne nápovede je prekopírovaný do clipboardu, odkiaľ môže byť prenesený ďalej do iného okna

Clear – vyznačený text z aktívneho okna je vymazaný, avšak nie je skopírovaný ani do clipboardu

- **Search**

Obsahuje položky orientované na vyhľadávanie textu alebo iných významných miest súborov v oknách. Obsahuje položky: **Find**, **Replace**, **Search again**, **Go to line number**, **Previos error**, **Next error** a **Locate function**).

Go to line number – umožňuje nastaviť kurzor v otvorenom okne na riadok zadaného čísla

Locate function – realizuje vyhľadávanie deklarácií zadanej funkcie

- **Run**

Obsahuje všetky akcie súvisiace so spúšťaním programu. Obsahuje položky: **Run**, **Program reset**, **Go to cursor**, **Trace info**, **Step over**, **Arguments**.

Run – pri použití tejto voľby je spustený program. Ak bol zdrojový súbor upravovaný od jeho poslednej modifikácie, je automaticky prevedená kompilácia a linkovanie. Chod spusteného programu môžeme tiež zastaviť prostredníctvom kombinácie kláves CTRL + Break (Pause), ak je toto stlačenie bez efektu, tak jeho opakovaním. Pri opätovnom spustení programu program pokračuje od miesta násilného zastavenia.

Program reset – realizuje ukončenie krokovania, zavretím programom otvorených súborov a uvoľnenie alokovanej (obsadenej) pamäti. Po tejto akcií je chod programu vždy spustený od začiatku

Arguments – program uloží argumenty (parametre), ktoré sa pri spúšťaní nášho programu použijú

- **Compile**

Zahrňuje všetky činnosti súvisiace s kompiláciou a linkovaním zdrojového kódu, obsahuje položky: **Compile to OBJ**, **Make EXE file**, **Link EXE file**, **Build all** a **Remove messages**.

Build all – realizuje novú kompiláciu všetkých zdrojových súborov a linkovanie nových modulov bez ohľadu na to, či boli aktualizované od predchádzajúcej kompilácie alebo linkovania

Remove messages – vyprázdnenie okna varovných a chybových hlásení (Message window)

- **Debug**

Táto ponuka zahrňuje akcie pre krokovanie, obsahuje položky: **Inspect**, **Evaluate modify**, **Call stack**, **Watches**, **Toggle breakpoint** a **Breakpoints**.

Evalute modify – umožňuje vyčísliť a zobraziť hodnotu premennej alebo výrazu a prípadne upravovať túto hodnotu

Watches – krokovací prostriedok integrovaného programátorského prostredia umožňuje otvoriť špeciálne okno **Watch window**, zvoliť určité premenné a tieto premenné spolu s ich aktuálnymi hodnotami priebežne zobrazovať pri krokovaní programu.

- **Project**

Slúži k vytvoreniu a upravovaniu project súboru, ktorá obsahuje položky: **Open project**, **Close project**, **Add item**, **Delete item**, **Local options** a **Include files**.

- **Options**

Táto položka realizuje nastavenie parametrov integrovaného programátorského prostredia, obsahuje položky: **Compiler**, **Transfer**, **Make**, **Linker**, **Application**, **Debugger**, **Directories**, **Environment** a **Save**.

- **Window**

Položka realizuje organizáciu jednotlivých okien integrovaného programátorského prostredia prostredníctvom svojich položiek **Size/Move**, **Tile**, **Cascade**, **Next**, **Close**, **Message**, **Output**, **Watch**, **User screen**, **Register**, **Project**, **Project notes** a **List**.

User screen – zobrazenie výstupu programu pri jeho poslednom chode

- **Help**

Prostredníctvom tejto položky pracujeme so systémom nápovedy zabudovaným v integrovanom programátorskom prostredí. Môžeme ho vyvolať aj stlačením klávesy F1. Vyberanie a pohyb v nápovede sa realizuje šípkami, tabulátorom (TAB) a klávesov Enter. Voľba Help obsahuje položky: **Contents**, **Index**, **Topic search**, **Previous topic**, **Help on help**.

Contents – vyvolá tematický členený obsah nápovedy

Index – vyvolá abecedne zoradený register nápovedy obsahujúci všetky kľúčové slová nápovedy

Topic search – prostredníctvom tejto voľby je dostupná nápoveda vzťahovaná ku slovu na pozícii kurzoru v aktívnom editovanom okne

Previous topic – listovanie nápoveďou v predchádzajúcich dvadsiatich nápovedných oknách

Klávesové skratky

File	**F3** = Otvorenie už vytvoreného programu (OPEN) **F2** = Uložiť aktívny súbor (SAVE) **Alt + F3** = Zatvoriť aktívne okno (CLOSE) **Alt + X** = Koniec (QUIT)
Edit	**Shift +** ←→↑↓ = Označenie (od značenie) časti textu (bloku), alebo MYŠou **Ctrl + K + H** = Od značenie všetkého **Ctrl + Y** = Vymaže riadok **Shift + Delete** = Vystrihne vyznačený blok textu do schránky (CUT) **Ctrl + Insert** = Skopíruje vyznačený blok textu do schránky (COPY) **Shift + Insert** = Skopírovanie bloku na zvolené miesto v programe (PASTE) **Ctrl + Del** = Vymaže vyznačený blok textu (CLEAR)
Run	**Ctrl + F9** = Spustenie programu (RUN) **F9** = Vytvorenie spustiteľného modulu .EXE (MAKE) **Alt + F9** = Vytvorenie preloženého modulu .OBJ (COMPILE) **Ctrl + F2 (Break)** = Ukončenie behu programu (PROGRAM RESET)

	Alt + F5 = Zobrazenie výsledkov programu (USER SCREEN - podklad)
Ladenie	**F7** = Krokovanie programu po jednotlivých riadkoch (TRACE INTO). Vždy po vykonaní jedného riadku sa stlačí **F7** - prejde na ďalší. **F8** = Krokovanie bez vnorenia do jednotlivých funkcií (STEP OVER) **F4** = Program beží po riadok, v ktorom sa nachádza kurzor (GO TO CURSOR)
Help	**F1** = Nápoveda **Shift + F1** = Pomoc pre jazyk (INDEX) **Ctrl + F1** = Zobrazenie pomoci k určitému príkazu (Topic search) **Alt + F1** = Naspäť (predchádzajúca obrazovka syst. POMOC) Príklady z HELPu pri jednotlivých príkazoch sa dajú skopírovať do programu.

1.4 Štruktúra programu v jazyku C

Základnou programovou jednotkou v jazyku C je funkcia. Veľké množstvo štandardných funkcií je dopredu pripravených v knižniciach prekladača. Tieto funkcie vychádzajú na jednej strane z príslušnej normy ANSI, na druhej strane z "kuchyne" firmy - autora kompilátora. Každá štandardná funkcia je spravidla sprevádzaná svojím tzv. include - súborom (jedným alebo viacerými). Tento súbor má príponu **.h** od slova header (hlavička) a obsahuje súhrn informácií o určitej skupine štandardných funkcií (ich definície, globálne premenné a pod). Do zdrojového textu programu sú tieto súbory zaradené pomocou príkazu preprocesora **#include** (podrobnejšie viď kap. 2). Rovnakým spôsobom môžu byť do zdrojového textu programu zaradené užívateľom definované funkcie (podprogramy), ktoré je tiež možné združovať do knižnice.

Ďalším objektom, ktorý sa môže vyskytovať v štruktúre programu v jazyku C, sú globálne dátové objekty, ktoré sú dostupné vo všetkých funkciách daného programu. Doporučujeme však použitie týchto globálnych premenných vynechať, resp. obmedziť, pretože zhoršujú možnosti štruktúrovania programu.

Program v jazyku C je teda súhrnom definícií funkcií a deklarácií globálnych objektov. Jedna z funkcií sa musí volať **main** (označuje hlavný program). Jazyk C je typický svojou blokovou štruktúrou (napr. telo každej funkcie tvorí jeden blok). Blokom je zdrojový text, uzavretý v dvojici zátvoriek **{**, **}**. V rámci bloku je možné pracovať s globálnymi i lokálnymi premennými, ktoré sú definované len v rámci bloku (lokálne môžu byť len dátové objekty, všetky funkcie sú vždy globálne).

Štruktúra programu v jazyku C má teda obecný tvar:

1) skupina hlavičkových súborov štandardných funkcií
2) definície užívateľských funkcií
3) deklarácie globálnych premenných
4) funkcia **main**
5) ostatné užívateľské funkcie

2. Jazyk C

2.1 Prvý program

Jediný spôsob, ako sa dobre naučiť programovať je začať programovať, a preto začneme aj my s takým jednoduchým programom, ktorý je veľmi dôležitý. Má za úlohu vypísať text na obrazovku. Prvé čo musíš urobiť je spustiť program Turbo C alebo Borland C++. A v ňom máš otvorené prázdne okno (nový súbor), do ktorého už môžeš písať prvý program:

Príklad 1

```
#include <stdio.h>

void main()
{
  printf ("Ahoj svet, ja som programator");
}
```

Po napísaní celého programu si náš prvý zdroják ulož, a to tak, že stlač F2 alebo choď myšou na *File -> Save as*, otvorí sa ti dialógové okno. Ak si spustil Borland C z adresára, do ktorého chceš ukladať zdrojáky, nachádzaš sa hneď v ňom. Naťukaj meno - napr. ahoj.c (vlastne je jedno, akú má súbor koncovku, ale obyčajne sa zdrojákom v jazyku C dáva koncovka **.c** a zdrojákom v jazyku C++ koncovka **.cpp**).

Teraz stlač Ctrl + F9 , alebo choď na *Run -> Run* . Najskôr prebehne kompilácia (preklad) do exe-čka. Potom blikne obrazovka a ... no a nič sa nestalo. Spustenie programu v skutočnosti nastalo, ale program prebehol tak rýchlo, že si to nestačil zaregistrovať.

Ak chceš vidieť, čo vlastne program spravil stlač Alt + F5. Táto klávesová kombinácia skryje prostredie Turbo C (Borland C++) a ukáže výpis promptu MS-DOS. Na konci výpisu svietia slová Ahoj svet, ja som programator... Program funguje.

Textový výstup z nášho prvého programu zostane na obrazovke pokiaľ ho nevytlačí z plochy obrazovky ďalší text, alebo nebude obrazovka zmazaná. Rovnakou kombináciou (Alt + F5) sa dostaneš späť.

Ak znovu spustíš program, už sa nespustí kompilácia (pokiaľ si nič nerobil v programe), spustí sa program priamo. Na obrazovku pribudne ďalší nápis...

Ak vyjdeš z programu Turbo C (Borland C++) kombináciou Alt + X alebo *File -> Exit* , pozri sa do adresára, v ktorom má ukladať exe-čka. Budú tam súbory: *ahoj.obj* a *ahoj.exe*, ktorý je normálny **.exe** súbor a môžeš si ho spustiť.

A teraz si vysvetlíme trochu náš program. Prvý riadok je príkaz pre preprocesor, má za úlohu vložiť tzv. hlavičkový súbor, ktorý sa nazýva `stdio.h` (stdio je skratka z anglického "standard input output"). Slúži k informovaniu prekladača jazyka C, že budeme používať existujúcu knihovnu funkcií. Obecne platí, že v prípade použitia niektorej z knihovných funkcií musíme vždy vložiť odpovedajúci hlavičkový súbor. Zoznam všetkých dostupných funkcií a odpovedajúcich hlavičkových súborov je možné nájsť v referenčných príručkách alebo v helpe programov Turbo C a aj Borland C++.

Každý program v jazyku C musí obsahovať aspoň jednu funkciu, ktorá sa volá **main**. Táto funkcia nemá žiadne parametre, preto sú za ňou uvedené prázdne zátvorky. Tieto zátvorky u iných funkcií môžu obsahovať parametre. Telo funkcie, ktoré nasleduje a obsahuje

množinu príkazov pre vykonávanie, je ohraničené zloženými zátvorkami { }. Telo našej hlavnej funkcie obsahuje len jeden príkaz, a to príkaz `printf`, ktorý má za úlohu vypísať náš text (Ahoj svet, ja som programator) na obrazovku. Tento text je funkcii daný ako parameter a ako každý textový reťazec musí byť uvedený v úvodzovkách. Príkaz `printf` je definovaný v hlavičkovom súbore `stdio.h`, keby sme ho na začiatku programu nepoužili, tak by prekladač nevedel čo znamená príkaz `printf`.

Na začiatok by to už aj stačilo. Verím, že ste prvý program pochopili.

2.2 Typy premenných

Premenná je to oblasť v pamäti, ktorú si vyhradzuje program a môže si ukladať do nej čísla. Tejto oblasti pri vytvorení program priradí tzv. identifikátor (ang. identifier), ktorý túto oblasť pomenuje charakteristickým písmenom alebo slovom. V jazyku C však nie je jedno, či napíšeš *x* alebo *X*, to sú dva rôzne identifikátory. Samozrejme existuje viac typov premenných. Líšia sa veľkosťou oblasti (koľko majú bytov) a tiež tým, či používajú prvý bit na znamienko, alebo nie (tzv. znamienkové - ang. signed a neznamienkové - ang. unsigned). Veľkosťou premennej v bytoch je daný rozsah čísel, ktoré možno do premennej uložiť. Ak by sa program pokúsil uložiť do 1 bytovej premennej uložiť 2 bytové číslo, došlo by k tzv. pretečeniu (ang. overflow). Tá 1 bytová premenná by sa samozrejme zaplnila samými jednotkami (číslo 255) a zvyšok by sa zapísal o byte vedľa, aj keď ten priestor patrí inej premennej. Pôvodný obsah toho susedného bytu sa jednoducho prepíše.

```
int j;          - globálna premenná (mimo akýchkoľvek funkcií)


{
  int i;        - lokálna premenná (vo vnútri funkcií)
}
```

Celočíselné typy premenných

- *int* - najčastejšie používaný typ premennej v jazyku C. Ako už názov hovorí, slúži na ukladanie celých čísel. Zaberá 2 byty a je automaticky znamienkového typu (signed). Teda rozsah čísel, ktoré možno do premennej typu *int* uložiť je od -32768 do 32757. Ak neplánuješ premennú tohto typu používať na ukladanie záporných čísiel a nechceš zbytočne plytvať pamäťou, stačí napísať *unsigned int* a premenná bude mať rozsah od 0 do 65535 (bude neznamienková, teda unsigned).
- *long* - profesionálny programátori na zväčšenie rozsahu premennej typu *int* obyčajne nepoužívajú *unsigned int* ale *long int* (skracuje sa na *long*). Typ *long* je rovnako ako typ *int* znamienkový (signed), ale zaberá 4 byty. Má teda rozsah od -2^{31} až $(2^{31}-1)$. Samozrejme aj tento typ si ľahko môžeš prepnúť na neznamienkový (unsigned) zápisom *unsigned long int* (skracované na *unsigned long*), potom bude mať rozsah od 0 až $(2^{32}-1)$. Možno sa teraz pýtaš, aký má zmysel zbytočne zabrať 4 byty typom *long*, ak používaš len kladné čísla a chceš len zdvojnásobiť rozsah (na čo stačí *unsigned int*, ktorý zaberá len 2 byty)? Nikdy nemôžeš vedieť, kedy bude výsledok záporné číslo...
- *char* - premenná sa používa na ukladanie znakov. V BIOSe existuje tzv. tabuľka znakov. Táto tabuľka sa naťahuje pri zapnutí počítača a každému znaku je priradené číslo (dokopy je ich tam 256, teda sú číslované od 0 do 255). V Céčku sú príkazy, ktorým stačí zadať číslo znaku a vytlačia ho. Mal by som teda poopraviť prvú vetu - premenná typu *char* je určená na ukladanie čísla znaku v tabuľke znakov. Je potom logické, že premenná tohto typu zaberá 1 byte a je neznamienkového (unsigned) typu (v rozsahu 0-255 nie sú záporné čísla). Na

uloženie čísla znaku v tabuľke znakov sa môže používať aj typ *int* a často sa aj používa. Tiež možno používať typ char na ukladanie čísla od 0 do 255. Samozrejme funguje aj *signed char* (znamienkový *char*), ktorý má rozsah od -128 do 127. Tento typ má zmysel vtedy, ak používaš len spodnú polovicu tabuľky znakov a potrebuješ ukladať aj záporné čísla.

Reálne typy premenných

V jazyku C sa im hovorí čísla s pohyblivou desatinnou bodkou. Existujú tri typy reálnych premenných, líšia sa len rozsahom. Všetky tri sú implicitne (ak nenapíšeš bližšiu špecifikáciu) *signed* (znamienkové). Samozrejme si ich môžeš prepnúť na *unsigned*, ak by si chcel.

- *float* - premenná tohto typu zaberá 4 byty v pamäti (ako typ *long*). Rozsah je $\pm 3{,}4 \cdot 10^{\pm 38}$.
- *double* - táto premenná zaberá 8 bytov. Jej rozsah je $\pm 1{,}7 \cdot 10^{\pm 308}$.
- *long double* - no a táto premenná zaberá 10 bytov. Rozsah je $\pm(3{,}4 \cdot 10^{-4932} - 1{,}1 \cdot 10^{+4932})$

Typ	float	double	long double
Celková veľkosť [bit]	32	64	80
mantisa – exponent [bit]	24 -7	53 -10	64 -15
Počet desatinných miest	6	15	18
Najmenšie kladné číslo	1.192092896e-07	2.2204460492503131e-016	1.0842022172485504434e-019
Max./Min. uložiteľné číslo	3.402823466e+38/ 1.175494351e-38	1.7976931348623158e+308/ 2.2250738585072014e-308	1.189731495357231765e+4932/ 3.3621031431120935063e-4932
Maximálny exponent	38	308	4932

2.3 Vstup a výstup

Na začiatok musím konštatovať, že nikoho z nás práca so vstupom a výstupom neminie. Každý program spracováva nejaké údaje a výsledky zobrazuje alebo uchováva. Ťažké si je predstaviť program, ktorý by si údaje sám vyrobil a výsledky spracovania nezobrazil.

Zdrojom údajov, rovnako ako cieľom pre ich uloženie alebo zobrazenie, sú rôzne vstupné a výstupné zariadenia. Obecne všetky zariadenia rozdeliť na dva typy podľa spôsobu práce s údajmi. Prvým typom sú *znakové zariadenia*, ako je napr. klávesnica alebo sériová myš, ktoré poskytujú údaje po jednotlivých znakoch tak, ako sa so zariadením manipuluje. Druhým typom sú tzv. *blokové zariadenia*, ktoré vždy poskytujú celý blok informácií, napr. pevný disk, ktorý číta informácie po jednotlivých sektoroch.

Aby bolo možné správne používanie všetkých funkcií pre vstup a výstup, je potrebné na začiatku programu pripojiť hlavičkový súbor `stdio.h`, urobí sa to pomocou príkazu:

```
#include <stdio.h>
```
- tu nie je bodkočiarka !!!

Od tohto okamžiku je možné používať funkcie pre vstup a výstup.

2.3.1 Vstup a výstup znakov

Pre načítanie jedného znaku zo štandardného vstupu máme v jazyku C k dispozícii funkciu **getchar** vracajúcu číslo typu **int** a reprezentujúcu jeden znak. V prípade chyby, rovnako ako v prípade nájdenia konca súboru je vrátená hodnota `EOF`.

Funkcia zabezpečujúca opačnú operáciu – výstup jedného znaku – sa nazýva **putchar**. Deklaráciu oboch funkcií máme možnosť vidieť na nasledujúcich riadkoch:

```
int getchar(void);
int putchar(int znak);
```

Jediným parametrom funkcie putchar je znak, ktorý bude zobrazený. Návratová hodnota potom nadobúda rovnakú hodnotu ako je vypisovaný znak, a to len v prípade, že všetko prebehlo v poriadku. V opačnom prípade je návratovou hodnotou `EOF`.

Pozn.: Pri príkaze getchar musíme po stlačení klávesy stlačiť ENTER, ak sme stlačili viacej kláves po sebe, tak program zoberie len prvú.

Príklad 2
Napíšte program, ktorý prečíta znak z klávesnice a vytlačí ho na obrazovke.

```
#include <stdio.h>

int znak;

void main()
{
  znak = getchar();
  putchar(znak);
}
```

Príklad 3
Doplňte program z príkladu 2 o to, aby po vytlačení znaku na obrazovke odriadkoval text ("stlačil" ENTER).

```
#include <stdio.h>

int znak;

void main()
{
  znak = getchar();
  putchar(znak);
  putchar('\n');
}
```

Po napísaní programu vykonáme:
1) Uložíme si súbor (lepšie je ho ukladať priebežne, pretože by nám mohol zamrznúť počítač a pod., a pri dlhších programoch by sme stratili veľa času a naša vynaložená námaha by vyšla nazmar)
2) Spustíme program pomocou klávesovej skratky CTRL + F9
3) Ak kompilátor našiel program zobrazí sa chybné hlásenie (message) na spodnom okraji obrazovky s popisom chyby, ak nenašiel žiadne chyby, tak sa nám program spustí
4) Pretože program prebehol veľmi rýchlo a nič sme nevideli (len bliknutie obrazovky) použijeme klávesovú skratku ALT + F5, ktorá slúži na prechod medzi editorom a obrazovkou.

Pozn.: Toto bolo stručné zopakovanie postupu pri spúšťaní programu.

2.3.2 Formátovaný vstup a výstup

Ako programátori by sme mali veľmi ťažký život, keby sme museli napríklad čísla čítať alebo vypisovať vo forme textového reťazca, preto máme k dispozícii funkcie pre tzv. formátovaný vstup a výstup.

```
int printf(const char* format, ...);
int scanf(const char* format, ...);
```

Ako môžeme vidieť z deklarácií funkcií, obe funkcie majú premenný počet parametrov. Ten je presne určený prvým parametrom – textovým reťazcom – ktorému tiež hovoríme **formátovací reťazec**.

Formátovací reťazec funkcie **printf** môže obsahovať dva typy informácií. Jednak ide o bežné znaky, ktoré sú vytlačené na štandardný výstup a ďalej o špeciálne formátovacie sekvencie znakov začínajúce znakom percenta %. Znamená to teda, že tieto znaky predstavujú špeciálny formátovací znak. Ak majú byť vypísané ako obyčajný znak, tak musia byť zdvojene (napr. %%).

```
printf("Teraz vypiseme znak percento %% \n");
```

Súčasťou vypísaných znakov môžu byť tiež znaky reprezentované pomocou **escape sekvencie**. Začínajú znakom \ (napíšeš ho tak, že stlačíš ALT a napíšeš číslo ASCII kódu, ktoré je 92, a pustíš ALT). Príkladom môže byť prechod na nový riadok, ako vidíme na príklade - **\n**.

Prehľad všetkých escape sekvencií sa nachádza v tabuľke:

\a	zvukový signál (pípnutie)
\b	(backspace) posun kurzoru o znak naspäť
\f	(formfeed) posun na novú stránku v prompte MS-DOS
\n	(next line) prechod na ďalší riadok
\r	(carriage return) návrat kurzoru na začiatok riadku
\t	(tab) tabelátor – posun kurzoru o konštantný počet medzier (záleží na nastavení BIOSu)
\v	(vertical tab) vertikálny tabelátor - posun kurzoru o konštantný počet riadkov
\\	(backslash) vytlačenie samotného znaku \ (rovnaký dôvod ako u % v riadiacom reťazci)
\’	(quote) vytlačenie znaku ’ (potrebné v znakových konštantách kde by ’’’ bol nezmysel)
\”	(double quote) vytlačenie znaku ” (potrebné v reťazcových konštantách)
\0	(null character) vytlačenie nulového znaku (ASCII kód 0)
\aaa	vytlačenie ľubovolného znaku s kódom aaa (v osmičkovej sústave)
\xaaa	vytlačenie ľubovolného znaku s kódom aaa (v šesnástkovej sústave)

Formátovaný vstup umožňuje funkcie **scanf**. Ide opäť o funkciu s premenným počtom parametrov, ktorých počet je daný obsahom formátovacieho reťazca. Ten však nemôže byť taký bohatý ako u funkcie printf. Pri použití funkcie scanf si však musíme uvedomiť, že druhým a ďalším parametrom musí byť vždy ukazovateľ na premennú, pretože funkcia argumenty naplní načítanými hodnotami.

Formátovací reťazec funkcie scanf a printf začína znakom **percento** - %.

Nasleduje stručná tabuľka formátovacích reťazcov s % v riadiacom reťazci funkcií **printf** a **scanf**:

reťazec	typ	zobrazenie
%d %i	*int*	v desiatkovej sústave so znamienkom
%o	*unsigned int*	v osmičkovej sústave bez znamienka
%u	*unsigned int*	v desiatkovej sústave bez znamienka
%x	*unsigned int*	v šestnástkovej (hexadecimálnej) sústave s malými písmenami (a, b, c, d, e, f), bez znamienka
%X	*unsigned int*	v šestnástkovej sústave s veľkými písmenami (A,B,C,D,E,F) bez znamienka
%f	*float*	vo formáte *[-]dddd.dddd* (počet číslic a desatinných miest sa dá nastaviť)
%e	*float*	vo formáte *[-]d.dddd e [+/-]ddd* (počet desatinných miest sa dá nastaviť)
%E	*float*	rovnako ako pri *%e* , ale namiesto *e* píše *E*
%g	*float*	funkcia sa sama rozhodne, či je výhodnejšie použiť zápis *%f* alebo *%e*
%G	*float*	rovnako ako pri %g , ale namiesto *e* píše *E*
%s	*char*	reťazec znakov na vstupe oddelený medzerou od ostatných znakov
%c	*char*	jeden znak

Ak máš v tom chaos, tak sa pozri radšej na tieto príklady:

Príklady na prácu s funkciou **printf** :

1. Takto si určíš počet desatinných miest:

```
float f=1.56789;

printf("Desatinne cislo s dvomi desatinnymi miestami: %.2f", f);
```

Výstup bude:

```
Desatinne cislo s dvomi desatinnymi miestami: 1.57
```

Iste si si všimol, že sa číslo automaticky zaokrúhli. Dá sa nastaviť aj celkový počet číslic (ako číslica sa ráta aj desatinná bodka) formátom: *%6.2f* (vypíše napr. číslo 128.29). Číslo pred bodkou určuje celkový počet číslic a číslo za bodkou určuje, koľko z toho tvoria desatinné miesta. Ako si videl v prvom príklade, prvé číslo netreba uvádzať. Tiež nemusíš uvádzať druhé číslo (potom nie je nutná ani bodka) napr. *%6.f* alebo *%6f* . Ak nenapíšeš žiadne číslo (*%f*) automaticky sa použije formát *%.6f* . Pozor! Ak do premennej typu float uložíš číslo mimo jej rozsah výsledný výpis bude pochopiteľne ukazovať niečo iné.

2. Ďalšie príklady:

```
float f=1.56789;
double d=1258.12345678;

printf("%.3f , %f \n", f, f);
printf("%8lf , %lf \n", d, d);
```

Výstup bude:

```
1.568 , 1.567890
1258.12346 , 1258.123457
```

Iste si si všimol, že aj keď som obmedzil počet číslic na 8 zobrazil ich dokopy 11 (aj z desatinnou bodkou. Celú časť čísla totiž, bez ohľadu na nastavenie, vždy musí zobraziť kompletnú. Možno sa teraz opýtaš, načo je to potom dobré. Jednoducho ak zobrazuješ čísla pod sebou, môžeš ich usporiadať tak aby boli presne zarovnané. Skús si to na tomto programe:

3. Zarovnávanie desatinných čísel:

```
double d1=1258.12345678,
       d2=2.123456;  // v identifikatore mozno pouzit aj cisla

printf("%8.2lf\n%8.2lf \n", d1, d2);
```

Výstup bude:

```
1258.12
   2.12
```

4. Exponenciálny výpis:

```
double d=1258.12345678;

printf("%.2le , %le \n", d, d);
```

Výstup bude:

```
1.26e+03 , 1.258123e+03
```

Myslím, že ti to je dosť jasné. Výstup je v bežnom zápise: 1,26 .10^3 ; 1,258123 .10^3

5. Výpis reťazca:

```
printf("%s world", "Hello");
```

Výstup bude:

```
Hello world
```

Zatiaľ ešte nepoznáš typ premennej na ukladanie znakového reťazca, preto som použil reťazcovú konštantu. Tu sa dá nastaviť minimálny a maximálny počet znakov, ktoré sa majú z reťazca vytlačiť. Zápis *%5.10s* zobrazí minimálne 5 a maximálne 10 znakov reťazca.

Príklad 4

Program, ktorý vypíše text:
Pracovali na 100%

```
#include <stdio.h>

void main()
{
   printf("Pracovali na 100%%");
}
```

Príklad 5
Program, ktorý vypíše text:
James Bond \"Agent 007"\

```
#include <stdio.h>

void main()
{
   printf("James Bond \\\"Agent 007\"\\");
}
```

V týchto programoch boli použité špeciálne formátovacie znaky (\, **%**, **"**). Ako píšeme percento sme si už ukázali, ale ak chceme vypísať znak ako je úvodzovky " alebo lomítko \ musíme pred tieto znaky napísať lomítko \.

Príklad 6
Program vypíše súčet a súčin čísiel 7 a 4:

```
#include <stdio.h>

void main()
{
   int i=7, j=4;

   printf("Sucet je %d\tSucin je %d\n", i+j, i*j);
}
```

Príklad 7
Program prečíta z klávesnice dve čísla, vytlačí ich v obrátenom poradí, a potom vytlačí ich súčet.

```
#include <stdio.h>

void main()
{
  int i, j;

  scanf("%d", &i);
  scanf("%d\n", &j);
  printf("Obratene: %d%d\n", j, i);
  printf("Sucet je %d\n", i+j);
}
```

Príklad 8
1) `printf("Znak '%c' ma ASCII kod %d (%XH)\n", c, c, c);`
   vypíše: `Znak 'A' ma ASCII kod 65 (41H)`

2) `printf("Znak '%c' ma ASCII kod %d (%XH)\n", '*', '*', '*');`
   vypíše: `Znak '*' ma ASCII kod 42 (2AH)`

3) `printf("Je presne %2d:%2d\n", hodiny, minuty);`
   vypíše napr.: `Je presne  1:12`
   alebo napr.: `Je presne 13: 3`

- pretože počet cifier, ktorý sa bude tlačiť, sa dá presne určiť

4) `printf("Je presne %02d:%02d\n", hodiny, minuty);`
vypíše napr.: `Je presne 01:12`
alebo napr.: `Je presne 05:03`
- rozdiel oproti predchádzajúceho príkladu je v tom, že vo formátovacom reťazci doplníme číslo 0

Cvičenie

1) Napíšte program, ktorý prečíta znak a vypíše znak s ASCII hodnotou o jednu vyššiu, napr.:
vstup: `A`
výstup: `B (ASCII 66)`

2) Napíšte program, ktorý prečíta celé desiatkové číslo od (v rozsahu 0 až 255) a vypíše jeho hexadecimálnu hodnotu, napr.:
vstup: `127`
výstup: `7Fh`

3) Napíšte program, ktorý pripočíta 20% daň, napr.:
vstup: `Zadajte cenu bez dane: 100 Sk`
výstup: `Predajna cena s danou (20%): 125 Sk`

4) Napíšte program, ktorý vypočíta obsah štvorca, napr.:
vstup: `Zadajte stranu stvorca: 4`
výstup: `Obsah stvorca: 16`

5) Napíšte program, ktorý vypočíta obsah obdĺžnika, napr.:
vstup:
```
Zadajte dlzku: 5
Zadajte sirku: 4
```
výstup: `Obdlznik o dlzke 5 a o sirke 4 ma obsah 20`

6) Napíšte program, ktorý prečíta veľké písmeno a vypíše malé písmeno, napr.:
vstup: `Zadaj pismeno: A`
výstup: `a`

7) Napíšte program, ktorý prečíta malé písmeno a vypíše veľké písmeno, napr.:
vstup: `Zadaj pismeno: b`
výstup: `B`

Najčastejšie chyby v doteraz použitých príkazoch:

- `main();` - za funkciou nemá byť bodkočiarka
- `printf("Text")` - za funkciou musí byť bodkočiarka
- `printf("%d", i, j);` - veľa argumentov
- `printf("%d%d", i);` - málo argumentov
- `scanf("%d", i);` - chýba znak &, správne je teda `scanf("%d", &i);`
- `scanf("%d", &c);` - formát pre **char** je %c, správne je teda `scanf("%c", &c);`

2.4 Aritmetické výrazy

V aritmetických výrazoch je nutné si pripomenúť, že výraz ukončený bodkočiarkou sa stáva príkazom, napr.:

```
i = 2       - výraz s priradením
i = 2;      - príkaz
```

Skupinu aritmetických výrazov (operátorov) tvoria známe operátory +, -, *, / a ďalej potom operátor MODULO, pre ktorý sa používa symbol percento %.

2.4.1 Unárne operátory

Unárny operátor mínus -

Výsledkom je negácia jeho operátoru. Hodnota negácie bez znamienka je vypočítaná tak, že hodnota výrazu je odčítaná od hodnoty 2*n, kde n je počet bitov čísla typu int.

Operátor logickej negácie - !

Výsledkom operátoru logickej negácie **" ! "** je hodnota:

- 1, ak hodnota operandu je 0
- 0, ak hodnota operandu je 1

Typ výsledku je int.

Inkrement ++ a dekrement --

Patria medzi špeciálne unárne operátory. Inkrement znamená zväčšenie hodnoty o 1 a dekrement znamená zmenšenie hodnoty o 1.

Zapisuje sa:

```
x++   to isté ako   x = x + 1
x--   to isté ako   x = x - 1
```

Treba pripomenúť, že tieto dva operátory pracujú len s premennou, nie s výrazom! Operátory ++ alebo -- môžu byť pred premennou (++x), aj za premennou ako sme si už ukázali. V oboch prípadoch sa vykoná rovnaká operácia, ale ak je operátor pred premennou (++x), operácia sa vykoná pred vyhodnotením výrazu a ak je operátor za premennou (x++), operácia sa vykoná po vyhodnotení výrazu (teda do výrazu je ešte počítaná pôvodná hodnota premennej).

Ostatné priradzovacie operátory

Ostatné priradzovacie operátory sú vypísané v tabuľke aj s ich rozvinutým zápisom:

operátor	príklad	rozvinutý zápis
+=	x += 5	x = x + 5
-=	x -= 2	x = x – 2
*=	x *= 4	x = x * 4
/=	x /= 6	x = x / 6
%=	x %= 3	x = x % 3

2.4.2 Bitové operátory

Bitový súčin - & (AND)

Bitový súčin je i-tý bit výsledku bitového súčinu x & y. Bude 1, pokiaľ i-tý bit x a i-tý y budú 1, inak bude 0. Pre lepšie pochopenie si pozri tabuľku:

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

Teda jednotlivé bity výsledku budú závisieť na jednotlivých bitoch.

```
unsigned int x = 0, y = 1, z;
z = x & y;   /* z = 0 */

unsigned int x = 0, y = 0, z;
z = x & y;   /* z = 0 */

unsigned int x = 1, y = 1, z;
z = x & y;   /* z = 1 */
```

Z tabuľky a z príkladov vyplýva, že bitový súčin bude vtedy rovný 1, keď budú obidva bity rovné 1.

Bitový súčet - | (OR)

Bitový súčet je i-tý bit výsledku súčtu x | y (znak | má ASCII hodnotu 124). Bude 1, pokiaľ i-tý bit x alebo i-tý bit y bude 1. Pre lepšie pochopenie si pozri tabuľku:

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

```
unsigned int x = 0, y = 1, z;
z = x & y;   /* z = 1 */

unsigned int x = 0, y = 0, z;
z = x & y;   /* z = 0 */

unsigned int x = 1, y = 1, z;
z = x & y;   /* z = 1 */
```

Z tabuľky a z príkladov vyplýva, že bitový súčet bude vtedy rovný 1, keď bude aspoň jeden bit rovný 1.

Bitový exkluzívny súčet - ^ (XOR)

Bitový exkluzívny súčet je i-tý bit výsledku bitového exkluzívneho súčtu x ^ y. Bude 1, pokiaľ i-tý bit x sa nerovná i-tému bitu y. Ak budú obidva bity 0 alebo 1, bude výsledok 0.

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

2.4.3 Binárne operátory

sčítanie	+
odčítanie	-
násobenie	*
delenie	/
modulo	%

Delenie (podiel) - /

Pri použití operátoru delenia na operátoroch celočíselného typu sa prevádza tzv. celočíselné delenie. Výsledkom je potom iba celá časť podielu a desatinná časť je odrezaná, nezávisle od jej veľkosti (nejde teda o zaokrúhľovanie).

Napr.:

13 / 6 = 2
6 / 6 = 1
1 / 2 = 0

Aby delenie prebiehalo aj s výpočtom desatinnej časti, musí byť aspoň jeden z operandov reálne číslo.

int / int	- celočíselné
int / float	- reálne
float / int	- reálne
float / float	- reálne

Modulo - %

Úlohou operátora Modulo je zistiť, aký je veľký zvyšok po celočíselnom delení dvoch čísel.

```
int x = 9, y = 5;

void main ()
{
  printf ("%d / %d = %d a zvysok je %d", x, y, x / y, x % y);
}
```

Výstup programu bude:

```
9 / 5 = 1 a zvysok je 4
```

2.5 Booleovské výrazy

Slúžia k porovnaniu operátorov, teda zisteniu, či sú ich hodnoty rovnaké alebo odlišné. Skupinu Booleovských operátorov tvoria tieto operátory:

menší	<
väčší	>
menší alebo rovný	<=
väčší alebo rovný	>=
rovný	==
nerovnosť	!=
logický súčin (AND)	&&
logický súčet (OR)	||

Dôležité si je uvedomiť rozdiel medzi porovnaním == a priradením =, a to:

`x = 5` - je celočíselný výraz s hodnotou 5, ktorá sa priradí do premennej `x` a zmení jej pôvodnú hodnotu

`x == 5` - celočíselný výraz poskytujúci 1 (TRUE = pravda), ak má premenná `x` hodnotu 5, alebo poskytujúcu 0 (FALSE), ak má premenná `x` hodnotu inú ako 5
- ani v jednom prípade (porovnania!!!) sa hodnota premennej `x` nezmení

Tabuľka priorít vyhodnocovania niektorých operátorov

operátor		smer vyhodnocovania
`! ++ -- - +`		sprava doľava
`* / %`		zľava doprava
`+ -`		zľava doprava
`< > <= >=`		zľava doprava
`== !=`		zľava doprava
`&`	bitový AND	zľava doprava
`^`	bitový XOR	zľava doprava
`|`	bitový OR	zľava doprava
`&&`	logický AND	zľava doprava
`||`	logický OR	zľava doprava
`= += -= *= atď.`		sprava doľava

2.6 Zásady pri písaní zložitejších programov

Keďže sa o chvíľu začneme zaoberať s príkazmi, ktoré vytvárajú bloky a môžu byť aj trochu dlhšie (niekoľko riadkov prípadne aj strán) a neprehľadné pri našom doterajšom písaní programov, tak si teraz napíšeme pár dôležitých zásad, ktoré by sme mali dodržiavať pri písaní programu, aby sme sa v našich programoch ako tak vyznali:

- každé vnorenie logicky podriadeného úseku programu je o 2 medzery posunuté doprava (dobre je si nastaviť tabulátor v editore, ktorý používame, aby sa posúval o 2 medzere).
- na jednom riadku je len jeden príkaz!
- otváracia zátvorka **{** je na tom istom riadku (väčšinou oddelená medzerou) ako príkazy: `if`, `else`, `for`, `while`, `do`, `switch`, `for` ...
- za otváracou zátvorkou **{** nie je bodkočiarka!
- uzatváracia zátvorka **}** je u týchto príkazov na samostatnom riadku v stĺpci, kde začína kľúčové slovo!

```
if(y>1){
  if(x < 10){
    i++;
    printf("Ahoj");
  }
}
```

- pokiaľ nasledujú za sebou dve zátvorky **(** alebo **)**, tak ich radšej oddelíme jednou medzerou, aby sme videli, že sú tam dve

```
if( (x>=12)&&(y<=3) ){
  printf("%c, %d", x, y);
}
```

- prázdne riadky vkladáme podľa vlastného uváženia tam, kde zlepšujú čitateľnosť programu

Viem, že sa zdajú tieto zásady, ktoré by sme mali dodržiavať, nepotrebné, ale ich význam dobre pochopíme až keď budeme písať dlhšie a zložitejšie programy. Vtedy zistíme, že orientovať sa v našom programe, ktorý má niekoľko strán, nie je také jednoduché ako sa nám na prvý pohľad zdá, potom oceníme dodržiavanie našich zásad. Z vlastných skúsenosti Vám odporúčam dodržiavať tieto zásady už od začiatku nášho programovanie, lebo neskôr sa ich už nebudeme mať čas ani chuť naučiť.

2.7 Príkazy na vetvenie programu

Príkazy na vetvenie programu špecifikujú v akom poradí budú inštrukcie programu vykonané

Príkazy a bloky

Výraz ako je x=0 alebo i++ alebo printf(...) sa stáva príkazom, ak je ukončený bodkočiarkou.

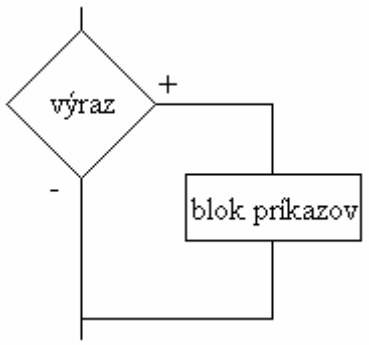
```
x=0;
i++;
printf(...);
```

Zložené zátvorky **{** a **}** sa používajú pre združovanie príkazov a deklarácií v zložených príkazoch alebo blokoch, ktorý je syntaktický rovnocenný jednému príkazu.

Jedným príkladom sú zložené zátvorky okolo jedného príkazu. Ďalším príkladom sú tieto zátvorky okolo niekoľkých príkazov za príkazom `if`, `else`, `while`, `do`, `switch` alebo `for`

2.7.1 Príkaz if

```
if(výraz){

  blok príkazov

}
```



Je to podmienkový príkaz.

Výraz je vyhodnotený, a ak je "pravdivý" (t.j. má nenulovú hodnotu), tak sa vykoná *blok príkazov*. Ak je vyhodnotený ako "nepravdivý", tak program pokračuje ďalej a nič sa nedeje.

Príklad 9

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  int a, b;

  clrscr();
  printf("Zadaj cislo a: ");
  scanf("%d", &a);
  printf("\nZadaj číslo b: ");
  scanf("%d", &b);
  if(a<b){
    printf("Cislo a=%d je mensie ako cislo b=%d", a,b);
  }
  getch();
}
```

Na začiatku programu si si asi všimol, že sme použili jednu knižnicu navyše. Použili sme ju preto, aby sme mohli používať dva nové príkazy, ktoré sú napísané v tejto knižnici, a to príkaz `clrscr` a `getch`. Za obidvoma sa píšu prázdne okrúhle zátvorky a bodkočiarka!

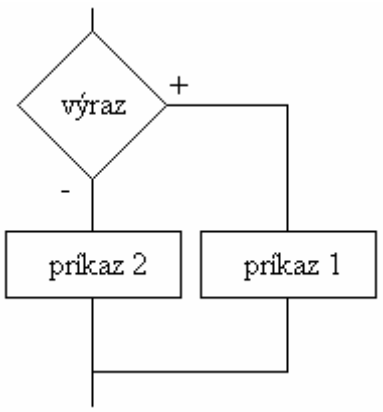
Príkaz `clrscr` slúži na vymazanie obrazovky v textovom režime (neskôr, keď budeme pracovať s grafikou, tak budeme používať iný príkaz na vymazanie obrazovky v grafickom režime, ale všetko postupne).

Príkaz `getch` slúži na pozastavenie programu. Príkaz zabezpečí, aby program počkal na stlačenie nejakej klávesy, po stlačení ľubovoľnej klávesy program pokračuje ďalej.

Program najskôr načíta dve čísla, a potom ich porovnáva pomocou príkazu `if`, ak je číslo **a** menšie ako **b**, tak vypíše text pomocou príkazu `printf`. Ak je číslo **a** väčšie alebo je rovnaké ako číslo **b**, tak sa nevypíše nič, pretože nie je splnená podmienka. Program ďalej pokračuje a počká na stlačenie ľubovoľnej klávesy, a potom skončí.

2.7.2 Príkaz if – else

```
if(výraz){
  príkaz1
}

else{
  príkaz2
}
```



Výraz je vyhodnotený a ak je "pravdivý" (t.j. má nenulovú hodnotu), tak sa vykoná *príkaz 1*. Ak je vyhodnotený ako "nepravdivý", tak sa vykoná *príkaz 2*.

Príklad 10

Program testuje či je stlačená klávesa malé písmeno.

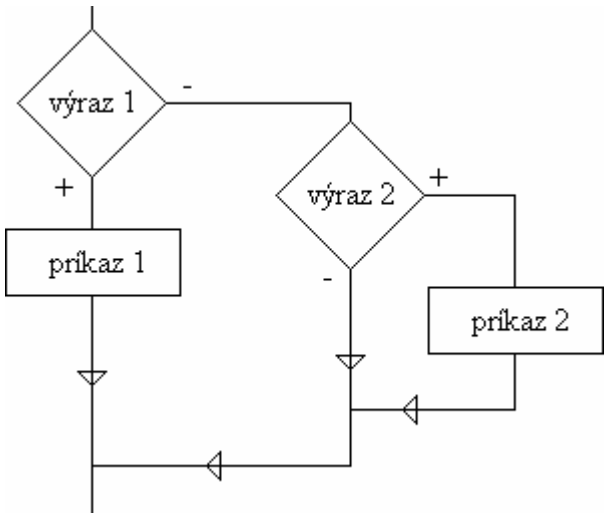
```
#include<stdio.h>
#include<conio.h>

void main()
```

```
{
  int c;

     clrscr();

     c  = getch();
     if( (c>='a') && (c<='z') ){
       printf("Stlacil si male %c", c);
     }
     else{
       printf("Nestlacil si male pismeno!");
  }
  getch();
}
```

Príkaz `getch` môžeme použiť aj na zachytenie ASCII kódu tak, že priradíme stlačenú klávesu premennej **c**. Potom pomocou podmienky testujeme či je stlačená klávesa malé písmeno. Zrejme tejto podmienke dosť dobre nerozumieš. Príkaz `if` testuje či je číslo (ASCII hodnota), ktoré je uložené v premennej **c**, z intervalu od a po z. Namiesto `'a'` by sme mohli napísať aj číslo (ASCII hodnotu písmena a) 97. Podobne by sme mohli napísať namiesto `'z'` číslo 122. Takže ešte raz, ale trochu inak. Príkaz if testuje či stlačený znak (číslo) je z intervalu od 97 po 122, ak nie je, tak príkaz `printf`, ktorý je za príkazom `else` vypíše text: `"Nestlacil si male pismeno!"`.

2.7.3 Príkaz if – else if

```
if(výraz1){

  príkaz1

}

else if(výraz2){

  príkaz2

}
```



Výraz 1 je vyhodnotený a ak je "pravdivý" (t.j. má nenulovú hodnotu), tak sa *príkaz 1* vykoná. Ak je vyhodnotený *výraz 1* ako "nepravdivý", tak sa vyhodnotí *výraz 2* a ak je "pravdivý", tak sa vykoná *príkaz 2*. Ak je vyhodnotený *výraz 2* ako "nepravdivý", tak sa nevykoná nič a program pokračuje ďalej.

Príklad 11

Program testuje stlačenú klávesu, či je to malé písmeno alebo veľké a podľa toho vypíše text na obrazovku.

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  int c;

      clrscr();

      c  = getch();
      if( (c>='a') && (c<='z') ){
        printf("Stlacil si male %c", c);
      }
      else if( (c>='A') && (c<='Z') ){
        printf("Stlacil si velke %c", c);
  }
  getch();
}
```

Príkaz `if` testuje či je číslo (ASCII hodnota stlačeného znaku), ktoré je uložené v premennej **c**, z intervalu od a po z. Ak nie je stlačený znak malé písmeno, tak sa pomocou príkazu `else if` testuje či je stlačené písmeno veľké. Ak nie je písmeno ani malé ani veľké, tak z toho vyplýva, že stlačený znak nie je písmeno. Program nič nevypíše a čaká na stlačenie ľubovoľnej klávesy.

Skúste porozmýšľať akoby ste spravili program (bez toho, aby ste sa pozreli na hotový program, kde je jedno možné riešenie programu, a prípadne vymyslite ten istý program trochu inak) rovnaký ako predchádzajúci len s tým, že by ste ošetrili poslednú časť tak, že keď nestlačíme ani malé ani veľké písmeno, aby vypísal, že si nestlačil žiadne písmeno, ale stlačil si znak...

Príklad 12

Celý program je skoro ten istý ako predtým len tam pridáme ešte jedno `else`, ktoré nám zabezpečí vypísanie textu: "`Nestlacil si pismeno, ale znak ...`". Je to jednoduché na pochopenie, keď nie je splnená prvá podmienka (nie je to malé písmeno), tak sa testuje či je to veľké písmeno a keď nie je splnená ani druhá podmienka (nie je to veľké písmeno), tak nám z toho vychádza, že to nie je písmeno (sú to všetky ostatné znaky, ktoré sa nachádzajú v tabuľke ASCII hodnôt, napr. {}-*<>+/ß$×÷012, ale aj klávesy ako ENTER, TABulátor, CAPS LOCK, SPACE BAR, BACK SPACE, ESC, F1÷F12, INSERT, DELETE, HOME, END, PAGE UP, PAGE DOWN, NUM LOCK, CTRL, ALT, ...). Takže nám stačí pridať len jeden príkaz `else` bez podmienky, pretože keď nie sú splnené dve predchádzajúce podmienky, tak máme istotu, že stlačená klávesa nie je písmeno.

To by už ja stačilo, teraz ten spomínaný program:

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int c;

  clrscr();

  c  = getch();
  if( (c>='a') && (c<='z') ){
    printf("Stlacil si male %c", c);
  }
```

```
  else if( (c>='A') && (c<='Z') ){
    printf("Stlacil si velke %c", c);
  }
  else{
    printf("Nestlacil si pismeno, ale znak %c", c);
  }
  getch();
}
```

Poznámka k príkazom if, ...

Teraz by som chcel poukázať ešte na jednu vlastnosť jazyka C, a to:

1)

```
#include<stdio.h>

void main()
{
  int z;

  z = getchar();
  if( (z>='a') && (z<='z') ){
    printf("Stlacil %c", z);
  }
}
```

2)

```
#include<stdio.h>

void main()
{
  int z;

  if((z = getchar()>='a')&&(z<='z')){
    printf("Stlacil %c", z);
  }
}
```

Priradenie môžeme teda realizovať aj priamo v podmienke, ušetrí nám to jeden riadok, ale môže to pôsobiť nejaké problémy, keď budeme mať dlhší program a budeme to chcieť nájsť. Je na Vás ako sa rozhodnete, ktorý spôsob budete používať.

2.7.4 Príkaz switch

Jazyk C obsahuje príkaz prepínač alebo príkaz pre viacnásobné vetvenie programu.

```
switch(premenna){
  case hodnota 1: príkaz 1; break;
  case hodnota 2: príkaz 2; break;
        .
        .
        .
  case hodnota n: príkaz n; break;
```

```
    default: príkaz def; break;          - príkaz break tu už ani nemusí byť
}
```

Pri príkaze `switch` platia tieto **zásady**:
- výraz podľa, ktorého sa rozhoduje musí byť typu int
- každá vetva prepínača musí byť ukončená príkazom `break`
- je podporovaná vetva `default` (vykoná sa vtedy, keď žiadna z vrstiev nevyhovuje)
- v každej vetve môže byť viac príkazov, ktoré nemusíme uzatvárať do zátvoriek

Ak nie je vetva prepínača (jeden alebo viac príkazov) ukončená pomocou príkazu `break`, program neopúšťa `switch`, ale spracováva nasledujúcu vetvu v poradí! A v tejto činnosti pokračuje až po najbližší príkaz `break` alebo po ukončenie príkazu `switch`.

Zaužívaná kultúra a rady

- súvisiace vetvy sa neoddeľujú novým riadkom, napr.:

```
switch(getchar()){
  case 'a':
  case 'b':
    putchar('1');
    break;

  default:
    break;
}
```

- vetva `default` sa väčšinou píše aj keď je prázdna, teda:

```
  default:
    break;
```

- príkaz `break` za posledným príkazom poslednej vetvy nie je nutný, ale väčšinou sa píše
- vetva `default` nemusí byť uvedená ako posledná vetva prepínača, ale väčšinou sa píše posledná
- príkaz `break` ruší vždy najvnútornejší cyklus alebo ukončuje príkaz `switch`. Je teda nutné dávať zvýšený pozor, ak `switch` obsahuje nejaký cyklus alebo naopak
- príkaz `continue` nemá s príkazom `switch` nič spoločné!
- príkazy vo vetve `default` sa vykonávajú vždy až vtedy, keď nie je nájdená žiadna vyhovujúca vetva, nech je vetva `default` uvedená v prepínači kdekoľvek

```
switch(getchar()){
  default:
    printf("Nestlacil si ani 1 ani 2\n");
    break;

  case '1':
    printf("Stlacil si 1\n");
    break;

  case '2':
    printf("Stlacil si 2\n");
    break;
}
```

Príklad 13

Nasledujúci program, ktorý je realizovaný pomocou prepínača, rozlišuje akú klávesu si stlačil. Ak si stlačil 0, tak vypíše, že si stlačil 0. Ak si stlačil číslo 1, 2 alebo 3, tak vypíše: "`Stlacil si cislo od 1 po 3, presnejsie ...`". Ak si stlačil číslo 4, 5, 6, 7, 8 alebo 9, tak vypíše: "`Stlacil si cislo od 4 po 9, presnejsie ...`". Pokiaľ si nestlačil žiadne číslo, tak ti program vypíše text: "`Nestlacil si cislo!`"

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int c;

  clrscr();

  c = getch()
  switch(c){
    case '0':
      printf("Stlacil si nulu - 0");
      break;

    case '1':
    case '2':
    case '3':
      printf("Stlacil si cislo od 1 po 3, presnejsie %c", c);
      break;

    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
      printf("Stlacil si cislo od 4 po 9, presnejsie %c", c);
      break;

    default:
      printf("Nestlacil si cislo!");
      break;
  }
  getch();
}
```

2.8 Príkazy break a continue

Obidva tieto príkazy môžeme použiť vo všetkých cykloch, ktoré si ukážeme v ďalšej kapitole. Obidva nejakým spôsobom ovplyvňujú "normálny" priebeh cyklu.

2.8.1 Príkaz break

Tento príkaz sme už používali pri prepínači. Takže by malo byť jasné načo slúži, ale ak nie tak to ešte trochu zhrniem.

Príkaz `break` hovorí, že beh programu nemá pokračovať nasledujúcim riadkom, ale až prvým príkazom, ktorý sa nachádza za cyklom (prípadne za ukončovaciu zátvorkou } príkazu `switch`). Príkazy, ktoré sa v cykle nachádzajú za príkazom `break`, sa už nevykonajú.

Príkaz `break` ukončuje vždy najvnútornejší neuzavretý cyklus a okamžite opúšťa cyklus!

2.8.2 Príkaz continue

Príkaz `continue` môže byť použitý iba v cykloch!!!

Úlohou príkazu `continue` je ukončiť práve prebiehajúcu priebeh cyklu a začať priebeh nový. Niekedy sa tiež používa pre popis jedného priebehu cyklu pojem ITERACE. Často tiež hovoríme, že príkaz `continue` začne novú iteraciu.

Ak je niekoľko cyklov vnorených do seba, príkaz `continue` spôsobí novú iteraciu v najbližšom cykle.

Príkaz continue skáče na koniec najvnútornejšieho neuzavretého cyklu, a tým si vynúti ďalšiu iteraciu cyklu – cyklus neopúšťa!

2.9 Príkazy na cyklenie programu

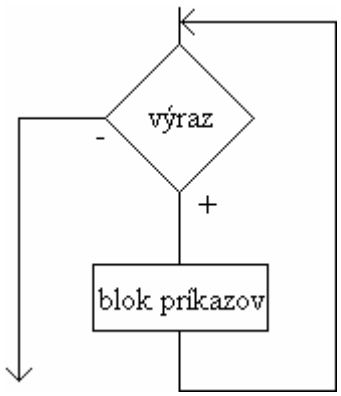
Pomocou cyklu realizujeme opakovanie určitého príkazu v závislosti na logickej hodnote zadaného výrazu.

2.9.1 Cyklus while

Cyklus `while` je prvým z troch typov cyklov, ktoré si ukážeme. Cyklus `while` má výraz (podmienku) určujúci jeho životnosť uvedenú hneď za kľúčovým slovom `while` v okrúhlych zátvorkách. Tento interačný príkaz testuje podmienku cyklu pred tým ako cyklus začne!!! Cyklus teda nemusí prebehnúť ani raz. Cyklus prebehne, keď je výraz (podmienka) splnená.

Všeobecne vyzerá cyklus `while` takto:

```
while(výraz){

  blok príkazov

}
```



Príklad 14

Program prečíta znaky z klávesnice, tlačitelné znaky vypíše na obrazovku, neviditeľné si nevšíma a zastaví sa po prečítaní znaku "z" a stlačení ENTER.

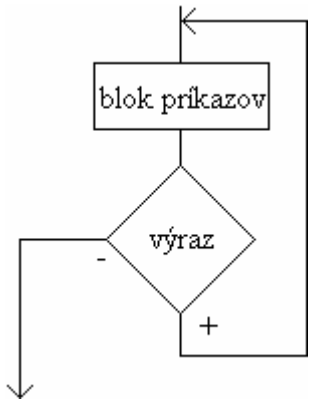
```
#include<stdio.h>
#include<conio.h>

void main()
{
  int c;
```

```
  clrscr();

  while( (c=getchar()) < 'z'){
    if(c>='_') putchar(c);
  }

  printf("\n Citanie znakov bolo ukoncene");
  getch();
}
```

2.9.2 Cyklus do – while

V tomto cykle sa podmienka testuje až po prechode cyklom (po vykonaní cyklu), to znamená, že aj keď nie je splnená podmienka, tak cyklus prebehne aspoň raz. Cykly sa vykonávajú do vtedy, pokiaľ výraz má hodnotu 1 (pravda = je splnená podmienka).

```
do {

  blok príkazov;

} while(výraz);
```



Príklad 15

Výpočet priemernej hodnoty z dopredu známeho počtu prvkov.

Program vypočíta priemernú známku žiakov. Počet žiakov si program vypýta na začiatku programu.

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int i=0;
  float znamka, priemer,c = 0, pocet;

  clrscr();
  printf("\n Zadaj pocet ziakov: ");
  scanf("%g", &pocet);                // nacita pocet ziakov

  do{
    i++;
    printf("\n Zadaj znamku %d. ziaka: ", i);
    scanf("%g", &znamka);             // nacita znamku
    c += znamka;                      // pripocita znamku k premennej c
  } while(i < pocet);

  priemer = c / pocet;                // vypocita priemer znamok
```

```
  printf("\n Priemerna znamka je: %3.2f", priemer);
  getch();
}
```

Príklad 16

Program vypočíta obvod a obsah obdĺžnika.

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a, b, c;

  do{
    clrscr();              // vymaze obrazovku

    printf("\n Zadaj stranu a:");
    scanf("%d", &a);                  // nacita stranu a
    printf("\n Zadaj stranu b:");
    scanf("%d", &b);                  // nacita stranu b

    printf("\n Obsah obdlznika je %d", a*b); //vypise obsah
    printf("\n Obvod obdlznika je %d", (a+b)*2);//vypise obvod

    printf("\n Stlac p alebo P, ak chces pocitat znovu");
    c = getch();                      // nacita stlacenu klavesu
  } while((c == 'p') || (c == 'P'));
}
```

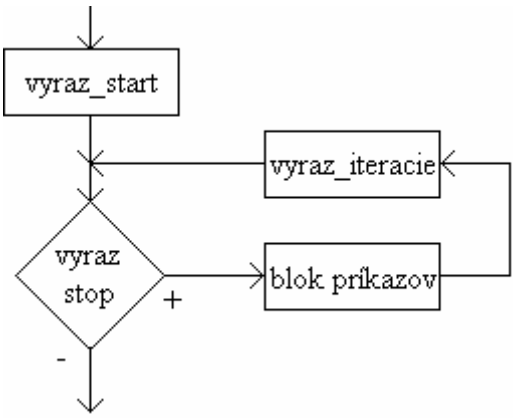
Poznámky:

- ak príkaz `do - while` obsahuje zložený príkaz, je vhodné písať príkaz `while` za uzatváracou zátvorkou, ako je vidieť na príklade 1
- ak je možné, tak radšej použijeme cyklus `while` alebo `for` pred cyklom `do - while`, pretože sú prehľadnejšie

2.9.3 Cyklus for

Ide o typický príkaz cyklu, ktorý použijeme v prípade, že poznáme vopred počet opakovaní – priechodov cyklom.

```
for (vyraz_start; vyraz_stop; vyraz_iteracie) {

  blok príkazov;

}
```



Teraz si to skúsime trochu vysvetliť na konkrétnom prípade, napr.:

```
for (i = 1; i < 10; i++)
  printf("%d\n", i);
```

Príkaz funguje tak, že do premennej `i` sa priradí 1 (`i = 1`). Program ďalej testuje či je podmienka cyklu splnená (`i < 10`). Ak je podmienka splnená, tak sa vykoná príkaz `printf`, ktorý vypíše hodnotu premennej i a od riadkuje. Potom sa vykoná výraz iterácie (`i++`, ak si zabudol tak je to to isté ako `i = i + 1`). Ten zvýši hodnotu premennej `i` o 1. Potom sa testuje, či je podmienka splnená, ak áno tak sa začne vykonávať ďalší. Ak nie je podmienka splnená, program pokračuje ďalej za cyklom.

Príklad 17

Program vypíše prvých 25 znakov z ASCII tabuľky aj s ich ASCII hodnotou.

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int i;

  clrscr();                // vymaze obrazovku

  for(i=1; i<=25; i++) {
    printf("%c = %d\n", i, i);
  }

  getch();
}
```

Poznámka:

- Aby sme sa vyhli prípadným problémom so zátvorkami, tak si radšej zvyknime písať zložené zátvorky **{ }** aj keď máme v cykle len jeden príkaz, pretože ak pridáme ďalšie príkazy do cyklu, tak môžeme zabudnúť pridať aj zložené zátvorky, a potom by nám program nefungoval, tak ako chceme.

2.10 Príkazy goto, return a exit

2.10.1 Príkaz goto

Príkaz `goto` sa v dobre napísaných programoch používa málokedy, pretože v štruktúrovanom jazyku ako je C sa mu môžeme vyhnúť. Ak sa predsa použije majú byť na to seriózne dôvody.

Jedno z mála seriózne použitých použití je výskok z vnorených cyklov (pozri sa na príklad).

Príkaz goto nemusíme dopredu definovať

Jedno z klasických použití príkazu `goto`. Využíva sa preto, že zjednodušuje a sprehľadňuje program.

```
for( i = 0; i < 10; i++) {
  for( j = 0; j < 10; j++) {
    for( k = 0; k < 10; k++) {
      if( y == 0 )
        goto error;
      a[i] = a[i] + b[i] / y;
    }
  }
}
goto dalsi_vypocet;
error:
printf("Chyba \n");
dalsi_vypocet:
...
```

Program funguje tak, že sa začne vykonávať najskôr cyklus **i** potom **j** a **k**. Cyklus **i** sa vykoná 10-krát a v rámci jednej iterácie cyklu **i** sa cyklus **j** vykoná tiež 10-krát a v rámci jednej iterácie cyklu **j** sa cyklus **k** vykoná tiež 10-krát.

Asi si sa ti zdá toto vysvetlenie trochu nepochopiteľné, na 100% to pochopíš až keď budeš písať svoje vlastné programy. Ešte dodám, že cyklus **i** sa spolu vykoná 10krát, cyklus **j** 10*10-krát a cyklus **k** 10*10*10-krát.

Predchádzajúca veta nemusí byť vždy pravdivá, pretože ak premenná y sa rovná 0, tak sa vykoná príkaz `goto`, ktorý vyskočí z cyklu a skočí na návestie `error`. Tým zabezpečíme, aby sa premenná y nerovnala 0, pretože 0 nemôžeme deliť.

Pokiaľ sa premenná y nebude ani raz rovnať 0, tak sa cykly dokončia bez prerušenia a program pokračuje ďalším príkazom `goto`, ktorý skočí na návestie `dalsi_vypocet`. Jedinou úlohou tohto príkazu je teraz preskočiť návestie `error`.

2.10.2 Príkaz return

Ak dôjde program na príkaz `return` ukončí sa práve prebiehajúca funkcia, ktorá tento príkaz obsahuje.

Vo funkcii `main()` ukončí príkaz `return` celý program. Často sa pomocou príkazu `return` vracia nejaká hodnota, ktorej typ záleží na typu funkcie.

Ak použijeme predchádzajúci program prevedený volaním funkcie, tak je potom výhodnejšie použiť príkaz `return` a nie `goto`. Pre zistenie, či funkcia splnila svoju úlohu

správne, nám poslúži návratová hodnota. Ak je hodnota 1 bude znamenať prevedenie bez chyby a hodnota 0 prevedenie s chybou.

```
nasobenie()
{
  int i, j ,k;

  for( i = 0; i < 10; i++) {
    for( j = 0; j < 10; j++) {
      for( k = 0; k < 10; k++) {
        if( y == 0 )
          return(0);              // neuspesne
        a[i] = a[i] + b[i] / y;
      }
    }
  }
  return(1);                      // uspesne
}
```

2.10.3 Príkaz exit

Občas sa môžeme stretnúť s použitím funkcie `exit`. Ta má podobný význam ako príkaz `return`. Rozdiel je v tom, že ak je vyvolaná z ktorejkoľvek funkcie, ukončí bezprostredne program bez vrátenia do volajúcej funkcie.

2.11 Textový režim

V textovom režime je možné pracovať s celou obrazovkou alebo s jej časťou, ktorú volíme príkazom `window`. Všetky funkcie a symbolické konštanty pre prácu s obrazovkou sú deklarované v hlavičkovom súbore conio.h

2.11.1 Práca s klávesnicou

Teraz si trochu zopakujeme príkazy, ktoré sme si už spomínali, ale opakovanie je matka múdrosti.

`getch();`
Vracia znak prečítaný z klávesnice – čaká na stlačenie klávesy

`getche();`
Pracuje ako getch, ale na viac vstupujúci znak automaticky zobrazí

`cscanf("format", ...);` - napr. `cscanf("%d", &x);`
Načíta zadaný znak (cislo) do premennej, ktorú si zadáme...

`cgets(char *str);`
Čítanie reťazca znakov z klávesnice do poľa

2.11.2 Riadiace služby

Parameter	Mód	Význam
LASTMODE	-1	Predchádzajúci textový mód
BW40	0	Čierno-Biely 40 stĺpcov
C40	1	Farebných 40 stĺpcov
BW80	2	Čierno-Biely 80 stĺpcov
C80	3	Farebných 80 stĺpcov
MONO	7	Monochromatických 80 stĺpcov
C4350	64	EGA a 43 riadkov VGA 50 riadkov

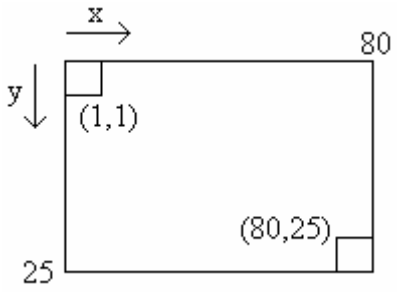
Pomocou príkazu `textmode(parameter)` si môžeme zvoliť textový mód našej obrazovky. Väčšinou sa to, ale nepoužíva, aspoň ja osobne som sa s tým nestretol.

Príklad 18

V nasledujúcom program si môžeme overiť funkčnosť textových módov.

```
#include <conio.h>

void main()
{
  clrscr();

  textmode(BW40);
  cprintf("ABC");
  getch();

  textmode(C40);
  cprintf("ABC");
  getch();

  textmode(BW80);
  cprintf("ABC");
  getch();

  textmode(C80);
  cprintf("ABC");
  getch();

  textmode(MONO);
  cprintf("ABC");
  getch();
}
```

Štandardne je nastavený príkaz `textmode(C80)`:




```
window(int left, int top, int right, int bottom);
```

Nastaví na obrazovke okno pre výstup všetkých služieb pre prácu s obrazovkou v textovom režime.

ľavý horný roh [left, top]
pravý dolný roh [right, bottom]

Štandardne je obrazovka nastavená na

```
window (1,1,80,25);
```

2.11.3 Nastavenie atribútov textu

Tu si ukážeme ako môžeme text napísať určitou farbou a na určitom pozadí.

Číslo	Farba - ENG	Farba - SK
0	BLACK	čierna
1	BLUE	modra
2	GREEN	zelena
3	CYAN	modrozelená
4	RED	červená
5	MAGENTA	fialová
6	BROWN	hneda
7	LIGHT GRAY	svetlo šedá
8	DARK GRAY	tmavo šedá
9	LIGHT BLUE	svetlo modrá
10	LIGHT GREEN	svetlo zelená
11	LIGHT CYAN	svetlo modro zelená (tirkisová)
12	LIGHT RED	svetlo červená
13	LIGHT MAGENTA	svetlo fialová
14	YELLOW	žltá
15	WHITE	biela
+128	BLINK	blikanie

Ak chceme, aby sa nám vypisoval text farebne, tak si nastavíme pomocou príkazu `textcolor` farbu, a potom vypíšeme pomocou príkazu `cprintf` text.

Farbu si môžeme vybrať z tabuľky. Do zátvoriek za príkaz `textcolor` môžeme napísať buď číslo od 0 po 15 alebo anglický názov farby.

```
textcolor(cislo_farby);
```

Nastaví farbu textu v textovom režime

Príklad 19

Napríklad prepíšeme náš prvý program v príklade 1 tak, aby vypísaný text bol modrý.

```
#include <stdio.h>
#include <conio.h>

void main()
{

  textcolor(BLUE);              // namiesto BLUE moze byt aj cislo 1

  cprintf("Ahoj svet, ja som programator");

  getch();
}
```

Príklad 20

Ak chceme, aby farby blikali tak k farbe pripočítame číslo 128, bude to vyzerať takto:

```
#include <stdio.h>
#include <conio.h>

void main()
{

  textcolor(BLUE+128);          // namiesto BLUE moze byt aj cislo 1

  cprintf("Ahoj svet, ja som programator");

  getch();
}
```

Ak by sme chceli zmeniť pozadie pod napísaným textom, tak to urobíme pomocou príkazu `textbackground`.

```
textbackground(cislo_farby);
```

Nastaví farbu pozadia na farbu, ktorú si zadáme. Môžeme použiť farby od 0 po 7.

Príklad 21

Prepíšeme predchádzajúci program tak, aby text bol modrý na hnedom podklade. Použijeme na to spomínaný príkaz `textbackground`.

```
#include <stdio.h>
#include <conio.h>

void main()
{

  textbackground(BLUE);
  textcolor(BROWN);

  cprintf("Ahoj svet, ja som programator");

  getch();
```

```
}
```

Príklad 22

Ak chceme, aby bola cela obrazovka farebná, tak za príkazom `textbackground` vymažeme celú obrazovku, použijeme `clrscr`. Tu je program:

```
#include <stdio.h>
#include <conio.h>

void main()
{

  textbackground(BLUE);
  clrscr();

  textcolor(BROWN);
  cprintf("Ahoj svet, ja som programator");

  getch();
}
```

2.11.4 Práca s kurzorom

```
gotoxy (int x, int y);
```

napr. `gotoxy(40,12);`

Tento príkaz umiestni kurzor na pozíciu 40 a riadok 12. Pozri si nasledujúci program.

Príklad 23

Nasledujúci program vypíše texty po celej obrazovke.

```
#include <stdio.h>
#include <conio.h>

void main()
{

  clrscr();

  gotoxy(10,4);
  printf("Ahoj");

  gotoxy(5,8);
  printf("Cau");

  gotoxy(40,2);
  printf("Dovidenia");

  getch();
}
```

Keby sme nepoužili príkazy `gotoxy` tak by sa nám texty vypísali na jednom riadku za sebou ("`AhojCauDovidenia`") , ale keď sme použili príkazy gotoxy tak sa nám texty vypísali presne na ten riadok, ktorý sme im zadali. Je to lepšie ako keby sme mali používať od riadkovanie pomocou escape sekvencie `\n`, ktorá je výhodná len na skočenie na začiatok ďalšieho riadku a nie na "skákanie" po celej obrazovke.

```
wherex();
```

Príkaz `wherex` vracia aktuálne pozíciu na x-ovej osi

```
wherey();
```

Príkaz `wherey` vracia aktuálne pozíciu na y-ovej osi

Príklad 24

Na programe si môžeme overiť

```
#include <stdio.h>
#include <conio.h>

void main()
{
  clrscr();
  gotoxy(10,10);
  printf("Pozicia kurzora: X:%d Y:%d\r\n", wherex(), wherey());
  getch();
}
```

Poznámka:

Ešte chcem dodať, že príkaz `gotoxy` mení pozíciu kurzora a príkazy `wherex` a `wherey` nemenia, len zisťujú aktuálnu polohu kurzora.

2.11.5 Výstup na obrazovku

```
cprintf(...);
```

Vypíše text v požadovanej farbe, ktorú sme zadali pomocou príkazu textcolor. Na rozdiel od príkazu `printf` nedoplňuje znak odriadkovania a znak návrat na začiatok riadku, preto ho musíme zadávať explicitne.

Príkaz `printf` je zadefinovaný v hlavičkovom súbore **stdio.h** a príkaz `cprintf` je zadefinovaný v hlavičkovom súbore **conio.h**.

```
putch(...);
```

Vypíše na obrazovku znak.

Toto bolo zoznámenie sa s príkazmi, ktoré sa používajú v textovom režime a teraz nasledujú dva programy na malé zopakovanie si príkazov:

Príklad 25

V textovom režime vypíšte do prostriedku červeným písmom na zelenom podklade text AHOJ.

```
#include<stdio.h>
#include<conio.h>

void main()
{
  clrscr();                     // vymaze obrazovku
  textmode(C80);                // nastavi TEXT MODE

  textcolor(RED);               // nastavi cervenu farbu textu
  textbackground(GREEN);        // nastvi zeleny podklad textu

  gotoxy(40,12);                // skoci na 40. riadok a 12. poziciu
  cprintf("AHOJ");              // vypise text AHOJ

  getch();                      // pocka na stlacenie klavesy
}
```

Príklad 26

Program vypisuje na modrú obrazovku s hnedou farbou písma text: text001 až text499 s oneskorením 20ms.

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>

void main()
{
  int i;

  textcolor(BROWN);             // nastavi hnedu farbu textu
  textbackground(BLUE);         // nastavi modru farbu podkladu

  clrscr();

  for(i=1; i<500; i++){
    cprintf(" text%03d", i);
    delay(20);
  }

  getch();                      // pocka na stlacenie klavesy
}
```

Oneskorenie zabezpečuje nový príkaz `delay`, ktorý je zadefinovaný pre nás v novej knižnici **dos.h**. Zadané číslo spôsobuje oneskorenie. V našom prípade to je 20ms. Podobný príkaz je príkaz `sleep`, ktorému zadávame oneskorenie v sekundách, napr. `sleep(1)` bude oneskorenie približne 1s. Píšem približne pretože na rôznych počítačoch je tento príkaz rôzne rýchlo vykonaný (napríklad na novších počítačoch).

2.12 Práca so súborom

Každý program spracováva nejaké vstupné údaje a nám ukazuje výsledky, ktoré touto činnosťou získal. Pokiaľ by to nebola pravda, tak by sme zrejme nemali dôvod takýto program písať. Programy, ktoré sme doteraz napísali čítajú zo štandardného vstupu (klávesnica) a píšu na štandardný výstup (monitor, tlačiareň).

V tejto kapitole si napíšme program, ktorý pracuje so súbormi. Program bude čítať, prepisovať a vytvárať údaje.

Vstup a výstup budeme často skracovať na zápis I/O.

Každý program v jazyku C má štandardne otvorený štandardný vstup **stdin**, štandardný výstup **stdout** a štandardný chybový výstup **stderr**. Tie sú obvykle napojené na klávesnicu a terminál.

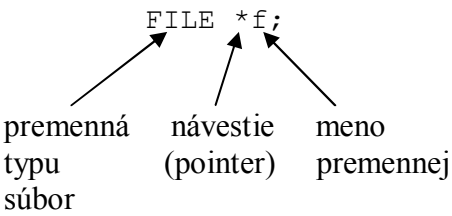
2.12.1 Dátový tok FILE

Je to typ premennej, s ktorým pracuje väčšina funkcii v knižnici **stdio.h**. V jazyku C sa dajú definovať aj vlastné typy premenných. V súbore **stdio.h** je definovaný typ `FILE *` (nové typy sa obyčajne píšu veľkými písmenami). Premennej typu `FILE *` hovoríme dátový tok.

Priamo s dátovým tokom sa pracuje len pri I/O operáciách so súbormi. Dátový tok je pointer a tak mu treba priradiť nejakú adresu.

Pracovať so súborom pomocou dátového toku môžeme v dvoch režimoch – v textovom a binárnom. Rozdiel medzi textovým s binárnym súborom je v spôsobe ako je chápaný znak konca riadku. V textovom režime sú konce riadku rozpoznané, zatiaľ čo v binárnom nie.

V práci so súborom musíme deklarovať premennú typu `FILE`:



2.12.2 Príkaz fopen - otvorenie dátového toku

Aby sme mohli pracovať so súborom musíme si ho pre prácu najskôr otvoriť. Robí to funkcia `fopen`:

```
fopen(const char *filename, const char *mode);
```

Funkcia `fopen` (“file open”) otvorí súbor zadaný v prvom parametri (ak nezadáš cestu, hľadá ho v aktuálnom adresári; pri zadávaní cesty nezabudni zdvojiť znak \ na \\ , pretože na rozdiel od príkazu `#include "súbor"` je to reťazcová konštanta). Druhým parametrom je spôsob práce so súborom. Parametre zobrazuje táto tabuľka:

parameter	popis
r	otvorenie súboru na čítanie
w	otvorenie súboru na zápis, alebo prepísanie
a	otvorenie súboru na pripájanie na koniec
r+	otvorenie súboru na čítanie a zápis
w+	otvorenie súboru na čítanie, zápis alebo prepísanie
a+	otvorenie súboru na čítanie a zápis na koniec, pokiaľ súbor neexistuje tak ho vytvorí
t	textový režim, môže byť kombinovaný so všetkými predchádzajúcimi parametrami
b	binárny režim, môže byť kombinovaný so všetkými predchádzajúcimi parametrami (okrem "t")

Ak otvoríme súbor pre zápis alebo pre pripájanie na koniec a súbor, ktorý chceme otvoriť neexistuje, tak je súbor vytvorený. Otvorenie existujúceho súboru pre zápis spôsobí jeho vymazanie a vytvorenie prázdneho súboru.

Skutočné volanie vyzerá asi takto:

```
FILE *f;

f = fopen("skuska.txt","r")
.
.
.
```

Príkaz `fopen` alokuje miesto v pamäti a uloží tam obsah súboru. Táto premenná je typu `FILE` a funkcia `fopen` vráti pointer na prvý znak v tejto premennej. Samozrejme nie vždy sa naraz do pamäte zmestí celý súbor. Funkcia `fopen` predá kontrolu nad dátovým tokom operačnému systému. Operačný systém postupne doťahuje do dátového toku informácie, podľa toho, ktorá časť súboru sa práve číta. Programátorovi sa to však javí ako keby bol celý súbor v pamäti a on má k dispozícii adresu prvého bytu tejto oblasti…

Príkaz `fopen` vráti pri chybe hodnotu NULL – symbolická konštanta pre nulový pointer.

2.12.3 Príkaz fclose - zatvorenie dátového toku

Po skončení programu sa dátový tok neuzavrie sám od seba (údajne to spraví operačný systém, ale nespoliehal by som sa na to). Slúži na to funkcia *fclose*:

```
fclose(f);
```

Funkcia v prípade úspechu vráti hodnotu 0, inak EOF. Uvoľní pamäť vyhradenú pre `FILE` a vyprázdni prípadnú vyrovnávaciu pamäť.

2.12.4 Príkaz getc

Ďalej je nutné vedieť pri práci so súborom ako čítať z práve otvoreného súboru. Je veľa možností.

Príkaz getc je najjednoduchšia možnosť. Ako argument je zadaný pointer na FILE, napr.:

```
c = getc(f);
```

V prípade úspešného načítania znaku z toku ho prevedie bez znamienka na typ int. Takto získaná hodnota je hodnota návratová, ktorú môžeme napríklad vypísať na obrazovku, napr. pomocou príkazu:

```
putchar(c);
```

V prípade chyby alebo dosiahnutia konca prúdu (súboru) je pre príkaz `getc` vrátená hodnota EOF.

Príklad 27

Program vypíše obsah súboru na obrazovku a zastaví sa až na konci súboru (znak EOF).

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int c;
  FILE *f;

  clrscr();

  f = fopen("data.txt","r");     // nacita subor na citanie do pamate
  if(f==NULL)   {          // testuje ci sa podarilo nacitat subor
    printf("Nenasiel som subor.");
    getch();
    exit();                      // ukonci program
  }

  while((c=getc(f)) != EOF){     // testuje ci nie je koniec suboru
    putchar(c);            // vypise znak na obrazovku
  }

  getch();
  fclose(f);                     // uzavrie datovy tok
}
```

Príklad 28

Program vypíše obsah prvého riadku na obrazovku a zastaví sa na konci riadku (znak \n).

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
  int c;
  FILE *f;

  clrscr();

  f = fopen("data.txt","r");     // nacita subor na citanie do pamate
  if(f == NULL) {     // testuje ci sa podarilo nacitat subor
    printf("Nenasiel som subor.");
    getch();
    exit(0);                     // ukonci program
  }

  while((c=getc(f)) != '\n'){    // testuje ci nie je koniec suboru
    putchar(c);                  // vypise znak na obrazovku
  }

  getch();
  fclose(f);                     // uzavrie datovy tok
}
```

2.12.5 Príkaz putc

```
int putc(int c, FILE *stream);
```

napr.:

```
putc(c, f);
```

Zapíše znak c do dátového toku f. Vráti rovnakú hodnotu ako zapísal. V prípade chyby, alebo dosiahnutia konca toku vracia `EOF`.

Príklad 29

Program zapíše do súboru `data.txt` ASCII tabuľku, každý znak na nový riadok.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void main()
{
  int i;
  FILE *f;

  clrscr();

  f = fopen("data.txt","w");
  if(f == NULL){
    printf("Nenasiel som subor.");
    getch();
    exit(0);          // ukonci program
  }

  for (i=0; i<256; i++){
    putc(i, f);       // zapise znaky do toku f
    putc('\n', f);          // zapise riadok do toku f
  }

  getch();            // pocka na stlacenie klavesy
  fclose(f);          // uzavrie datovy tok f
}
```

Príklad 30

Upravíme predchádzajúci program tak, aby nám zapísal do súboru `data.txt` len znaky malej a veľkej abecedy.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void main()
{
  int i;
  FILE *f;
```

```
  clrscr();

  f = fopen("data.txt","w");
  if(f == NULL){
    printf("Nenasiel som subor.");
    getch();
    exit(0);    // ukonci program
  }

  for (i=0; i<256; i++){
    if( (i>='a')&&(i<='z') || (i>='A')&&(i<='Z') ){
      putc(i, f);
      putc('\n', f);
    }
  }

  getch();                  // pocka na stlacenie klavesy
  fclose(f);                // uzavrie datovy tok f
}
```

2.12.6 Testovanie konca súboru

Používa sa konštanta EOF alebo makro `feof` na testovanie konca súboru pomocou EOF.

`if( (c = getc(f)) != EOF )` - premenná c nesmie byť typu char

V prvom príklade si ukážeme ako otestujeme či nie je koniec súboru pomocou hore uvedenej podmienky a v druhom príklade pomocou makra `feof` na testovanie konca súboru pomocou EOF

Príklad 31

Ukážeme si program, ktorý vytvorí z jedného súboru presnú kópiu. Program funguje pokiaľ príkaz `getc` nevráti konštantu EOF.

```
#include <stdio.h>

void main()
{
  FILE *fr, *fw;
  int c;

  fr = fopen ("orig.txt", "r");
  fw = fopen ("kopia.txt", "w");

  while ( ((c = getc(fr)) != EOF)) {
    putc(c, fw);
  }

  fclose(fr);
  fclose(fw);
}
```

Program funguje tak, že otvorí dva súbory jeden pre čítanie (orig.txt) a jeden pre zápis (kopia.txt). Potom pomocou cyklu `while` testuje či nie je koniec súboru, ak nie je tak vypíše znak, ktorý vráti príkaz `getc(fr)`, pomocou príkaz `putc(fw)`.

Testovanie konca súboru pomocou makra `feof` je vhodnejšie ak čítame znak z binárneho súboru. Toto makro `feof` nám vracia hodnotu TRUE (nenulová) pokiaľ posledné čítanie bolo už za koncom súboru. Hodnotu FALSE (0) vracia vtedy, ak sme pri čítaní súboru na koniec súboru ešte neprišli.

Príklad 32

Ten istý program ako v predchádzajúcom príklade, ale s testovaním konca súboru pomocou makra `feof`.

```
#include <stdio.h>

void main()
{
  FILE *fr, *fw;
  int c;

  fr = fopen ("orig.txt", "r");
  fw = fopen ("kopia.txt", "w");

  while ( feof(fr) == 0){
    c = getc(fr);
    putc(c, fw);
  }

  fclose(fr);
  fclose(fw);
}
```

Tento blok príkazov v tomto príklade:

```
c = getc(fr);
putc(c, fw);
```

by sme mohli nahradiť jedným príkazom:

```
putc (getc(fr), fw);
```

2.12.7 Test správnosti otvorenia a zatvorenia súboru

Funkcia `fopen` aj funkcia `fclose` sú funkcie, ktoré vracajú hodnotu. Táto hodnota sa požíva pre test či príkaz otvorenia alebo uzavretia súboru prebehol správne.

Pri nesprávne vykonanom otvorení súboru vracia príkaz `fopen` konštantu NULL.

Pri nesprávne vykonanom zatvorení súboru vracia príkaz `fclose` konštantu EOF.

Príklad 33

Program, ktorý skopíruje obsah jedného súboru do druhého súboru teraz doplníme o testovanie správnosti otvorenia a zatvorenia súboru a bude vyzerať asi takto:

```
#include <stdio.h>
```

```
#include <conio.h>

void main()
{
  FILE *fr, *fw;
  int c;

  if ((fr = fopen ("orig.txt", "r")) == NULL){
    printf("Subor ORIG.TXT sa nepodarilo otvorit.\n");
    getch();
    return;          // ukoncenie programu
  }

  if ((fw = fopen ("kopia.txt", "w")) == NULL){
    printf("Subor KOPIA.TXT sa nepodarilo otvorit.\n");
    getch();
    return;          // ukoncenie programu
  }

  while ( ((c = getc(fr)) != EOF)) {
    putc(c, fw);
  }

  if (fclose(fr) == EOF){
    printf("Subor ORIG.txt sa nepodarilo uzatvorit.\n");
    getch();
    return;
  }

  if (fclose(fw) == EOF){
    printf("Subor KOPIA.txt sa nepodarilo uzatvorit.\n");
    getch();
    return;
  }
}
```

2.12.8 Formátovaný výstup do súboru - fprintf

Táto funkcia realizuje zápis do súboru.

```
int fprintf (FILE *stream, const char *format [, argument, ...]);
```

napr.:

```
fprintf (f, "Ja som %d. programator", i);
```

Príkaz `fprintf` pracuje skoro rovnako ako príkaz `printf`. Jediný rozdiel je v tom, že príkaz fprintf vysiela svoj výstup na dátový tok, ktorý je zadefinovaný v prvom argumente tak, ako je to vidieť na vzore. Je tiež zadefinovaný v knižnici **stdio.h**.

Príklad 34

Program zapíše do súboru pokus.dat desať čísiel od 0 po 9 pomocou príkazu `fprintf`, ktorý sa nachádza v cykle `for`.

```
#include <stdio.h>
#include <conio.h>

void main()
{
  FILE *f;
  int i;

  if ((f = fopen ("pokus.dat", "w")) == NULL){
    printf("Subor JANKO.TXT sa nepodarilo otvorit.\n");
    getch();
    return;           // ukoncenie programu
  }

  for(i=0; i<10; i++){
    fprintf(f, "%d\n", i);
  }

  if (fclose(f) == EOF){
    printf("Subor JANKO.TXT sa nepodarilo uzatvorit.\n");
    getch();
    return;           // ukoncenie programu
  }
}
```

Príklad 35

Program zapíše do súboru jano.txt v desiatich riadkoch tri náhodné čísla oddelené medzerou.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
  FILE *f;
  int i, a, b, c;

  if ((f = fopen ("jano.txt", "w")) == NULL){
    printf("Subor JANO.TXT sa nepodarilo otvorit.\n");
    getch();
    return;           // ukoncenie programu
  }

  randomize();        // zabezpeci, aby sa cisla neopakovali

  for(i=0; i<10; i++){
    a = random(100);
    b = random(50);
    c = random(60);

    fprintf(f, "%d. %d %d %d\n", i, a, b, c);
  }

  if (fclose(f) == EOF){
    printf("Subor JANO.TXT sa nepodarilo uzatvorit.\n");
```

```
    getch();
    return;           // ukoncenie programu
  }
}
```

Princíp (algoritmus) ako program pracuje by ti mal byť už jasný. Jedine čo tu je nové sú príkazy `randomize`, `random` a práve spomínaný príkaz `fprintf`.

Príkaz `randomize` zabezpečí, aby sa vygenerovalo náhodné číslo. Pomocou príkazu `random` zvolíme z akej oblasti sa má číslo vygenerovať, napr. keď sme dali `random(100)`, tak sa nám náhodne vygeneruje číslo od 0 po 99, a to sa nám v programe uloží do premennej **a**. Podobne je to aj s premennou **b** a **c**.

Ak by sme chceli aby vygenerované číslo nebolo nikdy číslo 0, tak pripočítame číslo 1 k náhodne vygenerovanému číslu. Zápis bude vyzerať asi takto:

```
a = 1 + random(100);
```

Každé vygenerované číslo pomocou príkazu `random(100)` sa zväčší o 1. Ak by sa nám aj vygenerovalo číslo 0, tak po pričítaní jednotky sa nám číslo zmení na 1. Teraz je rozsah, z ktorého sa nám vygeneruje náhodné číslo od 1 po 100.

Ak by sme chceli zjednodušiť tieto 4 riadky v programe:

```
a = random(100);
b = random(50);
c = random(60);

fprintf(f,"%d. %d %d %d\n", i, a, b, c);
```

tak to urobíme takto:

```
fprintf(f,"%d. %d %d %d\n",i, random(100),random(50),random(60));
```

2.12.9 Formátovaný vstup zo súboru - fscanf

Táto funkcia realizuje čítanie zo súboru.

```
int fscanf (FILE *stream, const char *format [, address, ...]);
```

napr.:

```
fscanf (f, "%d %d", &a, &b);
```

Funkcia `fscanf` pracuje rovnako ako jej ekvivalent pre vstup znakov z klávesnice, lenže v tomto prípade číta znaky z dátového toku, ktorý je zadefinovaný v prvom argumente.

Funkcia `fscanf` vracia počet úspešne prečítaných položiek. Môžeme teda veľmi ľahko otestovať, či zadaný súbor obsahuje požadované údaje, napr. pomocou podmienky `if`. Ukážeme si to na príkladoch.

Príklad 36

Program prečíta 4 náhodne čísla, ktoré vygeneroval program v príklade 35, a potom ich vypíše na obrazovku a za nimi vypíše ich súčet.

Formát čísiel v súbore jano.txt je: `x. x x x`

```
#include <stdio.h>
#include <conio.h>

void main()
{
  FILE *f;
  int i;
  int a, b, c;

  clrscr();

  if ((f = fopen ("jano.txt", "r")) == NULL){
    printf("Subor JANO.TXT sa nepodarilo otvorit.\n");
    getch();
    return;           // ukoncenie programu
  }

  for(i=0; i<10; i++){
    if (fscanf(f, "%d. %d %d %d", &i, &a, &b, &c) == 4){
      printf("%d. %2d %2d %2d; Ich sucet je %3d\n",i,a,b,c,a+b+c);
    }

    else {
      printf("V subore sa nenachadzaju cisla vo formate  x. x x x");
      getch();
      return;         // ukoncenie programu
    }
  }

  if (fclose(f) == EOF){
    printf("Subor JANO.TXT sa nepodarilo uzatvorit.\n");
    getch();
    return;           // ukoncenie programu
  }

  getch();
}
```

Program prečíta 4 čísla pomocou príkazu `fscanf`, ktorý vráti počet úspešne načítaných čísiel. Ak je táto hodnota rovná 4, tak sa vypíšu všetky 4 čísla a text: "`Ich sucet je ...`". Ak nie je táto hodnota rovná 4, tak sa vykoná príkaz printf, ktorý vypíše text: "`V subore sa nenachadzaju cisla vo formate x. x x x`".

2.12.10 Štandardný vstup a výstup – stdin a stdout

V súbore **stdio.h** sú tiež už vytvorené tieto dva dátové toky (typu `FILE *`). Sú v priamej väzbe s operačným systémom a predstavujú vstup z klávesnice a výstup na obrazovku. S týmito tokmi pracujú základné funkcie pre vstup a výstup na obrazovku.

```
fprintf(stdout,"Hello world");
```

je to to iste ako:

```
printf("Hello world");
```

Podobne je to aj so vstupom:

```
fscanf(stdin, "%d", &c);
```

je to to iste ako:

```
scanf("%d", &c);
```

Príklad 37

Program vypíše na obrazovku text pomocou príkazu `fprintf`:

```
#include <stdio.h>
#include <conio.h>

void main()
{
  clrscr();

  fprintf(stdout, "Ahoj svet, ja som programator");
  getch();
}
```

Tento program môžete porovnať s programom v príklade č.1.

2.13 Funkcie a procedúry

Prechádzame na najdôležitejšiu časť jazyka C – *funkcie*. Funkcie hrajú v programe v jazyku C rovnakú úlohu ako bunky v tele, alebo atómy v hmote. Sú to základné kamene programu. Funkcia je vlastne podprogram zložený z deklarácii premenných, kľúčových slov a volaní ďalších funkcií.

Funkcia je nezávislá časť programového kódu, ktorá vykonáva určitú úlohu, a ktorá má pridelené meno. Použitím mena funkcie v programe sa vykonáva kód tela funkcie. Program môže tiež posielať funkcii informácie nazývané argumenty alebo parametre, a funkcia môže vrátiť informáciu, nazývanú návratová hodnota, časti programu, odkiaľ bola funkcia volaná. Každá funkcia má štandardne iba jeden výstup (návratová hodnota), ale môže vrátiť i viacero hodnôt – pomocou smerníkov resp. globálnych premenných.

Funkcie v jazyku c nemôžu byť vnorené, t.j. jedná funkcia nemôže obsahovať v svojom tele definíciu druhej funkcie. Z toho vyplýva, že formálne parametre a lokálne premenné sú prístupné iba vo funkcií, v ktorej boli definované a sú skryté zvonka tejto funkcie.

Program v jazyku C obsahuje jednu alebo viacej definícií funkcií, z ktorých jedna sa musí vždy nazývať **main()**. Spracovanie programu začína volaním tejto funkcie a končí opustením tejto funkcie.

2.13.1 Definícia funkcie

Každá funkcia, ktorú chceme použiť musí mať uvedenú tzv. definíciu a má niekoľko základných vlastností:

- má určené meno, pomocou ktorého bude vyvolaná
- môže mať parametre
- má návratovú hodnotu
- a telo zložené z príkazov, ktoré funkcia po svojom zavolaní vykoná

Definícia funkcie má nasledujúci syntax:

```
hlavička funkcie
{
  telo funkcie
}
```

Definícia funkcie určuje ako, tak aj jej telo, zatiaľ čo *deklarácia* funkcie špecifikuje iba hlavičku funkcie, t.j. meno funkcie, typ návratovej hodnoty a prípadne i typ a počet jej parametrov.

2.13.2 Hlavička funkcie

Je to prvý riadok definície funkcie. Obsahuje typ návratovej hodnoty funkcie, identifikátor funkcie a deklarácie parametrov funkcie. Napríklad:

```
int funkcia(int par1, char par2)
```

Je to hlavička funkcie s identifikátorom "funkcia", dvoma parametrami "par1" a "par2", pričom prvý je typu `int` a druhý typu `char`. Funkcia má návratovú hodnotu typu int.

2.13.3 Návratová hodnota funkcie

Umožní funkcii zmeniť hodnotu premennej vo funkcii, v ktorej je volaná. Tiež môže fungovať ako podmienkový výraz. Je to vlastne výstup funkcie. Napríklad:

```
zvysok = funkcia(a, b);
if(funkcia(a,b))
  printf("Cislo %d je delitelne cislom %d",a,b);
```

Uvedom si, že hodnota podmienkového výrazu je normálna hodnota typu *int*, ktorá je buď nulová alebo nenulová a tak sa môže do zátvoriek *if* vložiť priamo volanie funkcie, ktorá vracia hodnotu. V jazyku C sa to často využíva.

2.13.4 Parametre funkcie

Sú vstupnými údajmi pre funkciu, no môžeš ich používať aj na výstup pre ľubovolný počet hodnôt (nezabudni, že ak hodnotu vraciaš, môže byť len jedna !). Všimni si, že presne takto funguje funkcia *scanf* (a to je dôvod, prečo tam treba používať ten & pred premennými). Môže byť aj funkcia s premenlivým počtom parametrov (napríklad *printf* alebo *scanf*). Definícia týchto funkcii je zložitejšia a tak ťa to zatiaľ nemusí trápiť.

2.13.5 Telo funkcie

Za hlavičkou nasleduje blok príkazov ohraničený zloženými zátvorkami. Pozor, teraz tu musia byť vždy, aj keď je to len tzv. jednoriadková funkcia! V tele funkcie sa môžu nachádzať aj ďalšie deklarácie premenných a sú v ňom použité parametre funkcie. Na konci algoritmu tela funkcie je obyčajne kľúčové slovo *return*, ktoré vráti hodnotu. Napríklad:

```
{
 return(par1 % par2);
}
```

2.13.6 Deklarácia funkcie

Deklarácie funkcii majú presne rovnaký tvar ako hlavička funkcie, ale parametre môžeš vypustiť (typy by tam mali zostať). Dá sa povedať, že je to vlastne kópia hlavičky funkcie.

```
int funkcia(int, char);
```

Tomuto spôsobu deklarácie funkcie (so zoznamom typov parametrov) sa hovorí prototyp funkcie. Najvýhodnejšie je dať pred funkciu *main* prototypy všetkých funkcii v programe (samozrejme okrem funkcie *main*) a ich definície za funkciu *main*.

Príklad 38

```
#include <stdio.h>
#include <conio.h>

int scitaj(int i, int j)
{
  return i+j;
}

void main()
{
  clrscr();

  printf("%d", scitaj(1,10) );

  getch();
}
```

Na programe je vidieť použitie jednoduchej funkcie `scitaj`, ktorá sčíta dve zadané čísla a vráti ich súčet, ktorý potom pomocou príkazu `printf` vypíšeme na obrazovku.

Príklad 39

```
#include <stdio.h>
#include <conio.h>

int max(int i, int j)
{
  if(i>j)
    return i;

  return j;
}

void main()
{
  clrscr();

  printf("%d", max(1,10) );
```

```
  getch();
}
```

Funkcia vráti väčšie číslo, ktoré sa vypíše na obrazovku.

Príklad 40

Program vypočíta obvod a obsah obdĺžnika a vypíše ho farebne na obrazovku.

```
#include <stdio.h>
#include <conio.h>

int obvod(int i, int j)
{
  return 2*(i+j);          // vrati obvod obdlznika
}

int obsah(int i, int j)
{
  return i*j;              // vrati obsah obdlznika
}

void main()
{
  int a, b;

  clrscr();                // vymaze obrazovku

  gotoxy(2,4);
  printf("Program vypocita obvod a obsah obdlznika");

  gotoxy(2,8);
  printf("Zadaj stranu a: ");
  scanf("%d", &a);         // nacita stranu a do premennej a
  gotoxy(2,10);
  printf("Zadaj stranu b: ");
  scanf("%d", &b);         // nacita stranu b do premennej b

  gotoxy(2,14);
  textcolor(GREEN);
  cprintf("Obvod obdlznika je ");
  textcolor(GREEN+BLINK);
  cprintf("%d", obvod(a, b) );        // vypise obvod obdlznika

  gotoxy(2,16);
  textcolor(RED);
  cprintf("Obsah obdlznika je ");
  textcolor(RED+BLINK);
  cprintf("%d", obsah(a, b) );        // vypise obsah obdlznika

  getch();}
}
```

2.13.7 Procedúry a dátový typ void

Formálne síce v C procedúry neexistujú, ale sú dva spôsoby ako tento „nedostatok obísť:

1. Funkcia návratovú hodnotu síce vráti, ale nikto ju nechce. Typický príklad je čakanie na stlačenú klávesu funkcie `getchar()` z knižnice **stdio.h**, ktorá pri normálnom použití vráti stlačený znak, teda návratová hodnota funkcie je stlačený znak. Príklad volania funkcie ako procedúry:

```
getchar(); // cakanie na stlacenie klavesy
```

2. Funkcia sa definuje ako funkcia vracajúca typ void (to znamená, že nevracia žiadnu hodnotu), napr.:

```
void vypis_text()
{
  printf(„Ja som programator“);
}
```

Príkaz `return` v tomto prípade nie je nutný. Pokiaľ nie je uvedený, nahrádza ho uzatváracia zložená zátvorka funkcie. Príkaz `return` sa potom používa iba pre nútené ukončenie funkcie pred dosiahnutím jej konca, napríklad po nejakej podmienke a má tvar `return`.

Príklad 41

Teraz si napíšeme program, ktorý pomocou zavolanej funkcie vypíše text.

```
#include <stdio.h>
#include <conio.h>

void vypis_text()
{
  printf("Ja som programator");
}

void main()
{
  clrscr();

  vypis_text();

  getch();
}
```

Príklad 42

Pomocou nasledujúceho programu vypíšeme na obrazovku text a číslo, ktoré vložíme do funkcie. Ak si to nepochopil, tak sa pozri na program:

```
#include <stdio.h>
#include <conio.h>

void vypis(int i)
{
  printf("Zadal si cislo %d\n", i);
}
```

```
void main()
{
  clrscr();

  vypis(5);
  vypis(1000);

  getch();
}
```

Program vypísal 2 riadky:

```
Zadal si cislo 5
Zadal si cislo 1000
```

2.13.8 Funkcie nevracajúce int

Funkcie nemusia vraciať len typ **int**. V tomto prípade sa opäť nič nezmení len návratová hodnota (typ) funkcie.

Napr. v príklade 39, ktorý vracia väčšie z dvoch zadaných čísiel sme mohli zadávať len celé čísla, pretože sme použili premenne typu **int**. Teraz si ukážeme jednu možnosť ako by to mohlo vyzerať, aby nám program porovnával aj čísla s desatinnou čiarkou.

Príklad 43

```
#include <stdio.h>
#include <conio.h>

double max(double i, double j)
{
  if(i>j)
    return i;

  return j;
}

void main()
{
  clrscr();

  printf("%.03f", max(1.55,1.9) );

  getch();
}
```

Program vrati väčšie číslo a aj s desatinnými miestami.

Príklad 44

Program vynásobí zadané číslo s číslom $\pi = 3,14$

```
#include <stdio.h>
#include <conio.h>

double pikrat(double i)
{
```

```
  return (i * 3.14);
}

void main()
{
  clrscr();

  printf("%.5f", pikrat(1.541169) );

  getch();
}
```

2.13.9 Premena návratovej hodnoty funkcie

Pokiaľ nie je typ návratového výrazu alebo hodnoty zhodný s návratovým typom, tak sa robí typová premena.

Príklad 45

Program premenní číslo, ktoré je uložené v premennej typu **double** na číslo typu **int**. Jednoducho povedané program "odstrihne" čísla, ktoré sa nachádzajú za desatinnou čiarkou.

```
#include <stdio.h>
#include <conio.h>

int premena( double i )
{
  return(i);
}

void main()
{
  clrscr();

  printf("%d", premena(5.566));

  getch();
}
```

Volanie: `premena(5.566)` spôsobi, že hodnota `5.566` je pretypovaná na int ("odstrihnutá") a funkcia vráti číslo `5`.

2.14 Jednorozmerné pole

Polia sú veľmi často používaný typ premennej v každom programovacom jazyku. V jazyku C je ešte používanejší… Pole je typ, ktorý ti umožní združiť viac premenných rovnakého typu do jednej, pričom každému prvku je priradené číslo. Je to vlastne programátorský ekvivalent matematickej množiny. Najlepšie to ilustruje príklad:

Príklad 46

Program najskôr zapíše desať hodnôt do poľa, a potom pomocou príkazu `printf` hodnoty z pola vypíše na obrazovku.

```
#include<stdio.h>

void main()
{
  int pole[10], i;     // deklaracia pola 10-tich prvkov typu int

  for(i=0;i<10;i++)
    scanf("%d",&pole[i]); // pole sa zacina prvkom 0 a konci 9

  for(i=0;i<10;i++)
    printf("\n %d",pole[i]);
}
```

Ak si hneď pochopil, že program najprv načíta desať čísel a následne ich vytlačí každé na ďalší riadok, tak si môžeš gratulovať. Ak nie, tak nevadí, skúsime si to trochu vysvetliť.

Deklarácia poľa je jednoduchá a môžeš ju kombinovať aj s klasickými premennými. Napr.:

```
int pole[10], i, a;
```

Treba si uvedomiť, že deklarácia len priradí oblasť v pamäti, ale nezaručí ti, že v tej oblasti už predtým niečo nebolo (napríklad premenná iného programu, ktorý bol spustený predtým), zaručí ti len to, že údaje ktoré sa takto ocitnú v tejto pamäťovej oblasti už nepoužíva žiadny program (program, ktorý ich používal je už ukončený).

Ak chceme z pola čítať informácie (napr. čísla) už od začiatku môže sa nám stať, že načítame čísla, ktoré sme tam nezapísali, a preto je lepšie ak si na začiatku programu zapíšeme do poľa pomocou nejakého cyklu napr. číslo 0 alebo 1... Záleží na tom čo nám v danej situácii vyhovuje. V príklade 46 sme to nemuseli robiť, pretože ak tam aj boli nejaké čísla , tak sme ich pomocou prvého cyklu `for` premazali.

Polu možno priradiť údaje už pri deklarácii. V tomto prípade sa používa zápis podobný matematickému zápisu množiny.

```
int pole[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Hodnoty uložené v poli sa používajú zápisom *názov_poľa*[*výraz*] pričom tento zápis má hodnotu prvku poľa, ktorého poradové číslo je zhodné s (celočíselnou) hodnotou výrazu. Treba si uvedomiť, že pole sa začína prvkom 0 (nie 1) a posledný prvok má číslo n-1 (n je počet prvkov).

```
pole[0] == 1
pole[1] == 2
    .
    .
    .
pole[9] == 10
```

Teraz by ti to malo byť jasné.

Polia môžu byť dokonca aj dvojrozmerné (každý prvok má ďalšie prvky) alebo viacrozmerné.

2.15 Reťazec

Viem, že práca s číslami je základ, ale príťažlivejšia je práca s textom a grafikou. Prácu s textovými reťazcami si schopný zvládnuť už teraz. Možno sa teraz čuduješ, že som sa k tomu dostal až teraz, ale inak to nešlo. Teraz zrejme čakáš, že ti len popíšem ďalší typ premennej a ide sa ďalej. Celá vec však má jeden háčik. V jazyku C totiž takýto typ premennej neexistuje! Reťazce sa totiž načítavajú do poľa znakov. Je to špeciálny typ jednorozmerného pola.

Príklad 47

```
#include<stdio.h>

void main()
{
  char retazec[10];  // deklaracia retazca na ulozenie 10 znakov

  scanf("%s",retazec);      // operator & sa tentoraz nepise !
  printf("\n%s",retazec);
}
```

Tentoraz sa v príkaze `scanf` píše len názov pola a nepíše sa operátor *&*. Možno ťa teraz trápi, ako `printf` rozpozná, že sa reťazec končí, ak si vložil len napr. 5 znakov. Funkcia `scanf` totiž pridá na koniec reťazca znak '\0'. Ak funkcia `printf` narazí na tento znak, už nevypisuje ďalej. Preto je toto pole v našom príklade určené pre reťazec dlhý 9 znakov + '\0'. Ak text (aj so znakom '\0') presiahne kapacitu poľa, prepíše sa oblasť, ktorá nasleduje v pamäti za rezervovanou oblasťou poľa. Takto sa môžu prepísať iné premenné a môže to spôsobiť aj pád systému! Aby si tomuto zabránil treba obmedziť počet znakov zápisom:

```
scanf("%9s",retazec);
```

U funkcie `scanf` funguje špecifikácia dĺžky reťazca trochu inak. Dáva sa tu len jedno číslo, a to určuje maximálnu dĺžku reťazca. Znak '\n' na konci sa už do reťazca neukladá.

Príklad 48

Práca s reťazcami sa najlepšie dá naučiť na nejakom zaujímavom programe:

```
#include<stdio.h>
#include<string.h>

void main()
{
  char heslo[11];

  while(1){
    printf("\n\nVloz heslo (max 10 znakov): ");
    scanf("%10s",heslo);

    if(strcmp(heslo,"password")==0)
      break;
    else
      printf("\n\nPoslem na teba policiu...\a\a");
    }
```

```
 printf("\n\nGratulujem, vlozil si spravne heslo!");
}
```

V tomto programe ťa asi najviac upúta ďalší `#include`. Je potrebný, pretože v programe som použil ďalšiu funkciu (`strcmp`), ktorá sa nachádza v súbore **string.h** . Táto funkcia, ako už možno tušíš, slúži na porovnávanie reťazcov (`strcmp -> string compare` = porovnanie reťazcov). Musíš si uvedomiť, že heslo je znakové pole a nemôžeš jednoducho napísať:

```
if(heslo=="password")
```

Heslo musíš s reťazcovou konštantou porovnávať znak po znaku teda:

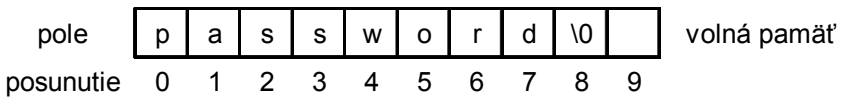
```
if(heslo[0]=='p')
  if(heslo[1]=='a')
    if(heslo[2]=='s')
      if(heslo[3]=='s')
        if(heslo[4]=='w')
          if(heslo[5]=='o')
            if(heslo[6]=='r')
              if(heslo[7]=='d')
                if(heslo[8]=='\0')
```

Ale načo si komplikovať život. Dobrodinci s firmy Borland napísali na tento účel funkciu `strcmp`. Táto funkcia má dva parametre kam napíšeš dva reťazce, ktoré porovnávaš. Ak sú reťazce zhodné, funkcia vráti hodnotu 0. Inak vráti počet znakov, v ktorých sa reťazce líšia. Ak je prvý reťazec kratší ako druhý, bude táto hodnota záporná, v opačnom prípade bude kladná.

Podobne ako pri číselnom poli, aj znakovému poľu možno priradiť hodnoty hneď v deklarácii:

```
char retazec[] = { 'p','a','s','s','w','o','r','d','\0' }
```

Podobne to môžeme aj nakresliť, napríklad tak to:



Všimol si si, že som neudal počet prvkov. Ak priradzuješ do pola pri deklarácii a chceš, aby pole malo taký istý počet prvkov ako priradzovaná množina, tak stačí zadať prázdne hranaté zátvorky. Predchádzajúci zápis je dosť zdĺhavý a tak môžeš použiť aj reťazcovú konštantu.

```
char retazec[] = "password";
```

V súbore **string.h** sa nachádza množstvo ďalších užitočných funkcii.

2.16 Štandardné funkcie pre prácu s reťazcami

Tu je nutné zdôrazniť, že C nedefinuje prácu s reťazcami ako súčasť jazyka. Pretože sú, ale práce s reťazcami veľmi časté, poskytuje jazyk C vďaka svojmu množstvu štandardných knižniciach, pomocou ktorých sa pracuje s reťazcami ľahko a rýchlo.

Ak chceme tieto funkcie využívať je nutné pripojiť do programu ďalší štandardný hlavičkový súbor, a to súbor **string.h**.

V tomto štandardnom hlavičkovom súbore sa nachádza množstvo funkcií, z ktorých ďalej uvedieme len tie najdôležitejšie.

2.16.1 Dĺžka reťazca

```
int strlen(char *ret);
```

Vracia dĺžku reťazca ret bez ukončovacieho znaku '\0'

Napr.: `strlen("ahoj")` vráti hodnotu 4

Príklad 49

Program vypíše zadaný reťazec a počet jeho znakov.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
  int p;
  char retazec[11];

  clrscr();

  printf("Zadaj slovo(max. 10 znakov):");
  scanf("%s", retazec);          // nacita slovo do retazca

  p = strlen(retazec);           // vrati pocet znakov

  printf("\nRetazec '%s' ma %d znakov.", retazec, p);

  getch();
}
```

2.16.2 Kopírovanie reťazca

```
char *strcpy(char *s1, char *s2);
```

Skopíruje obsah reťazca **s2** do **s1**. Vracia pointer na prvý znak reťazca **s1**.

Napr.: `strcpy(ret, "ahoj");` v `ret` bude "`ahoj`"

Príklad 50

Program skopíruje zadaný reťazec do ďalšieho reťazca.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
```

```
{
  int p;
  char retazec[11], ret[11];

  clrscr();

  printf("Zadaj slovo(max. 10 znakov):");
  scanf("%s", retazec);          // nacita slovo do retazca

  strcpy(ret, retazec);          // skopiruje retazec do ret

  printf("\n1.Retazec: %s", retazec);
  printf("\n2.Retazec: %s", ret);

  getch();
}
```

2.16.3 Spojenie reťazcov

```
char *strcat(char *s1, char *s2);
```

Prípoji **s2** k reťazcu **s1**. Vracia pointer na prvý znak reťazca **s1**.

Napr.: `strcat(retazec, "+ nazdar");`

Príklad 51

Program po zadaní slova priloží k tomuto slovu do reťazca text: "+ nazdar".

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
  int p;
  char retazec[20];

  clrscr();

  printf("Zadaj slovo(max. 10 znakov):");
  scanf("%s", retazec);          // nacita slovo do retazca

  strcat(retazec, " + nazdar"); // skopiruje retazec do ret

  printf("\nRetazec: %s", retazec);

  getch();
}
```

2.16.4 Nájdenie znaku v reťazci

```
char *strchr(char *s, char c);
```

Pokiaľ sa znak v premennej c vyskytuje v reťazci s, tak je vrátený pointer na jeho prvý výskyt, inak je vrátená hodnota NULL.

Napr.: `strchr(retazec, 'l')`

Príklad 52

Program hľadá v zadanom reťazci znak x.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main()
{
  int p;
  char retazec[11];

  clrscr();

  printf("Zadaj slovo(max. 10 znakov):");
  scanf("%s", retazec);          // nacita slovo do retazca

  if((strchr(retazec, 'x')) == NULL)
    printf("\nV retazci %s sa nenachadza znak %c", retazec, 'x');

  else
    printf("\nV retazci '%s' sa nachadza znak %c", retazec, 'x');

  getch();
}
```

2.16.5 Nájdenie podreťazca v reťazci

```
char *strstr(char *s1, char *s2);
```

Nájde prvý výskyt reťazca **s2** v reťazci **s1** a vráti pointer na tento výskyt alebo vráti NULL v prípade neúspechu.

2.16.6 Práca s obmedzenou časťou reťazca

V štandardných knihovnach sú tiež implementované funkcie, ktoré nemusia pracovať s celým reťazcom (teda až do ukončujucého znaku \0), ale iba so zadaným počtom jeho prvých znakov. Tieto funkcie vypadajú podobne ako funkcie, ktoré sme už spomenuli, ale s tým rozdielom, že majú v názve písmeno "n" – ako *number*.

Funkcie spracovávajú reťazec buď do zadanej dĺžky alebo do ukončujúceho znaku \0.

Príklad funkcie, ktorá skopíruje zadaný počet reťazca **s2** do reťazca **s1**:

```
char *strncpy(char *s1, char *s2, int max);
```

Napr. po príkaze `strncpy(retazec, "alkoholicke", 7);`
bude v reťazci reťazec "`alkohol`"

2.16.7 Práca s reťazcom pospiatky

Ďalšia množina funkcií pracuje s reťazcom "odzadu". Tieto funkcie sú opäť veľmi podobné základným funkciám s tým rozdielom, že majú vo svojom názve písmeno "r" – ako *reverze*. Rozdiel v ich práci je, že reťazec nespracováva od začiatočnej adresy reťazca, ale od adresy jeho koncového znaku \0 smerom k začiatku reťazca.

Príklad funkcie, ktorá hľadá znak, ktorý je v premennej c v reťazci:

```
int strrchr(char *str, char c);
```

2.17 Grafický režim

V grafickom režime sú informácie na obrazovke zobrazované pomocou jednotlivých elementárnych bodov – ich rozsvietením alebo zhasnutím, respektíve zafarbením. Programátor teda môže pristupovať k jednotlivým bodom na obrazovke a „vyfarbovať“ ich podľa svojej potreby.

Práca s grafikou v Borland C++ 3.1 je dosť jednoduchá vďaka množstvu jednoducho použiteľných funkcii. Myslím, že netreba pripomínať, že tieto funkcie boli vyvíjané firmou Borland pre operačný systém MS-DOS a pod iným systémom fungovať asi nebudú.

Všetky funkcie na prácu s grafikou sú deklarované v tomto hlavičkovom súbore **graphics.h**.

2.17.1 Inicializácia grafického režimu

MS-DOS normálne nedokáže zobrazovať grafiku. Ak chceš pri programovaní používať funkcie pre grafický režim, musíš ho najprv inicializovať. Na inicializáciu slúži funkcia `initgraph`.

```
initgraph(int *graphdriver, int *graphmode, char *path)
```

- `graphdriver` : pointer na kód grafického driveru.

Funkcia `initgraph` vie pracovať s rôznymi grafickými kartami. Tento parameter určuje ktorý driver (ovládač) má použiť. Ak tento pointer ukazuje na hodnotu DETECT (symbolická konštanta definovaná v súbore graphics.h), funkcia `initgraph` si automaticky zistí typ grafického adaptéru a použije príslušný driver. Ostatné hodnoty sú popísané v tabuľke, ale asi ich nikdy nevyužiješ:

Konštanta	Číslo
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7

ATT400	8
VGA	9
PC3270	10

- `graphmode` : pointer na kód grafického módu.

Ak si predchádzajúcu hodnotu nastavil na DETECT, túto hodnotu si funkcia `initgraph` nastaví sama. Tieto hodnoty sú uvedené v ďalšej tabuľke, ale tiež ich asi nikdy nevyužiješ:

Graphics driver	Graphics modes	Číslo	Rozlíšenie	Paleta	Počet stránok
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 farby	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 farby	1
	MCGAHI	5	640 x 480	2 farby	1
EGA	EGALO	0	640 x 200	16 farieb	4
	EGAHI	1	640 x 350	16 farieb	2
EGA64	EGA64LO	0	640 x 200	16 farieb	1
	EGA64HI	1	640 x 350	4 farby	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 farby	1
HERC	HERCMONOHI	0	720 x 348	2 farby	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 farby	1
	ATT400HI	5	640 x 400	2 farby	1
VGA	VGALO	0	640 x 200	16 farieb	2
	VGAMED	1	640 x 350	16 farieb	2
	VGAHI	2	640 x 480	16 farieb	1
PC3270	PC3270HI	0	720 x 350	2 farby	1
IBM8514	IBM8514LO	0	640 x 480	256 farieb	
	IBM8514HI	1	1024 x 760	256 farieb	

- `path` : textový reťazec obsahujúci cestu k súboru s príslušným driverom (definovanom na adrese `graphdriver`). Ak je tu uvedený prázdny reťazec (konštanta *“”*), hľadá potrebný súbor v aktuálnom adresári. Tento súbor je potrebný pri každom spustení programu (**aj po kompilácii !!!**). Ty asi budeš potrebovať súbor **egavga.bgi**. Nájdeš ho v adresári C:\BORLANDC\BGI (je určený pre karty EGA a VGA , driver pre SVGA nie je súčasťou Borland C++ 3.1), ale dá sa zohnať na internete.

Teraz si to ukážeme ako to vyzerá všetko spolu:

```
int gdriver=DETECT, gmode;            // nastavenie autodetekcie
initgraph(&gdriver, &gmode, "");       // musi tam byt znak &
```

Pokiaľ sme nezadali žiadnu cestu tak program bude hľadať súbor v aktuálnom adresári.

2.17.2 Uzatvorenie grafického režimu

```
closegraph();
```

Na konci programu treba grafický režim uzavrieť. Slúži na to funkcia `closegraph`.

2.17.3 Príkaz cleardevice

```
cleardevice();
```

Vymaže grafickú obrazovku (`clrscr` funguje len v textovom režime).

2.17.4 Príkaz putpixel

```
void putpixel(int x, int y, int color);
```

Nakreslí jeden bod na súradnice [x, y] s farbou color.

2.17.5 Príkaz setcolor

```
void setcolor(int color);
```

Nastaví farbu čiary pre nasledujúce funkcie (štandardná je biela).

2.17.6 Príkazy outtext a outtextxy

```
outtext(text);
```

Vytlačí grafický text s nastavenými atribútmi na štandardnú pozíciu. Teraz si ukážeme predchádzajúce príkazy v programe.

Príklad 53

Program vypíše zadaný text najskôr na štandardnú pozíciu červenou farbou a počká na stlačenie klávesy, a potom vymaže celú obrazovku a vypíše na súradnice [x, y] zadaný text zelenou farbou a počká na stlačenie klávesy.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

void main()
{
```

```
  int gdriver = DETECT, gmode, errorcode;

  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if(errorcode != grOk){
    clrscr();
    printf("Chyba grafiky: %s\n", grapherrormsg(errorcode));
    printf("Stlac klavesu pre ukoncenie:");
    getch();
    exit(1);        // predcasne ukonci program
  }

  setcolor(RED);
  outtext("Text vypisany na standardnu poziciu.");
  getch();

  cleardevice();
  setcolor(GREEN);
  outtextxy(100,100,"Text vypisany na poziciu [100, 100]");
  getch();

  closegraph(); // ukonci graf. rezim
}
```

Príklad 54

Program vypíše v grafickom režime na zadané súradnice zadaný text a na štandardnú pozíciu vypíše aké boli zadané súradnice.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

void main()
{
  char text[50];
  char surad[20];
  int x, y;
  int gdriver = DETECT, gmode, errorcode;

  printf("Zadaj text: ");
  scanf("%s", text);
  printf("Zadaj X-ovu suradnicu: ");
  scanf("%d", &x);
  printf("Zadaj Y-ovu suradnicu: ");
  scanf("%d", &y);

  initgraph(&gdriver, &gmode, "d:\\bc_plus\\bgi");

  errorcode = graphresult();
  if(errorcode != grOk){
    clrscr();
    printf("Chyba grafiky: %s\n", grapherrormsg(errorcode));
    printf("Stlac klavesu pre ukoncenie:");
    getch();
    exit(1);        // predcasne ukonci program
```

```
  }

  setcolor(BLUE);
  outtextxy(x, y, text);

  sprintf(surad, "Zadane suradnice su  x:%3d y:%3d", x, y);

  setcolor(RED);
  outtext(surad);
  getch();
  closegraph();
}
```

V programe som použil nový príkaz `sprintf`, ktorý slúži na vytvorenie reťazca. Ak by som ho nepoužil tak by som nemohol vypísať pomocou grafického režimu zadané súradnice, pretože v príkazoch `outtext` a `outtextxy` nemôžeme použiť %d atď., tak ako sme to používali pri príkaze `printf`.

2.17.7 Príkazy getmaxx a getmaxy

`int far getmaxx(void);` -vráti maximálnu **x** -ovú súradnicu v grafickom režime

`int far getmaxy(void);` -vráti maximálnu **y** -ovú súradnicu v grafickom režime

Napr.:
```
mx = getmaxx();
my = getmaxy();
```

2.17.8 Príkazy line, lineto a moveto

`line(x1, y1, x2, y2);`	vykreslí čiaru od bodu [x1, y1] po bod [x2, y2]
`moveto(x, y);`	nastaví štandardnú pozíciu na bod [x, y]
`lineto(x, y);`	vykreslí čiaru zo štandardnej pozície na pozíciu [x, y]

Príklad 55

Program vykresli čiary najskôr pomocou príkazu `line` a potom pomocou `lineto`.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

void main()
{
  int mx, my;
  int gdriver = DETECT, gmode, errorcode;

  initgraph(&gdriver, &gmode, "");
```

```
  errorcode = graphresult();
  if(errorcode != grOk){
    clrscr();
    printf("Chyba grafiky: %s\n", grapherrormsg(errorcode));
    printf("Stlac klavesu pre ukoncenie:");
    getch();
    exit(1);        // predcasne ukonci program
  }

  mx = getmaxx();
  my = getmaxy();

  setcolor(BLUE);
  line(0, 0, mx, my);
  getch();
  setcolor(RED);
  line(0, my, mx, 0);
  getch();

  setcolor(GREEN);
  moveto(mx/2, my/2); //nastavi poziciu do stredu obrazovky
  lineto(mx/2, 0);
  getch();
  lineto(0, my/2);
  getch();
  lineto(mx/2, my);
  getch();
  lineto(mx, my/2);
  getch();
  lineto(mx/2, 0);

  getch();
  closegraph();
}
```

2.17.9 Príkaz setfillstyle

```
void setfillstyle(int pattern, int color);
```

Nastaví vzor (pattern) a farbu (color) výplne (štandardný vzor je 1).

2.17.10 Príkazy circle, ellipse a rectangle

```
void circle(int x, int y, int radius);
```

Vykreslí na súradnice [x, y] kružnicu s priemerom radius vyplnenú nastavenou farbou a vzorom

```
void ellipse(int x, int y, int a, int b, int r1, int r2);
```

Vykreslí elipsu vyplnenú nastavenou farbou a vzorom

```
void rectangle(int left, int top, int right, int bottom);
```

Vykreslí obdĺžnik vyplnený nastavenou farbou a vzorom.

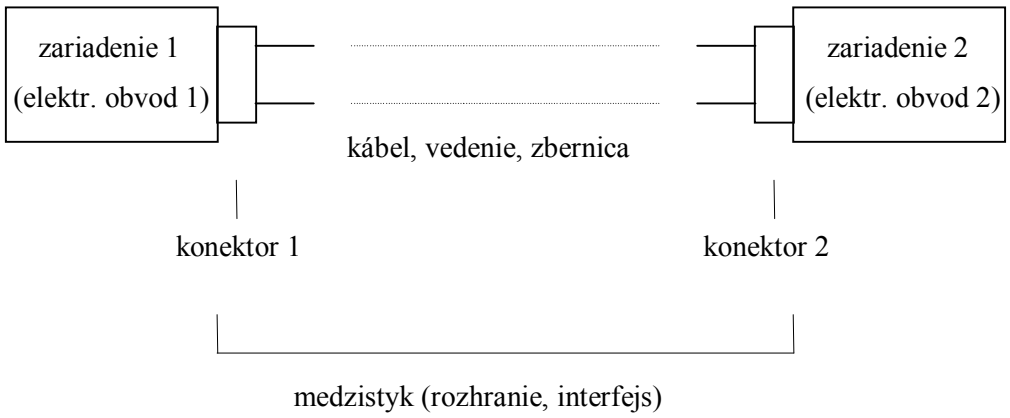
Príklad 56

Program najskôr vykreslí na obrazovku kružnicu, potom elipsu a nakoniec obdĺžnik.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

void main()
{
  int mx, my;
  int gdriver = DETECT, gmode, errorcode;

  initgraph(&gdriver, &gmode, "");

  errorcode = graphresult();
  if(errorcode != grOk){
    clrscr();
    printf("Chyba grafiky: %s\n", grapherrormsg(errorcode));
    printf("Stlac klavesu pre ukoncenie:");
    getch();
    exit(1);        // predcasne ukonci program
  }

  setcolor(BLUE);
  circle(200, 200, 100);
  getch();

  setcolor(RED);
  ellipse(200,300,100,10,20,100);
  getch();

  setcolor(GREEN);
  rectangle(10,10,500,300);

  getch();
  closegraph();
}
```

3. Prenos údajov

3.1 Základné pojmy



Medzistyk (rozhranie, interface) – stanovené prepojenie subjektov (2 alebo viacerých) v oblasti prenosu informácií na fyzickej úrovni (na úrovni *signálov* a *signálnych sledov*).

Protokol prenosu (na fyzickej úrovni, na medzistyku) - stanovené správanie sa na medzistyku.

Signál - stanovený význam určenej (napr. elektrickej) veličiny na príslušnom prenosovom médiu (napr. elektrickom vodiči).

Jednotka dátovej informácie (napr. 1 bajt = 8 bitov)

Paralelný prenos - je taký prenos dát, kde prenos všetkých jednotiek dátovej informácie sa vykonáva SÚČASNE vzhľadom k charakteristickému okamihu prenosu dátovej informácie.

Sériový prenos - je taký prenos dát, kde prenos všetkých jednotiek dátovej informácie sa vykonáva POSTUPNE ZA SEBOU (sekvenčne) vzhľadom k charakteristickému okamihu prenosu dátovej informácie.

Sériový port sa označuje **COMx** (napr. COM1, COM2…). Štandardne sa v PC nachádzajú dva sériové porty.
x – znamená číslo portu

Paralelný port sa označuje **LPTx** (napr. LPT1, LPT2…). Štandardne sa v PC nachádza jeden alebo dva paralelné porty.
x – znamená číslo portu

3.2 Paralelný prenos dát

3.2.1 IRPR

IRPR - prenos dát s hladinovou synchronizáciou
(tiež sa nazýva ŠTART - STOPNÝ prenos)

Základné informácie:

- potvrdzovanie prenosu sa vykonáva nastavovaním a sledovaním hladín logických úrovní signálov
- signály majú logické úrovne kompatibilné s TTL logikou
- aktívna úroveň všetkých signálov je "L" (0 - 0.4 V)
- používa sa prenos väčšinou 8 bitový (niekedy aj 16 bitový, prípadne iný)

3.2.2 Centronics

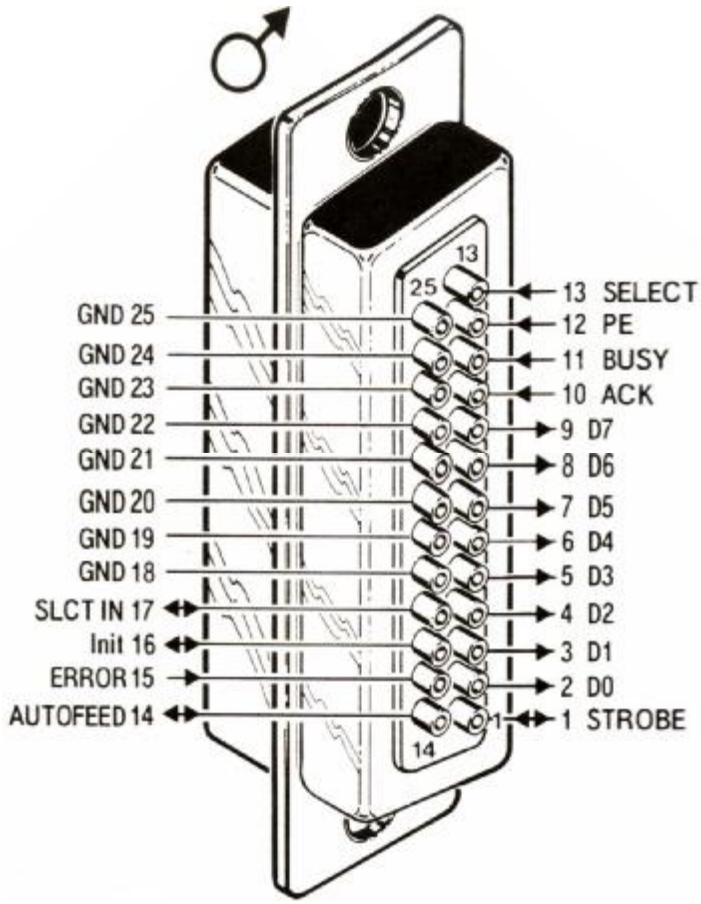
CENTRONICS - prenos dát s impulzovou synchronizáciou z roku 1981, ktorý sa používa aj v súčasnosti.

Základné vlastnosti:

- používa sa na pripájanie paralelných zariadení (pôvodne navrhnuté rovnomennou firmou pre pripájanie tlačiarní k počítačom, t.j. iba výstupne rozhranie)
- používa úrovne signálov TTL logiky
- potvrdzovanie prenosu dátových informácií sa vykonáva pomocou impulzov riadiacich signálov
- prenos dát je 8-bitový

Používané signály:

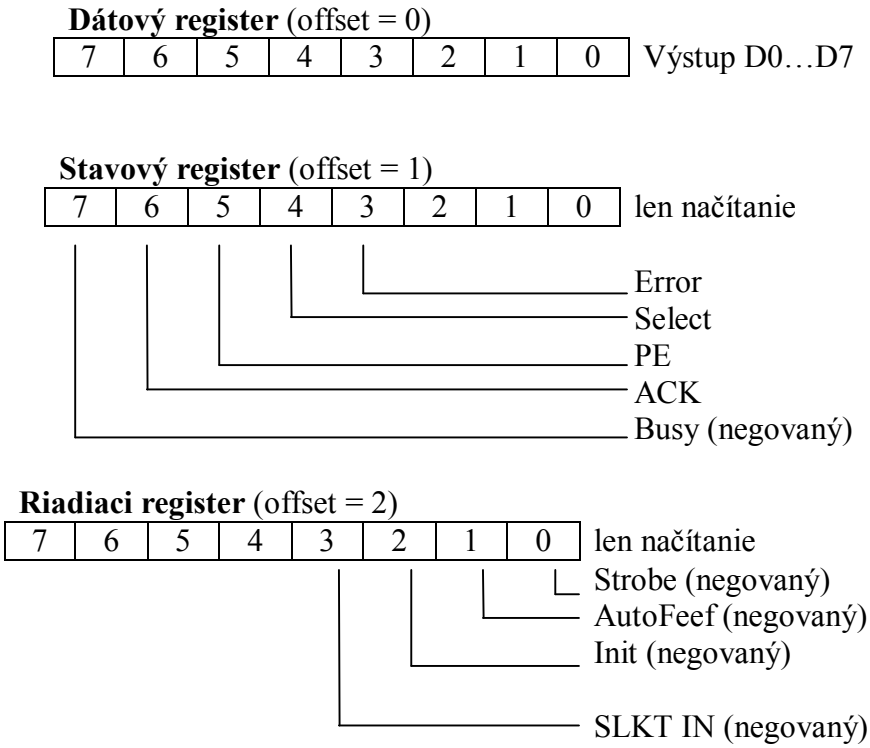
- dátové vedenie: 8 dátových vodičov: *DATA1 - DATA8*
- riadiace signály:
  - *STROBE* - impulz určujúci začiatok platnosti dát
  - *AUTO FEED XT* - automatické od riadkovanie (vloží LF za každé CR)
  - *SLCT IN* - výber zariadenia pre prenos dát = oznamuje tlačiarni, že je vybratá
  - *INIT* - nulovanie
- stavové signály:
  - *ACKNLG* - impulz určujúci pripravenosť prijímača prevziať ďalší bajt dát
  - *BUSY* - hladinový signál, ktorý informuje vysielač, že prijímač spracováva prenesený bajt dát
  - *PE* - koniec papiera (spolu s -ERROR)
  - *SELECT* - stav ON-LINE
  - *ERROR* – chyba



Toto je konektor paralelného portu a v nasledujúcej tabuľke sa nachádzajú vypísané bity portu:

PIN	Vysiela > / Príma <
D0 ÷ D7	>
STROBE	>
ACK	<
BUSY	<
PE	<
SELECT	<
AUTO FEED	>
ERROR	<
INIT	>
SLCT IN	>

Register	Offset	Poznámky
Dátový register	0	Výstup dát na tlačiareň
Stavový register	1	Čítanie stavu tlačiarne
Riadiaci register	2	Riadenie funkcií tlačiarne



V nasledujúcej tabuľke sa nachádza adresa dátového registra paralelného portu (hexadecimálne), na ktoré v jazyku C posielame údaje.

	LPT1	**LPT2**
Hexadecimálne	378	278
Dekadický	888	632

Jazyk C pozná 8, 10 a 16 sústavu:
- čísla v 8 sústave sa zapisujú 0o... (napr. 0o153)
- čísla v 16 sústave sa zapisujú 0x... (napr. 0x378, 0x278)

Ak si nepochopil predchádzajúce obrázky ako súvisia s uvedenou tabuľkou, tak ti to zhrniem pre istotu ešte raz:
Stavový register je na adrese **dátový register + 1**, teda 0x378+1
Riadiaci register je na adrese **dátový register + 2**, teda 0x378+2
Data0 ÷ Data7 – **bit 0. ÷ bit 7** dátového registra

Aby si si nemusel stále pamätať, že stavový register je na adrese, môžeš si spraviť tzv. **konštanty**. Tie budú ukázané na najbližšom príklade 57 o chvíľu.

Do dátového registra posielame údaje. Stavový register nám oznamuje stav tlačiarne a pomocou riadiaceho registra riadime tlačiareň.

3.2.3 Príkazy inp a outp

Príkaz `inp` slúži na čítanie údajov (dát) zo zariadenia (napr. tlačiarne) cez hardwarový port. Syntax tohto príkazu je:

```
inp(adresa portu);
```

Návratová hodnota, ktorú vracia príkaz `inp` sú načítané dáta, s ktorými môžeme ďalej pracovať

Príkaz outp slúži na posielanie údajov (dát) na zariadenie (napr. tlačiareň) cez hardwarový port. Syntax tohto príkazu je:

```
outp (adresa portu, vysielane data);
```

Ešte jedna poznámka: **!!! VŠETKY REGISTRE SÚ 8 BITOVÉ !!!**

Príklad 57

Na začiatok nám bude stačiť na názornú ukážku nejaký prípravok, ktorý bude obsahovať LEDky, pripojené na Data0 ÷ Data7, STROBE, ACK a BUSY. Teraz nám budú stačiť len Data.

Predpokladám, že na LPT1 máš nejaký prípravok (môžeš použiť aj ihličkovú tlačiareň, ale pre 100% funkčnosť programu nevysielaj zatiaľ viac ako 1Bajt, pretože ešte nevieš ako rýchlo tlačiareň prečíta údaje z dátového registra...). Nasledujúci program nám zapíše do 5.tého bitu dátového registra 1 a následné sa nám rozsvieti LED s označením D5 a počká na stlačenie ľubovoľnej klávesy, a potom rozsvieti LEDky D4 a D1 súčasne.

```
#include <dos.h>
#include <conio.h>

#define DAT_REG 0x378
#define D0 0x1          // dekadicky 1   0.bit
#define D1 0x2          // dekadicky 2   1.bit
#define D2 0x4          // dekadicky 4   2.bit
#define D3 0x8          // dekadicky 8   3.bit
#define D4 0x10         // dekadicky 16  4.bit
#define D5 0x20         // dekadicky 32  5.bit
#define D6 0x40         // dekadicky 64  6.bit
#define D7 0x80         // dekadicky 128 7.bit

void main()
{
  outp(DAT_REG, D5);
  getch();
  outp(DAT_REG, D4|D1);
  getch();
}
```

V programe som použil nový príkaz `#define`, ktorý slúži na zadefinovanie konštanty. Podľa nepísanej dohody sa píšu tieto konštanty veľkými písmenami a premenné malými písmenami, aby sme si to rozlíšili a nemuseli sme rozmýšľať, či je to konštanta alebo premenná.

Konštanta slúži na uľahčenie práce, pretože ľahšie si zapamätám názov konštanty ako hodnotu, ktorú predstavuje konštanta. Aj keď chcem hodnotu neskôr zmeniť tak mi stačí zmeniť hodnotu na začiatku a nemusím prezerať celý program

Príklad 58

Program bude rozsvecovať LEDky sprava doľava, a potom naspäť s oneskorením 300ms. Program sa bude vykonávať donekonečna, respektíve do stlačenia ľubovoľnej klávesy.

```
#include <dos.h>
#include <conio.h>

#define DAT_REG 0x378
#define D0 0x1          // dekadicky 1   0.bit
#define D1 0x2          // dekadicky 2   1.bit
#define D2 0x4          // dekadicky 4   2.bit
#define D3 0x8          // dekadicky 8   3.bit
#define D4 0x10         // dekadicky 16  4.bit
#define D5 0x20         // dekadicky 32  5.bit
#define D6 0x40         // dekadicky 64  6.bit
#define D7 0x80         // dekadicky 128 7.bit

int cas = 300;
void main()
{
  while(1){
    if( kbhit() ) break;

    outp(DAT_REG, D0);
    delay(cas);
    outp(DAT_REG, D1);
    delay(cas);
    outp(DAT_REG, D2);
    delay(cas);
    outp(DAT_REG, D3);
    delay(cas);
    outp(DAT_REG, D4);
    delay(cas);
    outp(DAT_REG, D5);
    delay(cas);
    outp(DAT_REG, D6);
    delay(cas);
    outp(DAT_REG, D7);
    delay(cas);

    // naspat
    outp(DAT_REG, D6);
    delay(cas);
    outp(DAT_REG, D5);
    delay(cas);
    outp(DAT_REG, D4);
    delay(cas);
    outp(DAT_REG, D3);
    delay(cas);
    outp(DAT_REG, D2);
    delay(cas);
    outp(DAT_REG, D1);
    delay(cas);
  }
}
```

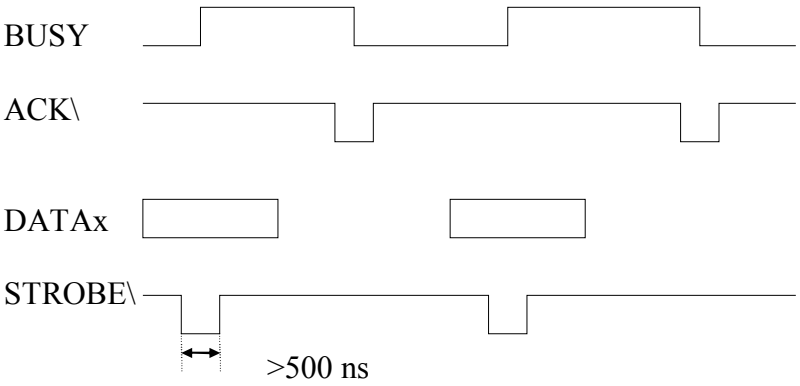
Príkaz `delay` slúži na zdržanie programu. Zadané číslo znamená na koľko milisekúnd sa program zastaví. Podobný je aj príkaz `sleep`, v ktorom zadané číslo znamená zdržanie v sekundách. Napr.:

```
delay(200);          // zdrží program 200ms
sleep(3);            // zdrži program 3s
```

3.2.4 *Výstup na tlačiareň cez paralelný port*

Bity BUSY a ACK vysiela tlačiareň. Bity DATAx a STROBE vysiela PC cez port. To lomítko za ACK a STROBE znamená, že tieto bity pracujú inverzne, t.j. keď je 0 tak vysielajú informáciu.

Princíp ako to celé funguje je jednoducho znázornený na obrázku.



Popis obrázku (princíp):

Keď je bit **BUSY na 0** znamená to, že tlačiareň je pripravená prijať údaje pre tlač. Teraz môžeme pripravené údaje vyslať na port (na bity DATA0 ÷ DATA7). Ak sme ich vyslali a chceme aby ich tlačiareň prijala, tak musíme jej oznámiť, že poslané údaje na porte sú platné. Musíme aspoň na 500ns nastaviť **STROBE na 0**. Rýchlosť odpovede tlačiarne závisí od typu tlačiarne. Toto časové oneskorenie zabezpečíme napr. príkazom `delay(1)`, ktorý počká 1ms (je to aj s rezervou). Tlačiareň v tomto časovom intervale stihne zareagovať a nastaví bit **BUSY na 1** a znamená to, že nesmieme na port vyslať žiadne údaje. Keď tlačiareň skončí oznámi nám to nastavením bitu **ACK na 0** a s veľmi malým časovým oneskorením aj nastavením bitu **BUSY na 0** čo znamená, že môžeme na port vyslať ďalšie údaje.

Príklad 59

Program vytlačí zadaný textový súbor na obrazovku počítača a na tlačiarni pripojenej na paralelný port LPT1.

```
#include <dir.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>

#define BA_DR     0x378      // Datovy register

#define BA_SR     BA_DR+1    // Stavovy register
#define ERROR     0X8
#define SELECT    0X10
#define PE        0X20
#define ACK       0X40
#define BUSY      0X80
```

```
#define BA_RR     BA_DR+2    // Riadiaci register
#define STROBE    0X1
#define AUTO_FEED 0X2
#define INIT      0X4
#define SLCT_IN   0X8

void main(void)
{
  char subor[128];        // plny nazov suboru
  FILE *sub;              // popisovac suboru
  char c;                 // premena na prenos znaku
  int  sr;                // pre stavovy register

  clrscr(); // vymaze obrazovku

  printf("Program na opis text. suboru na tlac. a monitor.\n");
  printf("Zadaj subor: ");
  scanf("%s",&subor);// nacita nazov suboru z klavesnice

  // Otvori subor pre citanie v binarnom rezime
  if((sub = fopen(subor, "rb"))==NULL){
    printf("\nNemozem otvorit vstupny subor \"%s\".\n",subor);
    printf("\nStlac lubovolnu klavesu");
    getch();
    return;
  }

  clrscr();               // vymaze obrazovku

  sr=inp(BA_SR);          // nacita stavovy register
  if((sr&SELECT)==0){
    printf("Tlaciaren nie je v stave ON-LINE!\n");
    printf("Stlac lubovolnu klavesu.\r");
    getch();
    return;
  }

  outp(BA_DR,0X0D);            // navrat vozika
  outp(BA_RR,INIT | STROBE);  // STROBE nastavi na 1,udaje su platne
  delay(1);                    // pocka 1ms
  outp(BA_RR,INIT);            // STROBE nastavi naspat na 0
  while(inp(BA_SR)&ACK);       // pocka na odpoved tlaciarne
  while((inp(BA_SR)&BUSY)==0); // caka kym tlac. nenastavi BUSY na 0

  while((c=getc(sub))!=EOF){   // citanie znakov az po koniec suboru
    putch(c);                  // zobrazenie znaku na obrazovke

    if(c=='\n'){          // ak je subor otvoreny v text. rezime
      outp(BA_DR,0x0D);          // navrat vozika na zaciatok riadku
      outp(BA_RR,INIT | STROBE);      //STROBE nastavi na 1
      delay(1);                       // pocka 1ms
      outp(BA_RR,INIT);          // STROBE nastavi naspat na 0
      while(inp(BA_SR)&ACK);          // pocka na odpoved tlaciarne
      while((inp(BA_SR)&BUSY)==0);    // caka kym BUSY nie je 0
      outp(BA_DR,0x0A);               // novy riadok
```

```
      outp(BA_RR,INIT | STROBE);      // STROBE nastavi na 1
      delay(1);                       // pocka 1ms
      outp(BA_RR,INIT);               // STROBE nastavi naspat na 0
      while(inp(BA_SR)&ACK);          // pocka na odpoved tlaciarne
      while((inp(BA_SR)&BUSY)==0);    // caka kym BUSY nie je 0
    }

    else{
      outp(BA_DR,c);             // zapise znak do datoveho registra
      outp(BA_RR,INIT | STROBE);      //STROBE nastavi na 1
      delay(1);                       // pocka 1ms
      outp(BA_RR,INIT);               // STROBE nastavi naspat na 0
      while(inp(BA_SR)&ACK);          // pocka na odpoved tlaciarne
      while((inp(BA_SR)&BUSY)==0);    //caka kym BUSY nie je 0
    }
 }

 fclose(sub); // zatvori subor

 printf("\nStlac lubovolnu klavesu");
 getch();
}
```

3.3 Sériový prenos dát

Jazyk C dokáže so zariadeniami komunikovať aj sériovo cez sériový port, ktorý sa nachádza na každom osobnom PC. Komunikovať so zariadeniami cez sériový port je však trochu zložitejšie, pretože obidve zariadenia (PC a ...) musia komunikovať rovnakou rýchlosť, inak budú prenášané údaje skreslené.

4. Záver

Učebnica, ktorú práve držíte v ruke, je súčasťou práce, ktorej cieľom bolo vytvoriť voľne dostupnú textovú príručku pre žiakov na Strednej priemyselnej školy v Bánovciach nad Bebravou, ktorí sa učia na hodinách výpočtovej techniky programovať v jazyku C a doteraz nemali učebnicu, z ktorej by sa mohli učiť, a preto všetko museli písať do zošita a na samotné programovanie zostávalo veľmi málo času, preto som začal písať túto učebnicu. Viem, že už existuje veľa príručiek na programovanie, ale nie také, ktoré by boli veľmi vhodné pre prácu našich žiakov, ktorí sa so samotným programovaním stretávajú prvý raz na hodinách výpočtovej techniky. Veľa príručiek je napísaných nie pre úplných začiatočníkov, ale pre tých, ktorý už vedia základy z programovania v jazyku C alebo vedia programovať v podobnom programovacom jazyku ako je programovací jazyk PASCAL a majú už z čoho vychádzať.

Táto učebnica (príručka) však môže poslúžiť aj ostatným čitateľom, ktorí nemajú skúsenosti z programovacieho jazyka C. Učebnica nie je učebnicou programovania, ale učebnicou jazyka C, pretože sa v nej oboznamujete s príkazmi, funkciami a konštrukciami programovacieho jazyka C.

Ako príloha k tejto učebnici sa prikladajú všetky zdrojové súbory programov z príkladov.

Táto učebnica je moja práca na súťaž v Stredoškolskej odbornej činnosti (SOČ) v roku 2004 v kategórií učebné pomôcky. Napísal som ju ja - Lukáš Vanek- žiak 4.A triedy na SPŠ v Bánovciach nad Bebravou.

P.S.: Prípadné gramatické chybičky a preklepy prosím ospravedlňte.

5. Použitá a doporučená literatúra

Borland C++

Návod k použití
vydavateľstvo: KOPP, České Budějovice
autor: Ing. Pavel Herout
rok vydania: 1994
počet strán: 188
Prvé vydanie

Borland C++

V příkladech
vydavateľstvo: GRADA, Praha
autor: Jaromír Kukal
Milan Jelen
rok vydania: 1994
počet strán: 232
Prvé vydanie

Jazyk C

Učebnica pre stredné školy
vydavateľstvo: Computer Press, Praha
autor: Dalibor Kačmár
rok vydania: 2000
Prvé vydanie

Jazyky C a C++ podle normy ANSI/ISO

Kompletní kapesní průvodce
vydavateľstvo: GRADA, Praha
autor: Dirk Luis
Petr Mejzlík
Miroslav Virius
rok vydania: 1999
počet strán: 644

Programovací jazyk C

Nazývaná aj bibliou C
autor: Brian W. Kernighan
Denis M. Ritchie

Programátorský Babylon

vydavateľstvo: GRADA, Praha
autor: Z. Fikar
Frimlová
J. Kukal
R. Letoš
rok vydania: 1992
počet strán: 323

Turbo C a Borland C++

Popis prostredia
vydavateľstvo: GRADA, Praha
autor: Karel Nenadál
Dana Václiková
rok vydania: 1992
počet strán: 136
Prvé vydanie

Učebnice jazyka C

vydavateľstvo: KOPP, České Budějovice
autor: Ing. Pavel Herout
rok vydania: 1992
počet strán: 272
Prvé vydanie

6. Obsah