**Programování v jazyce C**

* Vyvinul ho Dennis Ritchie a Ken Thompson v roce 1970
* V minulosti to byl vyšší programovací jazyk, dnes se ale považuje za výkonější nízkoúrovňový jazyk
* Byl navržen pro operační systém unix
* Velmi dobře pracuje s pamětí
* Vyvinula se z něj většina moderních programovacích jazyků -> mají podobnou syntaxi
* Je multiplatformní -> pracuje na většině OS
* Nepodporuje objektově orientované programování
* Neobsahuje žádnou standartní knihovnu pro práci v grafickém rozhraní, pro tyto účely se musí využit knihovny třetích stran
* Neumožňuje práci s textovými řetězci
* Je to kompilační jazyk tudíž vyžaduje kompilátor (překladač)
* Norma jazyka tzv. ANSI C pochází z roku 1989 a podporuje ji překladačů. Poslední standard C99 se od ANSI C nijak zvlášť neliší -> Proměnné mohou být deklarovány kdekoliv

Přídán datový typ bool, long long int (64bit)

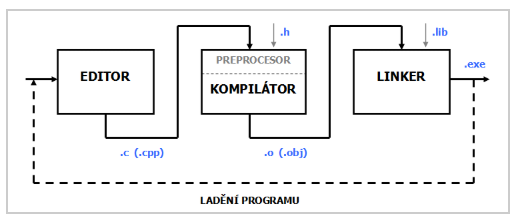
Podpora zakomentování jednoho řádku //

Pole s nekonstantní velikostí

Dat. Typ complex reprezentující komplexní číslo

Nové hlavičkové soubory (např. stdint.h)

**Zpracování programu v jazyce C**



EDITOR - program v jazyce C, tzv. zdrojový kód, můžeme napsat v jednoduchém textovém editoru. Jeho výstupem je soubor s příponou .c (případně .cpp). Je vhodné nepoužívat v názvech souborů české znaky a mezery.

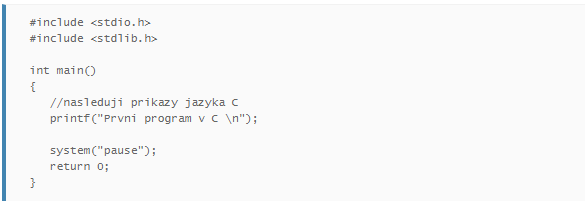
KOMPILÁTOR - PŘEKLADAČ - textový soubor nejdřív zpracuje preprocesor, který zdrojový kód částečně upraví. Vloží hlavičkové soubory, odstraní komentáře, nahradí symbolické konstanty a pod. Preprocesor je součástí překladače. Potom překladač přeloží příkazy, vznikne tzv. relativní kód (s příponou .obj nebo .o). V případě, že se v programu vyskytnou formální chyby, je nutné je opravit a kód znovu zkompilovat.

LINKER - SESTAVOVACÍ PROGRAM - poslední fází je propojení relativního kódu a knihovních souborů (.lib). Výsledkem je spustitelný soubor (ve Windows s příponou .exe).

Pokud spustitelný program nefunguje podle našich představ - dává např. špatné výsledky, musíme program přepsat. Procesu hledání logických chyb říkáme ladění programu. K tomu je možné využít ladící program tzv. debugger.

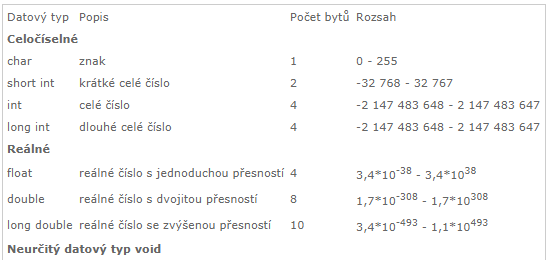
Editor, překladač a linker jsou většinou přímo součástí vývojového prostředí např. Visual Studio, Bloodshed Dev-C++, Code::Blocks, NetBeans.

**Struktura kódu**

****

**Datové typy**

Určuje: rozsah přípustných hodnot a povolené operace



Se standardem C99 přibyl typ bool (true,false), long long int (8B), complex

Rozsah se spočítá 2počet bitů -1

Celočíselné datové typy můžou být ještě se znaménkem (signed) nebo neznaménkové (unsigned). Implicitní nastavení pro celá čísla je signed (neplatí pro char).

Unsigned= datové typy mohou mít pouze 0 a kladné hodnoty, takže unsigned int jde od 0 do

4,294,967,295

Datový typ se používá k incializaci proměnné

Proměnná = místo v paměti, jejíž obsah se může v průběhu výpočtu měnit

Má svůj název a datový typ

Pravidla pro vytvoření názvu(identifikátoru)

-a…z, A….Z, 0….9, \_

-nemuže začínat číslicí

-rozlišují se malá a velká písmena(tzv. case sensitive)

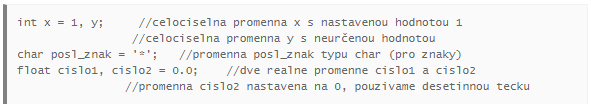
-nesmí se shodovat s klíčovým slovem

-délka názvu bývá omezená(ANSI C rozlišuje prvních 31 znaků, většinou dostačující)

Deklarace proměnné

Pokud chceme nějakou proměnnou použít musí být nejdříve deklarována. Ve starších standardech se deklarace prováděla na začátku bloku příkazů. Od standardu C99 již lze deklarovat kdekoliv v programu. Stále je ale lepši deklarovat na začátku kvůli lepši přehlednosti.

Při deklaraci se prvně uvádí datový typ. Před ním případně, jestli se jedná o unsigned nebo signed. A poté název proměnné. Pokud do proměnné chceme rovnou přiřadit hodnotu napíšeme rovnítko a hodnotu. Pokud se jedná o ukazatel pak se před název ještě přidá \*.



Konstanty

Definují se před mainem. #define n 3.14

#define jmeno „pavel“

Jsou to hodnoty, které se nám v průběhu programu neměnní. Používají se když danou hodnotu používáme v programu vícekrát tak si ji definujeme na začátku a pak pouze píšeme její název místo rozepisování hodnoty. Když ji pak budeme potřebovat změnit změníme jen definici konstanty a změní se nám hodnota všude kde byla použita. Nemusíme je měnit ručně

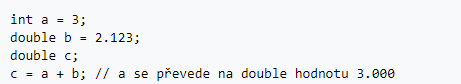
Přetypování

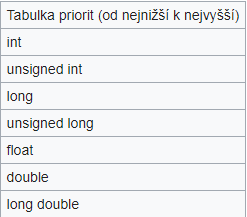
Jedná se o změnu datového typu proměnné

Jsou dva druhy implicitní a explicitní

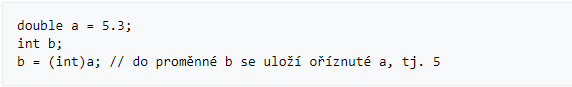
Implictiní: Tzv. automatické. Jedná se o převod datového typu s nižší prioritou na typ s vyšší. To je například z intu na float.

Přetypování vypadá takto





Explicitní: Tzv. vynucené. Převádíme datový typ s vyšší prioritou na ten s nižší. U toho přetypování si musíme dávat pozor na ztrátu dat.



Pozor nemůžeme ale přetypovat int,float na char nebo naopak. Výsledkem by byla nedefinovaná hodnota. Na tohle se používají speciální funkce

atof(), atoi()

funkce knihovny stdlib.h

převod z char na int nebo float

syntaxe: int a = atoi(char \*b);

sprintf()

funkce knihovny stdio.h

převod z jakéhokoliv typu na pole charu(string)

syntaxe podobná jako u printf() = sprintf(char \*vys,“%d“,a);

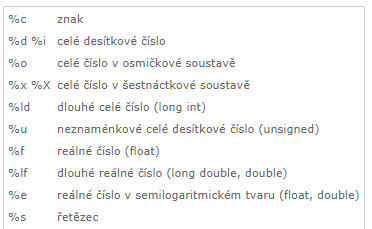
**Vstupy a výstupy**

Zajišťuje je knihovna stdio.h

Scanf(), printf();

**Scanf**

-Načte hodnoty z klávesnice.

-Syntaxe***: scanf(„%d“, &a);*** &, protože přistupujeme k adrese proměnné. %d protože se jedná o int

-při čtení čísel vynechává bíle znaky(enter,mezera,tab) a načte číselnou hodnotu pro další bílý znak

**Printf()**

-vypisuje hodnoty do konzole

-syntaxe: ***printf(„Vysledek je %d“,a);*** když chceme zobrazit proměnnou musíme použít %d a pak na konci za čárkou napsat jméno proměnné, případně proměnných podle toho jak jdou v printf za sebou.

-formátovací znaky= \n – odřádkování

\t – tabulátor

\kód znaku – pokud chceme vložit znak

Pro řetězce (pole znaků) je lepší používat funkce gets() a puts() jsou v knihovně string.h

gets() načte řetězec a puts() ho vypíše.

Při načítání několika řetězců za sebou může dojít k problémů s bílým znakem enter, který pak způsobí že další načtení řetezce se ihned přeskočí. V takovém případě použijeme funkci fflush(stdin), která nám vyčistí buffer. Nachází se v knihovně stdio.h

**Řídící struktury**

Používají logické operátory, operátory inkrementace, dekrementace a ternární operátor

Logické operátory

< menší > větší <=,>= menší, větší nebo rovno (rovnítko vždy druhé)

== je rovno != není rovno && a zároveň || nebo ! nepravda

Inkrementace, dekrementace

i++,i--; Sníží nebo zvýší hodnotu proměnné o jedno. Používá se u smyček

Ternární operátor

(podmínka) ? x:y -> pokud je podmínka pravdivá vrací hodnotu x, jinak y

Tento operátor se nepoužívá v řídících strukturách ale může nahradit jednoduchý if,else

Větvení

If(a>b){

}

Pokud Je podmínka v závorkách pravdivá provede se kód v {} pokud ne tak nic, když ale přidáme za konec }

else {

}

Provede se v případě nepradivé podmínky v závorkách kód v elsu

Můžeme také přidávat další podmínky else if(){}

Switch

Syntaxe:

switch(volba){

case 1: printf(„a“);break;

case 2: printf(„b“);break;

default: printf(„0“);break;

}

Pokud se volba bude rovnat 1 provede se case 1, pokud 2 provede se case 2, jinak se provede default.

Cykly

Máme různé druhy smyček. Smyčka se opakuje dokud je podmínka pravdivá

For,while,do while

While(a<b){}

Kód se bude provádět dokud bude a menší než b

For(i=0;i<pocet;i++){}

Kód se bude provádět dokud i bude menší než počet. i=0 se provádí na úplném začátku smyčky.

I<počet se kontroluje na každém začátku smyčky a i++ se provádí vždy na konci smyčky.

Do{}while(a<b);

Smyčka provede poprvé kód bez toho aniž by pohlížela na podmínku. Pak pojede dokud bude a<b.

Příkazy break; a continue;

Break úplně vyskočí ze smyčky. Continue přeskočí na další cyklus.

**Práce s polem**

Pole je specifický druh proměnné ve které se nachází více než 1 hodnota stejného datového typu.

Int a[12];

Deklaruje se tak že za název proměnné se vloží do hranatých závorek velikost pole.

Pokud hned při deklaraci chceme vložit hodnoty do pole dělá se to tímto způsobem

Int a[5]={10,12,13,14,15};

Pole charů se v Cčku používá jako řetězec.

Zápis do pole a výpis se používá index. Ten začíná 0, protože pole začíná od 0.

Zápis do pole:

for(i=0;i<5;i++){

a[i]=b;

}

Taky můžeme zapisovat pomocí smyčky while dokud nebude třeba zadána hodnota -99 nebo jiná, kterou budeme chtít indikovat konec zadávání hodnot do pole. Mějme na paměti že pro takto ukončení zadávání hodnot musíme zvolit hodnotu, kterou nebudeme chtít normálně používat v poli. Tudíž pokud budeme třeba chtít zadávat čísla na kostce pak může být konečné číslo 0, protože to na kostce nemůže padnout, tudíž ho nebudeme používat v poli.

Výpis:

for(i=0;i<5;i++){

printf(„%d“,a[i]);

}

Můžeme také přistoupit a přepsat určitou hodnotu v poli

a[3]=5;

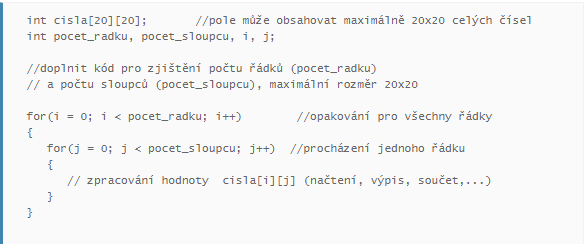
Tímto jsme přepsali 3. hodnotu v poli na 5. (Nezapomínat že se počítá od 0, takže jsme přepsali až 4. hodnotu)

Dvojrozměrné pole

Char sach[8][8];

Při deklaraci dvojrozměrného pole se používají 2 hranaté závorky. Toto pole se používá při hrách jako jsou šachy nebo lodě. Taky se dá použít při poli stringů.

Procházení 2- rozměrného pole se musí použít 2 smyčky. Jedná co bude procházet řádek druhá sloupce.



Může být i 3-rozměrné pole to se deklaruje pomocí 3 hranatých závorek.

**Algoritmy pro třídění pole**

Pro zjistění nejmenší či největší hodnoty. Si před smyčkou nastavíme max nebo min na první hodnotu v poli (min=pole[0]);). Poté vytvoříme smyčku, která bude procházet celým polem a v této smyčku vytvoříme podmínku ( if(min>pole[i]) min=pole[i]; ) Tato podmínka ošetří, že pokud bude aktuální hodnota v poli menší než naše aktuální min hodnota pak se naše minimální hodnota přepíše na tu novou. U maximální je pouze naopak znaménko provnání.

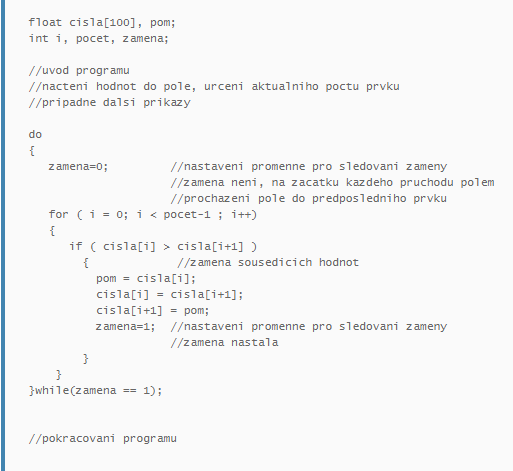
Pro třídění pole se používají nejčastěji 2 algoritmy: Selection sort a Bubble sort

Bubble sort

Tzv. řazení záměnou. Jedná se o jednoduchý algoritmus, vhodný pro malé počty hodnot. Porovnává 2 sousedící hodnoty. Po každém průchodu se největší hodnota dostane na konec.

První varianta

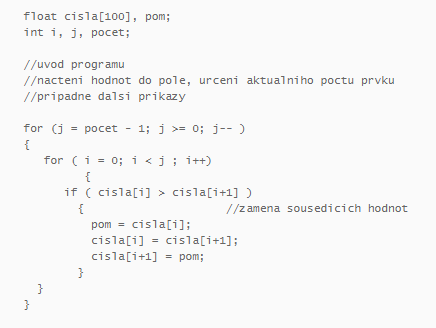
Pole se prochází tak dlouho dokud nebude provedena záměna. Vždy prochází celé pole i když víme že na konci už jsou hodnoty seřazeny. Je efektivní, když už je pole seřazeno. Pak ho projede jen jednou a stačí.



Pro seřazení od největšího prvku pouze prohodíme znaménko v ifu.

Druhá varianta

Výhoda druhé varianty je v tom, že neprochází zbytečně stále celé pole, ale vynechá hodnoty o kterých ví že už jsou seřazeny. Problém nastává, když už jsou hodnoty seřazeny, protože to seřazuje znova i když už jsou a nestačí mu pouze jeden cyklus.

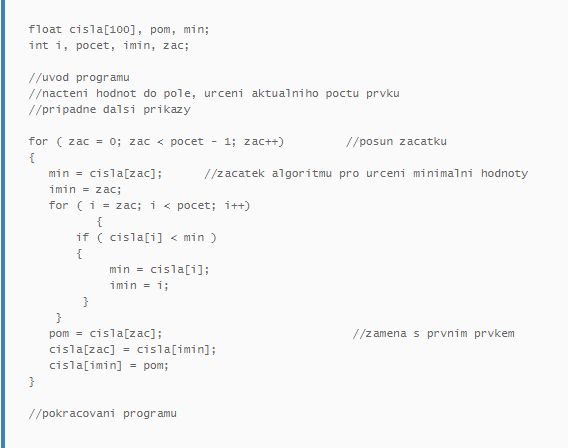


Selection sort – řazení výběrem

-jednoduchý algoritmus vhodný pro malé množství dat

Data jsou rozdělena na seřazená a neseřazená. Jsou dvě varianty výběrem minima a výběrem maxima

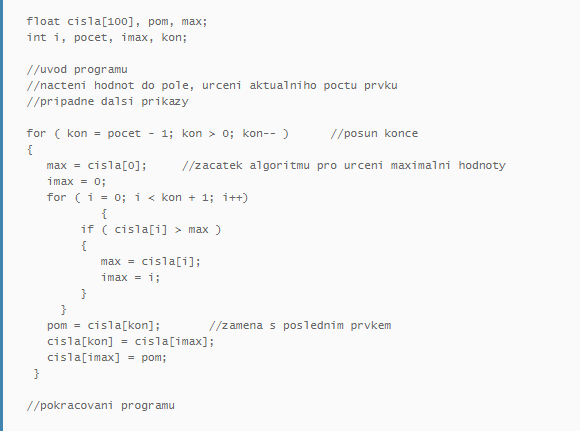
Vyběrem minima

Na začátku si nastavíme min hodnotu na první hodnotu v poli a pak procházíme pole. Když najdeme novou min hodnotu uložíme si její index a přepíšeme min hodnotu. Po doběhnnutí smyčky zaměníme počáteční hodnotu s naší nalezenou minimální.

Výběr minima se dá použít i pro sestupné řazení, když nastavíme min hodnotu na poslední hodnotu v poli a pole budeme procházet opačným směrem a hodnoty přehazovat místo poslední. Přehlednější je ale použít výběr maxima.

Výběr maxima

Zde je postup obrácen jako je popsáno v předchozím kroku pro lepší představení



**Funkce**

Dělí se na standartní funkce, funkce main, vlastní funkce

Standartní funkce – funkce z knihoven (gets(), scanf(), printf())

Funkce main – hlavní funkce

Vlastní funkce

Obsahují ucelenou část programu. Řeší 1 dílčí úkol

Používají se pro opakující se akce. Hodnoty do funkce předáváme přes tzv. parametry

Lokální proměnné - deklarujeme ji ve funkci a platí jen v dané funkci

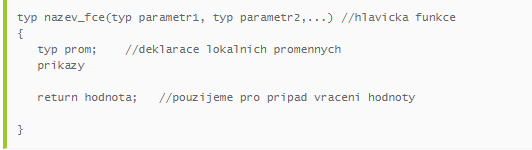
Globální proměnná – deklarujeme ji mimo funkci a platí od místa deklarace v celém programu

Deklarace funkce – před funkcí main

* Obsahuje název funkce, typ návratové hodnoty a typy parametrů

Definice funkce – za funkcí main

* Obsahuje kompletní zápis funkce hlavička + tělo



Parametr funkce může být jakákoliv proměnná se kterou chceme pracovat ve funkci. Parametrem funkce může být i pole. To se zapisuje do parametru takto: ***int pole[]*** Zapisuje se bez velikosti. Návratová hodnota, ale nemůže být pole. Návratová hodnota je to co chceme vrátit jako výsledek do hlavní funkce. Bez návratové hodnoty se jedná o funkci typu void. Funkce nemusí mít žádné parametry.

Volání funkce se provádí v hlavní funkci, v jiné funkci nebo funkce může volat sama sebe. Volání vypadá následovně***: názevfunkce();*** v závorce se udávaj parametry. Pokud má nějakou návratovou hodnotu tak ji musíme uložit do nějaké proměnné stejného typu ***int a = nazevfunkce();***

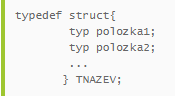
**Struktury a ukazatele**

Struktura

Obsahuje hodnoty různých datových typů. Definujeme ji jako uživatelský datový typ.

K jednotlivých položkám struktury přistupujeme přes tečku.

Uvádí se nad funkcí main.



Deklarace:



Typedef struct{

int cislo;

}NUMBER;

Int main(){

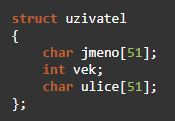
NUMBER a;

a.cislo = 5;

}

Když definujeme strukturu tímto způsobem tak musí být název struktury vždy VELKÝMI PÍSMENY.

Další způsob definice struktury je



U tohoto způsoby se struktura pojmenovává malými písmeny.

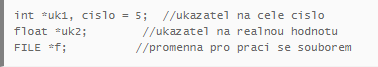
Ukazatel

Ukazatelé se používají pro práci s polem, jako parametry funkce, při práci s dynamickou alokací paměti atd.

*Ukazatel je proměnná jako ostatní, pouze hodnota v ní uložená má jiný význam, než jsme byli zvyklí dosud. V jazyce C je ukazatel přísně typový, t.j. deklarujeme ho jako ukazatel na určitý datový typ. Obsahem ukazatele je adresa v paměti. Při práci s ukazatelem nás ve většině případů zajímá hodnota, uložená na dané adrese.*

Jednoduše řečeno, Ukazatel je proměnná, která ukazuje na místo v adrese, kde se daná proměnná nachází.

Deklarace ukazatele:

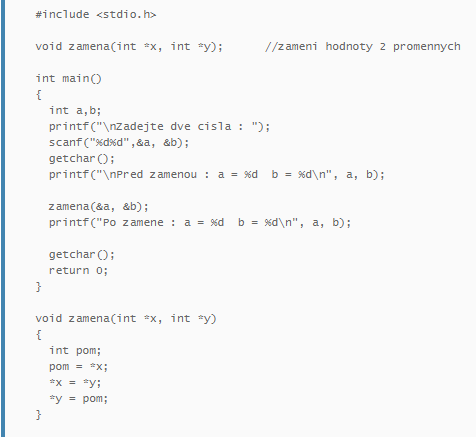


Pro nastavení ukazatele se používá &



Teď bude ukazatel uk. Ukazovat na adresu proměnné číslo. Pokud bychom chtěli pracovat s hodnotou na kterou ukazatel ukazuje (čili 5) a ne s jeho adresou použijeme v program \*uk

Při použití ukazatele jako parametru funkce. Může funkce změnit hodnotu parametru a zároveň ji takto změnit ve funkci main bez toho aby použila návratovou hodnotu.



Díky toho můžeme ve funkci měnit i pole a „vrátit“ ho do mainu. Jednoduše uděláme parametr ukazatel do pole a potom s ním pracujeme ve funkci.

**Práce se soubory**

V C máme 2 typy souborů: textový a binární.

Pro práci s polem potřebujeme knihovnu stdio.h a speciální proměnnou pro pole ***FILE \*nazev***;

Název si můžeme zvolit libovolný.

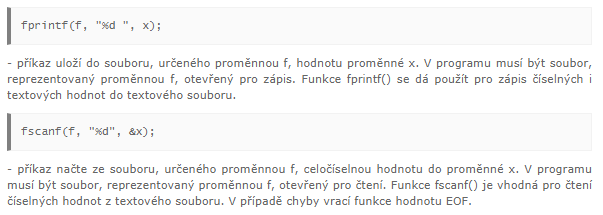
Pro otevření pole se používá funkce fopen()

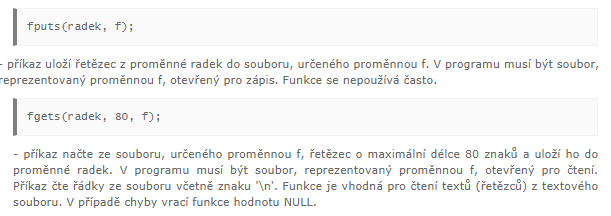
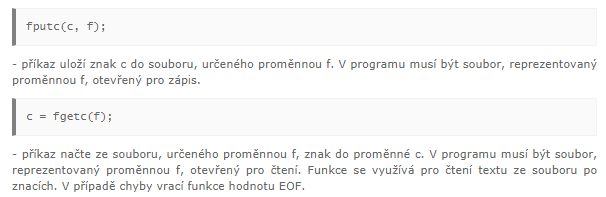
Syntaxe : nazev = fopen(„cesta k souboru“,“režim“)

Cesta k souboru se nemusí udávat pokud se soubor nachází na stejné adrese jako program.

Režimy: r- čtení w- zápis a – přidání na konec

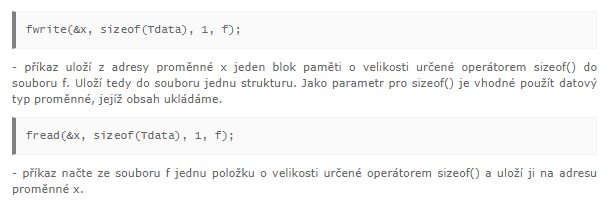
Pro zavření se používá fclose(nazev);





Binární soubor

Pří binárním souboru se používají soubory s příponou .dat nebo .bin. Režimy jsou stejné pouze se přidává b (wb, rb, ab).



**Knihovny**

Knihovny jsou buď dané v C dané nebo si je můžeme nainstalovat. Knihovny obsahují předdefinované proměnné, funkce, jiné knihovny.

String.h, time.h, stdio.h, stdlib.h, ctype.h, windows.h

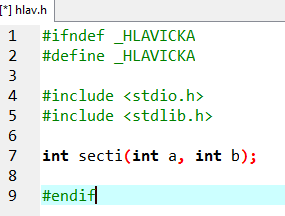
Tvorba vlastní knihovny:

Knihovna musí mít příponu .h. Definice funkcí se většinou vytváří v jiném souboru než deklarace. Deklarace se vytváří přímo v knihovně. Definice mimo ní v jiném souboru .c.

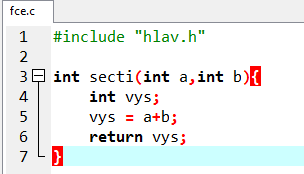
Když chceme includovat naši knihovnu nepoužíváme klasicky špičaté závorky ale uvozovky.

***#include „mojeknihovna.h“***

Jak vypadá soubor .h :



Knihovna musí obsahovat ošetření pokud není definována \_HLAVICKA definuj \_HLAVICKU. Ukončeno endif. Když používáme vlastní knihovnu includujeme všechny potřebné knihovny v naší knihovně, protože s těmito knihovnami funkce naší knihovny většinou pracují.



Takto vypadá program, který definuje funkce naší knihovny.

**Komentáře**

// komentování jednoho řádku

/\* komentování více řádků | komentování více řádků | komentování více řádků | komentování více řádků | komentování více řádků | \*/