Přehled jazyka C (ISO C99)
David Martinek © 2004–2007

Základní stavební bloky programu

Komentář	`/* text */`
Řádkový komentář	`// text`
Deklarace a definice proměnné	`typ jmeno;`
+ inicializace	`typ jmeno = hodnota;` `typ jmeno = {složky};`
Deklarace funkce (prototyp)	`typ jmeno(typ1 par1, ...);`

Definice funkce (příkaz `return` je možné použít kdekoli)	`typ jmeno(typ1 par1, ...)`
	`{`
	`  kód`
	`  return výsledek;`
	`}`

Volání funkce	`proměnná = jméno(par1, par2);`
+ bez parametrů	`proměnná = jméno();`
+ bez návratové hodnoty	`jméno();`

Pořadí vyhodnocování parametrů funkce není definováno!

Hlavní program	`int main(void)`
	`{`
	`  kód`
	`  return errCode;`
	`}`

+ s parametry	`int main(int argc, char *argv[])`
	`{`
	`  kód`
	`  return errCode;`
	`}`

`errCode == 0` ⇒ program končí bez chyby.

Preprocesor

Dovoz rozhraní standardních knihoven	`#include <soubor.h>`
Dovoz rozhraní lokální uživatelské knihovny	`#include "soubor.h"`
Definice konstanty (zastaralé, raději používejte const, viz dále)	`#define PI 3.1415926535`

Datové typy

znaménkový modifikátor (+ i -) jen pro celočíselné typy	`signed typ`
bezznaménkový modifikátor (jen +) jen pro celočíselné typy	`unsigned typ`

Pro všechny celočíselné typy je implicitní modifikátor `signed`. Znaménkové modifikátory nemění velikost typu.

znak (1Byte)	`char`
celé číslo	`int`
malé celé číslo	`short`
velké celé číslo	`long`
velmi velké celé číslo	`long long`

reálné číslo (cca 7 míst)	`float`
– dvojitá přesnost (cca 15 míst)	`double`
– s velkou přesností (cca 20 míst)	`long double`
logická hodnota (viz <stdbool.h>)	`bool` hodnoty: `true, false`
prázdný typ (bez hodnoty)	`void`
obecný ukazatel	`typ *`
ukazatel na číslo	`int *, float *, ...`
pole	`typ jmeno[ROZMER];`
textový řetězec (končí nulou '\0')	`char jmeno[N]` (kompatibilní s `(char *)`)

Pozor! Mezi oběma typy jsou rozdíly (viz literatura, přednášky).

struktura	`struct jmeno {...};`
výčet	`enum jmeno {...};`
přejmenování typu (zkratka, nevytváří nový typ)	`typedef typ noveJmeno;`
zjištění velikosti v B (výsledek je typu `size_t`)	`sizeof(výraz)`
	`sizeof(typ)`
velikost pole	`sizeof(int [10]) == 10*sizeof(int)`
přetypování výrazu	`(typ)výraz`

Pozor! Ne všechny typy jsou vzájemně převeditelné (kompatibilní).

Pro číselné typy platí:

- `sizeof(char) = 1 Byte`
- `sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`
- `sizeof(unsigned int) = sizeof(signed int)`
- `sizeof(float) <= sizeof(double) <= sizeof(long double)`

Konkrétní velikost číselných datových typů je dána architekturou počítače. Velikost typu `int` je většinou shodná s velikostí registrů daného procesoru. Aby byly programy přenositelné, není dobré se spoléhat na pevnou velikost číselných typů. Velikost typu je vždy nejlepší zjišťovat pomocí operátoru `sizeof`.

Konstanty, literály

Definice „konstanty“	const typ jmeno = hodnota;

Přesněji je to nemodifikovatelná konstantní proměnná.

Celé číslo v soustavě:	
– desítkové	`12345`
– osmičkové	`012345` (prefix 0 – nula)
– hexadecimální	`0x123AB` (prefix 0x nebo 0X)
Číslo typu `long`	`1234L` (přípona L)
Číslo typu `double`	`123.0`
Číslo v exp. tvaru	`1.23e12`
znak	`'a'`
znak (osmičkově, hexa)	`'\101','\x41'`
nový řádek, cr, tabulátor	`'\n' '\r' '\t'`
speciální znaky	`'\\' '\?' '\'' '\"'`
řetězec (končí nulou '\0')	`char jm[] = "hubero";` `// sizeof(jm) == 7`
řetězec (znaky na indexu 6–9 jsou '\0')	`char jm[10] = "kororo";` `// sizeof(jm) == 10`
POZOR! strlen(jm) == 7, ale sizeof(jm) == 4	`char *jm = "ororok";` `// sizeof(jm) != 7`

Ukazatele

ukazatel na jednoduchý typ	`int * pCislo;`
Nulová hodnota ukazatele	`NULL`
Obecný ukazatel, kompatibilní se všemi ukazateli	`void *`
alokace paměti	`pCislo = malloc(sizeof(int));`
uvolnění paměti	`free(pCislo);`
dereference – přístup k hodnotě odkazovaného objektu	`*pCislo = 10;` `hodnota = *pCislo;`
reference – získání adresy	`pCislo = &promenna;`

Pole

jednorozměrné pole	`typ jmeno[N];`
vícerozměrné pole	`typ jmeno[K][L][M]...;`
statická alokace (implicitní hodnoty prvků nedefinovány)	`int pole[ROZMER];` `int matice[RADKU][SLOUPCU];`
+ inicializace	`int pole[] = {1, 2, 3};`
++ všech prvků nulou	`int pole[ROZMER] = {0};`
→ {1, 0, 0, 5, 0}	`int pole[5] = {1, [3]=5};`

Nevyjmenované prvky inicializovaného pole mají hodnotu nula.

dynamická alokace	`int*p=malloc(N*sizeof(int));`

Implicitní hodnoty nejsou definovány, nutno nastavit „ručně“.

indexace	`pole[0] = 27;`

POZOR! Jazyk C nekontroluje meze polí. Překladač v takovém případě nevypisuje ani varovné hlášení!

Struktury, výčty

definice struktury	`typedef struct coord {`
	`  int x, y, z;`
	`} TCoord;`

definice proměnné	`TCoord vrchol;`
inicializace proměnné	`TCoord vrchol = {1, 4, 9};`
přístup ke složkám	`vrchol.x = 2;`

přístup přes ukazatel	`TCoord *pVrchol=...;`
	`pVrchol->x = 10.1; nebo`
	`(*pVrchol).x = 10.1;`

výčet (pro shlukování konstant)	`enum jablka {GOLDEN, JONATAN=10, CERVIVE};`

Každá konstanta výčtu je typu `int`.

Operátory (seřazeno podle priority)

pr.	význam	operátory	as.
1.	volání funkce	`()`	→
	indexace pole	`[]`	
	přístup ke složkám struktury	`jmeno.člen` `ukazatel->člen`	
2.	inkrementace, dekrementace	`++, --`	←
	unární +, -	`+, -`	
	logický, bitový not	`!, ~`	
	reference, dereference	`&jmeno, *ukazatel`	
	přetypování	`(typ)proměnná`	
	velikost objektu	`sizeof`	
3.	násobení, dělení, modulo	`*, /, %`	→
4.	sčítání, odčítání	`+, -`	→
5.	bitový posuv vlevo, vpravo	`<<, >>`	→
6.	porovnávání	`>, >=, <=, <`	→
7.	porovnávání	`==, !=`	→
8.	bitový and	`&`	→
9.	bitový xor	`^`	→
10.	bitový or	`|`	→
11.	logický and	`&&`	→
12.	logický or	`||`	→
13.	podmíněný výraz	`podmínka ? výraz1 : výraz2`	→
14.	přiřazení	`=, +=, -=, *=, /=, %=, >>=, <<=, &=, |=, ^=`	←
15.	čárka	`,`	→

Naznačená asociativita je pouze orientační. Překladač může změnit pořadí vyhodnocení operandů, pokud dojde k závěru, že to nebude mít vliv na výsledek výrazu. V tom případě ale nebere ohled na případné vedlejší efekty, pokud se ve výrazu vyskytuje volání funkce.

Závorky `()` mění přirozené pořadí vyhodnocování operátorů. Je dobré je používat pro zvýšení čitelnosti výrazů. Pokud si nejste jisti prioritami, závorkujte.

Řídící struktury

ukončovač příkazů	`;`
blok – složený příkaz	{}

Za blokovým příkazem se středník nepíše. Středník se píše pouze za závorkou }, která je součástí deklarace datové struktury (struct, enum) nebo inicializátoru.

násilný konec příkazů `switch, while, do, for`	`break;`
skok na další iteraci `while, do, for`	`continue;`
ukončení a návrat hodnoty funkce	`retun výraz;`

příkaz `if`	`if (podmínka)`
	`{příkazy}`
	`else if (podmínka)`
	`příkaz;`
	`else`
	`{příkazy}`

Pokud je před `else` blokový příkaz, středník se nepíše. Pokud je tam samostatný příkaz, musí být ukončen středníkem.

cyklus `while`	`while (podmínka)` `{příkazy}`
cyklus for	`for(int i = 0; i < N; i++)` `{příkazy}`

cyklus `do-while`	`do {`
	`  příkazy`
	`} while (podmínka);`

příkaz `switch`	`switch (výraz)`
	`{`
	`  case konst1: příkazy break;`
	`  case konst2: příkazy break;`
	`  default: příkazy`
	`}`

V příkazu `case` nejde použít konstanty definované pomocí `const`.

Přehled jazyka C (ISO C99), Autor: David Martinek ©2004–2007. Tento dokument neobsahuje úplný popis jazyka C. Je to pouze přehled základních konstrukcí, které jsou potřeba pro tvorbu projektů pro kurz Základy programování na FIT VUT v Brně. Připomínky posílejte na martinek@fit.vutbr.cz nebo na adresu: David Martinek, Fakulta Informačních Technologií, VUT v Brně, Božetěchova 2, 612 22, Brno.