Zdroj: http://www.sallyx.org/sally/c/index.php

Struktura programu

- První program
- Komentáře
- Knihovny
- Funkce
- Funkce printf()

Vše o vzniku programu již bylo objasněno a tak se můžeme věnovat čistě jen syntaxi jazyka C. V této kapitole si na prvním jednoduchém příkladě osvětlíme velkou spoustu nových věcí. Některé z nich se budou probírat později podrobněji. Pokud pochopíte věci vysvětlované v této kapitole, již se můžete začít považovat za programátory amatéry :-).

První program

Správně bych měl říkat první zdrojový kód, ale myslím, že říkat "program" není zase tak velký prohřešek. Tak mě za to nekamenujte. Však vy si z toho zdrojového kódu program vytvoříte.
Podívejme se tedy na něj. Následující kód můžete uložit do souboru s příponou **.c** (např. kod.c). Přípona .c je pro některé překladače (zvláště v Linuxu) povinná, někdy však ne a soubor se může jmenovat libovolně. Nevidím však jediný rozumný důvod, proč jí nepoužít.
Ve jménech souboru nepoužívejte českou diakritiku ani mezery!

```
 1: /*------------------------------------------------*/
 2: /* kod.c                                          */

 4:  /* Toto je libovolny komentar
 5:   * v souboru kod.c           */

 7: #include <stdio.h>                //standardni knihovna

 9: int main(void)
10: {
11:     printf("Hello World\n");
12:     return 0;
13: }

15: /*------------------------------------------------*/
```

Čísla na začátku řádků nejsou součástí zdrojového kódu. Budu je však uvádět před každým zdrojovým kódem, abych se na ně mohl v textu odkazovat.

Všimněte si mezer, odsazování a konců řádků ve zdrojovém kódu a vůbec celkové úpravy. Bílé znaky, jako je mezera, tabulátor nebo znak nového řádku většinou ve zdrojovém kódu jazyka C nemají žádný význam. Odsazováním program pouze získá na přehlednosti a čitelnosti. Funkci main() by jste mohli napsat do jednoho řádku, to je překladači jedno.

```
int main(void) { printf("Hello World\n"); return 0;}
```

Posuďte sami, co je přehlednější. Konce řádků jsou důležité pouze pro direktivy začínající znakem # (čti „šarp“) a u komentářů začínajících dvěma lomítky, protože jejich konec je dán právě koncem řádku.

Komentáře

Tím nejjednodušším, co je v programu vidět, jsou **komentáře**. Vše co je mezi znaky /* a */ je komentář. V jazyce C++ se navíc považuje za komentář vše za dvěma lomítky // až do konce řádku. Většina moderních překladačů C také rozezná dvě lomítka jako komentář (mnohdy je překladač C a C++ jeden a ten samý program). Až budete dělat větší programy, uvidíte jak je dobré mít správně okomentovaný zdrojový kód. **Dobré komentáře vám pomohou se v kódu orientovat**. Je dobré poznamenávat takové věci, jako třeba ... *toto číslo musí být nezáporné z toho a toho důvodu* ... . Při změně programu v budoucnosti se tím vyvarujete chyb.

Knihovny

Existují takzvané standardní knihovny, což jsou knihovny dodávané s překladačem, a uživatelské knihovny, které si vytváří programátor sám. **Hlavičkové soubory** jsou soubory, v nichž jsou uloženy definice funkcí z knihovny.
Jejich přípona je většinou **.h**, ale není povinná (stejně jako .c). Pro C++ se někdy knihovny označují příponou .hpp.

Direktiva #include říká překladači, jaký hlavičkový soubor má načíst před překladem zdrojového kódu. Pokud je jméno v takovýchto špičatých závorkách < >, pak se soubor hledá ve standardních adresářích. Které to jsou, to záleží na překladači.

Standardní knihovna **stdio** (která obsahuje funkci printf()) je tak notoricky používána, že mnohé překladače ani nevyžadují uvední direktivy #include <stdio.h>. Přesto, v zájmu kompatibility, tuto direktivu používejte (pokud jí potřebujete).
stdio = **st**an**d**ard **i**nput **o**utput (knihovna pro standardní vstup a výstup)

Jména uživatelských hlavičkových souborů jsou uzavřeny ve dvojitých uvozovkách a hledají se v aktuálním adresáři. Například #include "kkk/knihovna.h" je soubor se jménem knihovna.h v podadresáři kkk/ aktuálního adresáře. Jak si vytvořit vlastní hlavičkový soubor se naučíme později. Nejdříve se budeme seznamovat se standardními knihovnami a funkcemi.

Funkce

V našem příkladě máme v programu dvě funkce. Funkci main() a funkci printf(). Funkci main() definujeme. To znamená, že popisujeme co tato funkce bude dělat. Funkci printf() pouze „voláme“, tj. chceme po ní, aby udělala to, co je dáno její definicí. Je to funkce definovaná v knihovně stdio.h a její definice způsobý vypsání textu.

Funkce má své jméno (main), návratovou hodnotu (int) a argumenty určené v závorkách za jménem funkce ((void)). Návratová hodnota určuje, jaká data funkce vrátí při svém skončení (např. int je celočíselná hodnota). K čemu to je, to se dozvíte později. Argumenty jsou pro změnu data, která funkce dostává ke zpracování. Hodnota void znamená „nic“, v našem případě to znamená, že funkce main() žádné argumenty neočekává. Návratová hodnota je jen jedna a její typ se píše před jméno funkce. Argumenty se píší za jméno funkce do závorek a

je-li jích více, oddělují se čárkou. Funkce printf() dostává jako argument řetězec (řetězce jsou vždy ve dvojitých uvozovkách). Znaky \n reprezentují nový řádek (ENTER, chcete-li).

Tělo funkce je všechno to, co je ve špičatých závorkách { }. Funkce se ukončuje klíčovým slovem return za kterým je návratová hodnota funkce.

Funkce jsou poměrně složitou záležitostí a budeme se jim věnovat později podrobněji. Důležité je vědět, že **funkce main() je speciální funkce, která je volána[1] v programu jako první**. Ve funkci main() pak můžete volat další funkce, dle libosti. Z toho také vyplývá, že funkci main musí mít každý program, jinak by překladač nevěděl čím začít při provádění programu. Funkce main() má vždy návratovou hodnotu typu int (celé číslo).

Funkce printf()

Z toho přísunu nových informací můžete být dost zmateni. Hlavně proto, že jsem u spousty věcí říkal, že je probereme podrobněji později). Pokud jste se prokousali až sem, pak již máte vyhráno. Už víte jak udělat program v jazyce C. Nejdříve začnete funkcí main(), protože ta se volá v programu jako první. Přidáte knihovny pomocí #include, aby jste mohli používat v nich nadefinované funkce a ty pak voláte ve funkci main(). Funkce main() skončí příkazem return. Pokud by jste jej neuvedli, pravděpodobně by jste program také přeložili, ale překladač by vás na tuto chybu upozornil varováním. Stejně tak by bylo odpustitelnou chybou, kdyby jste za jménem funkce main() do závorek nenapsali void ale nechali je prázdné. Překladač by si „void“ domyslel sám. Funkce main() může být buďto bez argumentů, nebo se dvěma speciálními argumenty, o kterých si ale povíme později :-).

Na závěr si ukážeme ještě jeden program a další využití funkce printf(). Funkce printf() má jako první argument textový řetězec. Ten může obsahovat speciální sekvence. Už jsme se s jednou setkali. Sekvence \n přesune kursor na nový řádek. Funkci printf(). si později vysvětlíme systematičtěji i se všemi speciálními znaky. Teď vám chci ukázat ještě dva, které se nám budou při výkladu hodit. První je %s. Za tento znak se dosadí textový řetězec, který je dalším argumentem funkce printf(). Druhý je %i, za který se dosadí celé číslo. Prostudujte si následující program a zkuste napsat vlastní.

```
 1: /*------------------------------------------------*/
 2: /* kod2.c                                         */

 4: #include <stdio.h>

 6: int main(void)
 7: {
 8:     printf("1 + 1 = %i\n", 1 + 1);
 9:     printf("%i + %i = %i\n", 1, -2, 1 + (-2));
10:     printf("%s\n", "Konec programu.");
11:     /* return ukonci program. */
12:     return 0;
13:     printf("Tohle se uz nezobrazi %s!\n");
14: }

16: /*------------------------------------------------*/
```

Všimněte si posledního volání funkce printf(). Je až za příkazem return a proto k jejímu volání v programu ani nedojde. Navíc obsahuje chybu. V řetězci je %s ale funkce nemá žádný další argument. Pokud by tato funkce byla volána, „sáhla“ by si kamsi do paměti pro

neexistující řetězec. Buď by se vytiskly nějaké nesmysly, nebo by program skončil s chybou (neoprávněný přístup do paměti; program provedl nepovolenou operaci atp.). Takže si na to dávejte pozor. Na tuto chybu vás překladač nemusí upozornit, protože překladač zkontroluje jen to, že funkce printf() má mít jako první argument textový řetězec (a to má). Jelikož je tato chyba opravdu častá (a taky dost nebezpečná), dokáží moderní překladače tyto chyby odhalit, ale mnohdy se o to musí „požádat“ nějakým nastavením.

A to je pro dnešek vše. Přečtěte si tuto kapitolu tolikrát, kolikrát bude potřeba, aby jste zdrojový kód kod2.c celý pochopili. Zvykněte si taky na pojmy „volání funkce“, „definice funkce“, „argumenty funkce“ a „návratová hodnota funkce“. Určete všechny části zdrojového kódu, které se k těmto pojmům vztahují.

[1)] „Voláním funkce“ se myslí provedení příkazů, které funkce obsahuje.

Datové typy, bloky, deklarace a definice

- Základní datové typy
- Boolovské datové typy
- Bloky, deklarace a definice
- Ukazatele
- NULL

Základní datové typy

Základní datové typy jazyka C

char	8	znak	'a', 'A', 'ě'
short[1)]	16	krátké celé číslo	65535, -32767
int	16/32	celé číslo	--||--
long[1)]	32	dlouhé celé číslo	--||--
long long[1)]	64	ještě delší celé číslo	9223372036854775807, -9223372036854775807
enum	8/16/32	výčtový typ	
float	32	racionální číslo	5.0, -5.0
double	64	racionální číslo s dvojitou přesností	5.0l, 5l, -5.0L
long double	80	velmi dlouhé racionální číslo	5.5L, 5l, -5.0l
pointer	16/32	ukazatel	

V prvním sloupci vidíte základní datové typy jazyka C. Ve druhém vidíte to, kolik zabírají (orientačně) bitů v paměti. Čím více bitů, tím větší číslo je možné zapsat. Na druhou stranu zabírají více paměti a práce s nimi je pomalejší! Ve třetím sloupci jsou významy čísel a ve čtvrtém příklady.

Co je na jazyku C zajímavé (a za co ho někteří nemají rádi) je to, že velikost všech datových typů není standardem pevně dána. Můžete se spolehnout je na to, že velikost char = 8 bitů (= 1 byte) a že:
sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long).
Proč? Závisí to především na procesoru. Na 16-bitovém procesoru budete mít nejspíše int velikosti 16 bitů (16bitovému procesoru se s takto velkým číslem "nejlépe" pracuje).

Při výběru vhodného datového typu by jste měli počítat s tím, že rozsah datového typu (tj nejmenší a největší číslo, které do něj můžete nacpat) se bude lišit „počítač od počítače“. Pokud budete mít při návrhu pocit, že by mohlo dojít k „přetečení“ rozsahu (takhle se říká tomu, když se snažíte do nějaké proměnné uložit větší čislo, než se tam vejde), používejte pro ověření velikosti čísla konstanty z knihovny <limits.h>.

Konkrétní rozsahy datových typů můžete najít ve standardní knihovně <limits.h>.

Existuje ještě další způsob jak zjistit velikost datového typu. Například je zde operátor sizeof(), která vrací počet bytů (nikoliv bitů) datového typu (nebo proměnné). Například sizeof(char) by se vždy mělo rovnat 1.
Další možnost si ukážeme o několik kapitol dále, v příkladu s binárními operátory (to teď ale pusťte klidně z hlavy).

Typy char, short, int, long a long long mohou být se znaménkem nebo bez znaménka. Určí se to pomocí klíčových slov **signed** (se znaménkem) nebo **unsigned** (beze znaménka). „signed“ je defaultní hodnota, proto jí není třeba psát. Vzpomeňte si na kapitolu o bitech a bajtech.

Na následujícím příkladě si osvětlíme použití datových typů.

```
 1: /*-----------------------------------------------*/
 2: /* typy.c                                        */

 4: #include <stdio.h>
 5: int a;

 7: int main(void)
 8: {
 9:     /* deklarace promennych */
10:     unsigned int b;
11:     float c;

13:     a = 5;
14:     b = 11;
15:     a = a / 2;
16:     c = 5.0;
17:     printf("a=%i\nb=%u\nc=%f\n", a, b, c / 2);

19:     return 0;
20: }

22: /*-----------------------------------------------*/
```

Výstup z tohoto programu bude následující:

```
a=2
b=11
c=2.500000
```

Všimněte si, že do proměnné a lze uložit pouze celé číslo. Vyzkoušejte si do proměnné b uložit záporné číslo. Uhodnete, co se stane?

Boolovské datové typy

V některých programovacích jazycích se používá datový typ **bool**, který může nabývat pouze hodnot **true** (pravda) nebo **false** (nepravda). Tyto hodnoty se používají ve vyhodnocování podmínek při řízení běhu programu a podmíněných operátorů. Oboje budeme probírat později. Na tomto místě si jen řekneme, že v jazyce C takovýto typ neexistuje.

V jazyce C se jako **nepravda** vrací 0 a jako **pravda** 1. A dále platí, že cokoliv nenulového se vyhodnocuje jako pravda (tj. vše kromě 0 a NULL).

Nic není straceno. V jazyce C si můžete vytvářet vlastní datové typy pomocí operátoru typedef a hodnoty *true* a *false* by jste si mohli nadefinovat třeba pomocí preprocesoru. Ale na to zapomeňte :-). Standard C99 (stejně jako C++) totiž přináší knihovnu <stdbool.h>, která to už za vás udělala.

Takže odvolávám co jsem odvolal :-). Datový typ **bool** můžete používat, pokud zahrnete do svého zdrojáku:

```
#include <stdbool.h>
bool x = true;
x = false;
```

Příklad vám neukážu, protože to nemá smysl, dokud se nedostaneme k vyhodnocování podmínek.

Bloky, deklarace a definice

Blokem se rozumí vše, co je mezi špičatými závorkami { a }. V příkladě nahoře máme jediný blok, a to tělo funkce main(). Všechny proměnné, které deklarujeme v bloku, jsou tzv. lokální proměnné a proměnné mimo blok jsou globální. Lokální proměnné platí jen v těle bloku a v blocích vnořených (jinde je překladač neuvidí).

Proměnné jsou místa kdesi v paměti počítače, které definujeme v programu. Velikost místa v paměti závisí na datovém typu proměnné. Do proměnných lze vkládat hodnoty, měnit je a číst. V příkladě typy.c jsme vytvořili globální proměnnou a a dvě lokální proměnné b a c.

Deklarací proměnných určujeme jaké proměnné se v programu objeví. Nejdříve se určí datový typ a za ním názvy proměnných oddělené čárkou. Deklarace se ukončí středníkem. Při deklaraci je možné do proměnné rovnou vložit hodnotu. Názvy proměnných nesmí obsahovat mezeru, národní znaky (č, ř, š atp.) a nesmí začínat číslem. Jejich maximální délka je závislá na překladači. Jazyk C rozlišuje malá a velká písmena, takže můžete vytvořit proměnné se jmény Ahoj, ahoj, aHoj ... a program je bude chápat jako různé identifikátory. Dobrým zvykem je pro proměnné používat vždy jen malá písmena.

```
unsigned int i,j,k;
signed int cislo1, cislo2 = cislo3 = 25;
```

Dokud neurčíte jakou má proměnná hodnotu, může v ní být jakékoliv číslo. Např. proměnné cislo2 a cislo3 mají hodnotu 25, všechny ostatní mají náhodnou hodnotu!

Proměnné mohou být v jazyce C deklarovány pouze na začátku bloku funkce (tzv. lokální proměnné), nebo na začátku programu (globální proměnné). Lokální proměnné „existují“ jen v bloku, ve kterém jsou definovány (případně v blocích v něm vnořených) a nelze je tudíž používat mimo něj. V jazyce C++ je možné deklarovat proměnné téměř kdekoliv, ale stále platí, že proměnná má platnost pouze v bloku, ve kterém je deklarována.

Definice říká, jak bude nějaký objekt vypadat. V programu například definujeme funkci main(). Základní datové typy jsou definovány normou ANSI C. Můžeme definovat i vlastní typy dat, další funkce atd. K tomu se dostaneme později. Jen si nepleťte pojmy definice a deklarace. Definovat objekt můžeme jen jednou, deklarovat vícekrát.

Ukazatele

Ukazatele (též zvané pointry) jsou jsou jednou z nejtěžších věcí na pochopení v jazyku C. Ukazatele jsou proměnné, které **uchovávají adresu** ukazující do paměti počítače. Při jejich deklaraci je třeba uvést, na jaký datový typ bude ukazatel ukazovat. Paměť počítače je adresována po bytech. Číslo 0 ukazuje na první bajt v paměti, číslo 1 na druhý bajt atd. Pokud máte šestnáctibitový překladač, bude velikost ukazatele 16 bitů, u 32 bitového překladače 32 bitů. Deklarace ukazatelů vypadá takto:

```
typ * jmeno;
```

Například:

```
float *uf; // Ukazatel na float, ma velikost typu "ukazatel", nikoliv typu
float!
int *ui;
void *uv;
```

Proměnné uf, ui a uv jsou ukazatele.
Proměnná uf je ukazatel na typ float, ui je ukazatel na typ int a uv je tzv. prázdný ukazatel u kterého není určeno, na jaký datový typ se bude ukazovat. To překladači neumožňuje typovou kontrolu, proto je lepší jej nepoužívat, pokud to není nutné. Do všech třech proměnných lze vložit libovolné (celé) číslo, které pak bude překladačem chápáno jako adresa do paměti. K získání adresy nějaké proměnné slouží operátor & (viz příklad). K získání dat z adresy, která je uložena v ukazateli, slouží operátor * (viz příklad).

```
 1: /*-----------------------------------------------*/
 2: /* ukazatel.c                                    */

 4: #include <stdio.h>

 6: int main(void)
 7: {
 8:     int i;        /* promenne int */
 9:     float f, f2;
10:     int *ui;        /* ukazatel (na typ int) */
11:     float *uf;

13:     f = 5.5;
```

```
 14:     uf = 50;        /* v ukazateli uf je hodnota 50. Co je v pameti na
50 bajtu
 15:                     nikdo nemuze vedet, proto se takto do ukazatele
adresa
 16:                     nikdy (!) neprirazuje */
 17:     uf = &f;        /* v ukazateli uf je hodnota adresy promenne f
(tak je to
 18:                     spravne :-)) */
 19:     ui = &i;

 21:     f2 = *uf;        /* do f2 se priradi hodnota z promenne f, tedy
5.5 */
 22:     *ui = 10;        /* tam, kam ukazuje adresa v ui (tedy do i) se
ulozi 10.
 23:                     Hodnota promenne ui se nezmeni (je to stale adresa
 24:                     promenne i) */

 26:     printf("f = %f\n", f); /* hodnota v f se od prirazeni 5.5
nezmenila */
 27:     printf("f2 = %f\n", f2); /* do f2 se priradila hodnota z promenne
f, na
 28:                                 kterou jsme se odkazovali pomoci
ukazatele uf */
 29:     printf("i = %i\n", i); /* do i jsme pres ukazatel ui priradili 10
*/

 31:     printf("uf = %u, *uf=%f\n", uf, *uf); /* vytiskneme si hodnotu uf
a hodnotu
 32:                                           na teto adrese (tj.
adresu f2 a
 33:                                           hodnotu f2) */
 34:     return 0;
 35: }
 36: /*-----------------------------------------------*/
```

Výstup z programu bude následující:

```
f = 5.500000
f2 = 5.500000
i = 10
uf = 3221224276, *uf=5.500000
```

Proměnná uf obsahuje adresu paměti, na které byla proměnná f uložena. Ta se bude při každém spuštění programu lišit, neboť paměť přiřazuje programu operační systém. Do které části paměti program nahraje nelze předem vědět (alespoň mi, obyčejní smrtelníci, to nevíme).

V programu jsem se dopustil dvou chyb. Jednak to bylo přímé přiřazení hodnoty do proměnné uf (uf = 50;) a pak v poslední funkci printf, kde se tiskne hodnota ukazatele uf jako celé číslo bez znaménka (unsigned int). Je možné, že vám tyto konstrukce překladač ani nedovolí přeložit!

Tisknutím ukazatele (pointeru) jako unsigned int jsem chtěl zdůraznit, že je to číslo (které uchovává nějakou adresu paměti). Ukazatele (pointery) by se správně měli tisknout přes %p:

```
 31:     printf("uf = %p, *uf=%f\n", uf, *uf); /* vytiskneme si hodnotu uf
...
```

Za zmínku ještě stojí přiřazení f2 = *uf;. Jak to funguje? Nejdříve se vezme hodnota z uf a z této adresy se přečte číslo dlouhé tolik bitů, kolik bitů má typ, pro který je ukazatel určen. Proměnnou uf jsme deklarovali jako ukazatel na typ float, který má 32 bitů. Proto se vezme 32 bitů začínajících na adrese uložené v uf a toto číslo se pak chápe jako racionální a jako takové se uloží do f2. Kdybychom deklarovali uf jako ukazatel na typ char (char *uf;), pak by jsme se podívali sice na stejnou adresu, ale z ní přečetli jenom 8 bitů!

Asi vám v tuto chvíli není moc jasné, k čemu jsou ukazatele dobré. K tomu se však velice rychle dostaneme. Zatím je třeba, aby jste pochopili, k čemu slouží operátor & a * a také pochopili, že ukazatel, ať už ukazuje na jakýkoliv typ, je stále jen ukazatel.
Z toho také vyplývá, že velikost ukazatele je stejná, ať už ukazujeme na char, nebo long double (sizeof(char*) == sizeof(long double*).

NULL

NULL je speciální znak, který označuje velké nic. Používá se například v souvislosti s ukazateli. NULL můžete vložit do ukazatele a pak můžete testovat, zda ukazatel obsahuje NULL nebo adresu někam. NULL se také často používá jako argument funkcí, když tam, kde je argument vyžadován žádný argument vložit nechceme. Praktické využití této hodnoty si ukážeme později, například v ukázce použití NULL.

NULL je definováno v knihovně <stddef.h>, která je načítána např. pomocí <stdlib.h> a <stdio.h>.

[1)] long je ve skutečnosti zkratka pro *long int*, short pro *short int* a long long pro *long long int*.