Rýchlokurz jazyka C

RNDr. Jaroslav Janáček

Úvod

Cieľom tohto materiálu je poskytnúť študentom informatiky, ktorí majú absolvovaný základný kurz programovania na FMFI UK, základy jazyka C potrebné na vypracovanie projektu zo systémového programovania. Cieľom nie je poskytnúť úplný (skôr veľmi obmedzený) prehľad jazyka – materiál celkom ignoruje mnohé dôležité časti jazyka C (ako napr. floating point typy, dynamicky alokovanú pamäť, pointer-ovú aritmetiku) a celkom ignoruje štandardné knižnice (až na zopár vybraných funkcií). Cieľom tiež nie je vysvetliť základy programovania – predpokladá sa, že čitateľ vie programovať v inom štruktúrovanom jazyku, najlepšie v Pascale.

Základná štruktúra programového modulu

Jeden programový modul (jeden zdrojový súbor s príponou .c) sa skladá z deklarácií globálnych premenných, deklarácií funkcií a definícií funkcií. Deklarácia funkcie deklaruje typ funkcie, meno funkcie a zoznam parametrov, definícia obsahuje aj telo. Narozdiel od Pascalu, v C neexistuje „hlavný program“ a nie je rozdiel medzi „procedúrou“ a „funkciou“. Deklarácie funkcií a premenných, ktoré majú byť dostupné aj z iných modulov, sa zväčša zapisujú do súboru s príponou .h. Súbory s príponou .h (analogický koncept v Pascale je *interface* časť unit-u) sa vkladajú do zdrojových súborov iných modulov pomocou direktívy *#include*. Komentáre v C sa začínajú /* a končia */. Medzery, tabulátory a konce riadkov sú rovnocenné a nie sú zaujímavé (až na ukončovanie mena). Mená typov, premenných, funkcií a pod. začínajú písmenom a obsahujú písmená, číslice a podtržník. Pozor – veľkosť písmen je podstatná (narozdiel od Pascalu).

```
/* Vloženie súboru stdlib.h, ktorý sa hľadá v systémových adresároch (na UNIX-
ových systémoch /usr/include) */
#include <stdlib.h>

/* Vloženie súboru modul1.h, ktorý sa hľadá v aktuálnom adresári alebo
používateľom špecifikovaných adresároch */
#include "modul1.h"

/* Deklarácia globálnej premennej a typu int */
int a;

/* Definícia funkcie f typu int, ktorá berie jeden argument typu int*/
int f(int x)
{
  int b; /* lokálna premenná */

  b = (x + 5)*a;
  if (b > 0) return b;

  /* volanie funkcie g, ktorá nič nevracia, alebo nás výsledok nezaujíma */
  g(4, b);

  return 0;
}
```

Vo vyššie uvedenom príklade prepokladáme, že modul1.h obsahuje deklaráciu funkcie g. Mohol by vyzerať napríklad takto:

```
void g(int cislo, int cislo2);
```

Nič zo súboru stdlib.h sme v uvedenom fragmente nepoužili, jeho vloženie bolo len ilustračné.

Konštanty (čísla, znaky, reťazce)

Celé čísla môžeme zapisovať v desiatkovej, osmičkovej a šestnástkovej sústave:

- číslo v desiatkovej sústave začína číslicou 1 až 9 a obsahuje číslice 0 až 9,
- číslo v osmičkovej sústave začína číslicou 0 a obsahuje číslice 0 až 7,
- číslo v šestnástkovej sústave začína *0x* a pokračuje číslicami 0 až 9, a, b, c, d, e, f (písmená môžu byť aj veľké).

Príklad: 17, 021, 0x11 predstavujú tú istú hodnotu.

Znakové konštanty sa zapisujú v jednoduchých apostrofoch: *'A'*. Špeciálne znaky sa dajú zapísať pomocou ich kódu v osmičkovej alebo šestnástkovej sústave: *'\101'* a *'\x41'* reprezentujú znak s kódom 65, t.j. *'A'*. Niektoré špeciálne znaky majú aj vlastné zápisy:

- '\n' – LF – prechod na nový riadok, kód 10
- '\r' – CR – kód 13
- '\t' – HT – tabulátor
- '\f' – FF – form feed – prechod na novú stranu
- '\b' – BS – back space
- '\a' – BEL – pípnutie
- '\\' – \
- '\'' – '
- '\"' – "

Znakové reťazce sa zapisujú do úvodzoviek ("). Ak obsahujú špeciálne znaky, tie sa zapisujú rovnako ako v zankových konštantách, t.j. pomocou \ a kódu v osmičkovej alebo šestnástkovej sústave alebo pomocou špeciálneho mena podľa vyššie uvedenej tabuľky. Znakové reťazce sú automaticky ukončené znakom *'\0'*.

Jazyk C nemá špeciálnu konštrukciu pre pomenované konštanty (ako const v Pascale), zvyčajne sa na to používa triviálna definícia makra:

```
#define meno hodnota
```

Keď sa potom použije *meno*, je to ekvivalentné použitiu uvedenej hodnoty.

Základné dátové typy

Základné celočíselné typy (to zahŕňa typy pre znaky, a necelými číslami sa nebudeme zaoberať) sú:

signed char	8 bitov	-128, 127
unsigned char	8 bitov	0, 255
char	8 bitov	gcc na Linuxe na i386 = signed char
signed short int, signed short, **short**	16 bitov	-32768, 32767
unsigned short int, **unsigned short**	16 bitov	0, 65535
signed int, **int**		gcc na Linuxe na i386 = signed long
unsigned int, **unsigned**		gcc na Linuxe na i386 = unsigned long
signed long int, signed long, **long**	32 bitov	-2147483648, 2147483647
unsigned long int, **unsigned long**	32 bitov	0, 4294967295

Typy *int* a *unsigned* sú na 16-bitových platformách zvyčajne zhodné s *short* a *unsigned short*. Podľa štandardu musia byť aspoň 16 bitové, na 32-bitových platformách sú zvyčajne 32 bitové, t.j. zhodné s *long* a *unsigned long*.

Typy struct, union a enum

Typ *struct* umožňuje definovať typ, ktorého hodnoty majú niekoľko položiek (analógia typu record z Pascalu). Typ *union* umožňuje pristupovať k tomu istému miestu v pamäti ako k premenným niekoľkých typov (v Pascale var časť recordu). Typ *enum* je celočíselný typ, ktorý má hodnoty pomenované menami.

```
struct cas {
  unsigned char hodina;
  unsigned char minuta;
  unsigned char sekunda;
};

union cislo {
  unsigned char znak;
  unsigned short male_cislo;
  unsigned long velke_cislo;
};
/* všetky položky sú na rovnakej adrese, teda zmysel má väčšinou len jedna */

enum pohlavie {zena, muz}; /* zena má hodnotu 0, muz 1 */
enum farba {cervena=1, biela=3, modra=5, zlta, cierna};
/* mená majú explicitne priradené hodnoty, tie, čo nemajú, ich majú priradené
automaticky vždy zväčšením o 1 */
```

Meno takéhoto typu sa skladá zo slova *struct*, *union*, resp. *enum* a definovaného mena. Tiež je možné definíciu typu spojiť priamo s deklaráciou premennej. V takom prípade je možné meno za *struct, union,* či *enum* vynechať.

Deklarácie premenných

Deklarácia premennej má základný tvar:

```
typ deklarátor1, deklarátor2;
```

Deklarátor jednoduchej premennej je meno premennej. Deklarátor poľa je *meno[počet prvkov]*, deklarátor smerníku je ***meno*. Polia sú indexované vždy od nuly a kompilátor nerobí kontrolu hraníc!

```
int a, pole[5], *p;
/* a je premenná typu int, pole je 5-prvkové pole int-ov, p je smerník na int */

struct cas cas;
/* označenia structov, unionov a enumov nekolidujú s menami premenných */

union {
  int i;
  short s;
  char c;
} h;
/* spojenie definície unionu/structu s deklaráciou premennej a vynechanie mena*/

/* zložitejšie konštrukcie */
char *x[10]; /* x je 10-prvkové pole, každý prvok je smerník na char */
char (*y)[10]; /* y je smerník na 10-prvkové pole charov - [] majú vyššiu
prioritu ako * */
```

Deklarácie premenných môžu obsahovať aj inicializačnú hodnotu (inak sú globálne premenné inicializované na 0):

```
int a = 10;
int b[3] = {10, 20, 30};

unsigned char c[] = "Hello World";
/* Pole znakov sa môže inicializovať reťazcovou konštantou, nešpecifikovaný počet
prvkov sa určí podľa hodnoty, t.j. v tomto prípade 12 */
```

Definovanie vlastných typov

Tak ako je možné deklarovať premennú daného typu, je možné danému typu definovať vlastné meno, ktoré potom môže byť použité rovnako ako základný typ pri deklaráciach. Je to vhodné najmä pre zložitejšie typy.

Meno nového typu sa pri definovaní píše na mieste, kde by sa pri normálnej deklarácii nachádzalo meno premennej.

```
typedef int * smernik_na_int;
typedef struct {int a; int b;} X;

smernik_na_int p;
X x;
```

Deklarácie a definície funkcií

Ako sme už spomenuli v úvode, v C nerozlišujeme medzi funkciou a procedúrou. C definuje špeciálny typ *void*, ktorého množina hodnôt je prázdna a v súvislosti s funkciami slúži pre funkcie, ktoré nevracajú žiadnu hodnotu (v terminológii Pascalu procedúry). Funkcie v C prijímajú parametre len hodnotou; ak je potrebné predávať parametre odkazom (var parametre v Pascale), musí sa to riešiť pomocou smerníkov.

Deklarácia funkcie vyzerá nasledovne:

```
typ meno(typ1 arg1, typ2 arg2);

int scitaj(int a, int b);
void vypis(int a);
```

Definícia funkcie je deklarácia funkcie nezakončená bodkočiarkou a nasledovaná telom funkcie v zložených zátvorkách. Telo funkcie sa skladá z deklarácií lokálnych premenných nasledovaných príkazmi.

```
typ meno(typ1 arg1, typ2 arg2)
{
telo
}

int scitaj(int a, int b)
{
  return a+b;
}
```

Polia versus smerníky

V jazyku C je možné smerník považovať za pole (ktoré začína na adrese, kam smerník ukazuje) a naopak meno poľa je možné považovať za konštantný smerník, ktorý ukazuje na prvý prvok poľa (prvok s indexom 0). Toto je dôležité napr. pri použití polí ako argumentov funkcie – do funkcie vstupuje smerník. Nasledujúce dve deklarácie sú v podstate rovnaké:

```
void f(char * p);
void f(char p[]); /* počet prvkov netreba špecifikovať, je irelevantný */
```

Výrazy

Jazyk C obsahuje pomerne veľký počet operátorov, ktoré majú 15 úrovní priority. Začiatočníkom sa odporúča v zložitejších výrazoch radšej zátvorkovať (normálne okrúhle zátvorky), niekedy je priorita niektorých operátorov zradná. Dôležitým rozdielom oproti Pascalu je fakt, že aj priradenie je výraz – jeho hodnotou je priradená hodnota. Pozor na to, že operátor = je priradenie, test na rovnosť je ==.

Výrazy manipulujú s číslami (znaky a enum-y sú čísla). Niektoré operátory (priradenie, porovnávanie, odčitovanie, pripočítavanie čísla, inkrement, dekrement) sa dajú použiť aj na smerníky (pripočítanie čísla k smerníku spôsobí posun smerníku o daný počet položiek veľkosti typu, na ktorý smerník ukazuje). Priraďovací operátor sa môže použiť aj na struct-ové a union-ové typy. Nie je však možné priraďovacím operátorom napríklad skopírovať pole (na to treba spraviť cyklus).

Operandy môžu byť vo všeobecnosti premenné, konštanty alebo iné výrazy.

Aritmetické operátory

-x – opačná hodnota

*x+y, x-y, x*y, x/y, x%y* – sčitovanie, odčitovanie, násobenie, celočíselné delenie, zvyšok po delení

Bitové operátory

~x – bitová negácia

x&y – bitový AND

x^y – bitový XOR

x|y – bitový OR

x<<y – bitový posun hodnoty x o y bitov doľava

x>>y – bitový posun hodnoty x o y bitov doprava

Logické operátory

Nulová hodnota znamená false, nenulová hodnota znamená true.

!x – logická negácia

x&&y – logický AND

x||y – logický OR

Porovnávacie operátory

x==y – test na rovnosť

x!=y – test na nerovnosť

x<y, x<=y, x>y, x>=y – menší, menší alebo rovný, väčší, väčší alebo rovný

Priraďovacie operátory

x=y – jednoduché priradenie

x+=y, x-=y, x=y, x/=y, x%=y, x<<=y, x>>=y, x&=y, x^=y, x|=y* – priradí do *x* hodnotu *x op y*, kde *op* je operátor pred =, teda napr. *x+=2* je to isté ako *x=x+2*.

Inkrementačné a dekrementačné operátory

x++, x-- – postinkrement, postdekrement – zväčší/zmenší hodnotu *x* o 1, hodnota výrazu je pôvodná hodnota *x*

++x, --x – preinkrement, predekrement – zväčší/zmenší hodnotu *x* o 1, hodnota výrazu je nová

hodnota *x*

Špeciálne operátory

x[y] – políčko v poli *x* na indexe *y*; *x* musí byť pole alebo smerník, *y* musí byť číselný výraz

x.y – položka *y* v *x*; *x* je struct alebo union, *y* je meno položky

x(y) – volanie funkcie; *x* je meno funkcie alebo smerník na funkciu, *y* je zoznam argumentov

**x* – dereferencia smerníka; *x* je smerník, **x* je premenná (miesto v pamäti), kam *x* ukazuje; je to to isté ako *x[0]*

x->y – položka *y* v **x*; *x* je smerník na struct alebo union, *y* je meno položky

&x – adresa premennej *x*

sizeof x – veľkosť premennej *x* (alebo typu *x*)

(typ)x – pretypovanie *x* na typ *typ*

z ? x : y – ak *z* je nenula, tak *x*, inak *y*

x,y – vyhodnotí najprv *x*, hodnotu zahodí, potom vyhodnotí *y*

Pri logických operátoroch (&&, ||) sa používa skrátené vyhodnocovanie – najprv sa vyhodnotí ľavý operand, ak z jeho hodnoty už vyplýva hodnota výrazu, pravý operand sa nevyhodnocuje. Pri iných binárnych operátoroch (ešte okrem čiarky) nie je poradie vyhodnotenia ľavého a pravého operandu určené. V nasledujúcej tabuľke sú operátory uvedené podľa ich priority (precedencie) pri vyhodnocovaní. Operátory s rovnakou prioritou sa asociujú zľava (L) alebo sprava (R).

priorita (1=najvyššia)	***operátory***	***asociovanie***
1	x++, x--, x[y], x(y), x.y, x->y	L
2	sizeof x, ++x, --x, &x, *x, -x, ~x, !x, (typ)x	R
3	x*y, x/y, x%y	L
4	x+y, x-y	L
5	x<<y, x>>y	L
6	x<y, x<=y, x>y, x>=y	L
7	x==y, x!=y	L
8	x&y	L
9	x^y	L
10	x|y	L
11	x&&y	L
12	x||y	L
13	z?x:y	R
14	x=y, x+=y, x-=y, x*=y, x/=y, x%=y, x<<=y, x>>=y, x&=y, x^=y, x|=y	R
15	x,y	L

Príkazy

; – prázdny príkaz.

`výraz;`
– vyhodnotí sa výraz, jeho výsledok sa zahodí – užitočné, ak má výraz nejaký vedľajší efekt (napr. priradenie, inkrement, volanie funkcie, ktorá má aj nejaký iný efekt ako spočítanie hodnoty).

`{ príkaz1; príkaz2; ... príkazN; }`
– blok – podobne ako begin a end v pascale, príkazy sa vykonávajú postupne, pred prvým príkazom

môžu byť aj deklarácie premenných, ktoré existujú len počas vykonávania príkazov v bloku – po opustení bloku zaniknú (definícia funkcie je v podstate deklarácia nasledovaná blokom).

```
if (výraz) príkaz
```
– príkaz sa vykoná, ak je výraz true, t.j. nenulový.

```
if (výraz) príkaz1 else príkaz2
```
– ak je výraz true, vykoná sa príkaz1, inak príkaz2; narozdiel od Pascalu sa nevynecháva bodkočiarka pred else.

```
while (výraz) príkaz
```
– kým je výraz nenulový (true), vykonáva sa príkaz.

```
do príkaz while (výraz);
```
– vykoná sa príkaz, potom sa vyhodnotí výraz, ak je nenulový, cyklus sa zopakuje, ak je nulový, cyklus končí (t.j. analógia pascalovského repeat ... until, ale pravdivý výraz neznamená koniec ale opakovanie).

```
for(init; podm; iter) príkaz
```
– najprv sa vyhodnotí výraz *init*, jeho hodnota sa zahodí (dôležité sú vedľajšie efekty – najčastejšie priradenie), testuje sa výraz *podm*, ak je nulový, cyklus skončí, ak je nenulový, vykoná sa príkaz, potom sa vyhodnotí výraz *iter*, hodnota sa zahodí a opäť sa vykonávanie vráti na vyhodnotenie výrazu *podm*; dá sa to celé prepísať ako:

```
init; while (podm) { príkaz; iter; }
```
Všetky tri výrazy vo *for* môžu byť aj komplikované, napr. pomocou operátora „čiarka“ je možné do výrazov *init* a *iter* spojiť inicializáciu a modifikáciu niekoľkých premenných, môžu obsahovať volania funkcií a pod.

```
switch (výraz)
{
case k1: príkazy1
case k2: príkazy2
default: príkazyd
}
```
– vyhodnotí výraz, ak sa hodnota zhoduje s konštantným výrazom *ki* v niektorom *case*, pokračuje prvým príkazom v *príkazyi*. Ak sa nezhoduje, pokračuje prvým príkazom v *prikazyd*, ak existuje *default*, alebo pokračuje za koncom *switch* bloku. Narozdiel od pascalovského case vykonávanie nekončí vykonaním posledného príkazu v *prikazyi*, ale pokračuje ďalej, ak nie je prerušené príkazom *break.*

```
break;
```
– príkaz *break* spôsobí skok za koniec najvnútornejšieho *for, while, do, switch.*

```
continue;
```
– príkaz *continue* spôsobí skok na koniec cyklu *for, while, do*, t.j. na vyhodnotenie podmienky cyklu.

```
return výraz;
```
– príkaz *return* ukončí vykonávanie funkcie a vráti hodnotu výrazu ako výsledok funkcie. Pre funkciu typu *void* sa výraz neuvádza.

Funkcia *main*

Vykonávanie programu v jazyku C začína v „neviditeľnom“ obale, ktorý pripraví prostredie a následne zavolá funkciu *main*. Po skončení funkcie *main* sa jej hodnota použije ako návratový kód programu – ak program skončil normálne, mal by byť 0, inak by mal byť nenulový.

```
int main(unsigned argc, char * args[])
{
  unsigned i=0;
  while (i < argc){ printf("args[%d]=%s\n", i, args[i]); i++; }
  return 0;
}
```
Prvým argumentom je počet parametrov na príkazovom riadku vrátane nultého parametra, ktorým je

meno (cesta) k programu. Druhý argument je pole smerníkov na text jednotlivých parametrov.

Funkcie *printf, snprintf, scanf, sscanf*

Tieto funkcie sú deklarované v súbore *stdio.h* a slúžia na formátovaný textový výstup a vstup. Funkcia *printf* zapisuje na štandardný výstup programu, funkcia *scanf* číta zo štandardného vstupu. Funkcia *snprintf* zapisuje do poľa znakov a funkcia *sscanf* číta textovú reprezentáciu z poľa znakov.

```
int printf(const char *format, ...);
int snprintf(char *str, size_t size, const  char  *format, ...);
```

Funkcia *snprintf* zapisuje nulou zakončenú postupnosť znakov do poľa *str* a nezapíše viac ako *size* bytov (vrátane zakončovacej nuly). Účelom je, aby výstup neprepísal pamäť mimo miesta vyhradeného pre pole. Funkcie vracajú dĺžku výsledného reťazca bez zakončovacej nuly, v prípade *snprintf* je to dĺžka, ktorú by funkcia chcela zapísať. Preto ak je vrátená hodnota rovná alebo väčšia ako *size*, znamená to, že reťazec bol skrátený pre nedostatok miesta.

Argument *format* predstavuje formátovací reťazec, ktorý okrem textu (ten sa na výstup kopíruje) môže obsahovať formátovacie príkazy, ktoré sa na výstupe nahradia naformátovanou hodnotou príslušného argumentu. Funkcie akceptujú premenlivý počet argumentov – pre každý riadiaci príkaz vo formátovacom reťazci, ktorý potrebuje hodnotu, je potrebný ďalší argument. Formátovacie príkazy začínajú znakom percento (*%*), za ním môžu nasledovať príznaky, za nimi môže nasledovať minimálna šírka políčka, za ňou môže nasledovať presnosť, za ňou môže nasledovať modifikátor veľkosti a na konci je znak určujúci typ konverzie.

príznak	***význam***
0	číslo sa zľava doplní nulami namiesto medzerami
-	hodnota sa zarovnáva doľava namiesto doprava
medzera	pred kladným číslom sa nechá jedna medzera
+	číslo sa vždy uvedie znamienko (inak sa uvádza len záporným)

Minimálna šírka políčka určuje najmenší počet znakov, ktorý bude naformátovaná hodnota zaberať. Ak je počet znakov hodnoty menší, doplní sa zľava alebo sprava medzerami, príp. zľava nulami – podľa príznakov.

Presnosť pre celé čísla určuje minimálny počet cifier, ktoré budú zobrazené (tiež sa dá využiť na doplnenie núl pred číslo). Pre reťazec znamená maximálny počet znakov, ktoré sa vypíšu (zvyšok sa usekne).

Modifikátor veľkosti určuje, či sa číslo interpretuje ako *short* (h) alebo *long* (l), prípadne *char* (hh) a *long long* (ll) (*long long* sme nespomínali, nie je to štandardný C typ, v gcc na i386 je to 64 bitové číslo, *hh* a *ll* nie sú štandardné, ale gcc ich pozná).

typ konverzie	***význam***
d,i	znamienkové celé číslo v desiatkovej sústave
o,u,x,X	neznamienkové celé číslo v osmičkovej (o), desiatkovej (u), šestnástkovej (x,X) sústave (x – malé písmená, X – veľké písmená)
c	znak
s	znakový reťazec zakončený nulou
p	smerník zapísaný ako číslo v šestnástkovej sústave
%	%, nespotrebuje žiadny argument, nepripúšťa žiadne voliteľné hodnoty

Príklad:

```
printf("Cislo: %04d, znak: %c, retazec: %s\n", 65, 65, "Ahoj");

Cislo: 0065, znak: A, retazec: Ahoj
```

Na konverziu textového vstupu slúžia funkcie *scanf* a *sscanf*:

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

Funkcia *sscanf* analyzuje pole znakov *str* zakončené nulou, *scanf* analyzuje štandardný vstup. Argument *format* obsahuje formát vstupu podobný ako formát výstupu v *printf*. Medzery zodpovedajú ľubovoľnému počtu medzier a tabulátorov vo vstupe, iné znaky zodpovedajú samé sebe, riadiace príkazy začínajú percentom, ďalej môže byť číslo predstavujúce maximálny počet znakov (ak nie je uvedená, znamená nekonečno, určuje max. počet znakov, ktoré sa použijú pre príslušnú konverziu), modifikátor veľkosti (h alebo l) a znak určujúci typ konverzie. Každá konverzia potrebuje ďalší argument, ktorým je smerník na miesto, kam sa má hodnota uložiť. Funkcie vrátia počet úspešných konverzií (môže byť menší ako požadovaný počet, ak napríklad vstup obsahoval znak, ktorý nebolo možné spracovať).

typ konverzie	***význam***
%	zodpovedá zanku % na vstupe, nespotrebuje žiadny argument
d	celé číslo, ktoré môže mať znamienko, argument musí byť typu *int* *, prípadne *short *, long* *, ak je použitý modifikátor *h,l*
i	ako d, ale číslo môže byť aj v osmičkovej alebo šestnástkovej sústave, ak je zapísané spôsobom ako konštanta v C
u	neznamienkové číslo v desiatkovej sústave, argument je typu *unsigned* *, prípadne *unsigned short *, unsigned long* *podľa modifikátora
o	ako *u*, ale v osmičkovej sústave
x,X	ako *u*, ale v šestnástkovej sústave
s	znakový reťazec po prvú medzeru/tabulátor/koniec riadka; argument je *char* *, ktorý musí ukazovať na dostatočne veľké miesto pre uloženie všetkých znakov (nepoužívať bez obmedzenia max. počtu!) aj ukončovacej 0
c	1 znak, argument je typu *char* *; ak je uvedný max počet znakov, tak kopíruje zadaný počet znakov (pozor na dostatočnú veľkosť poľa), nepridáva žiadny ukončovač, nezastavuje na medzerách, nepreskakuje medzery na začiatku

Príklad:

```
int main()
{
  unsigned short u = 0;
  char rest[100] = {0};
  int n;
  n = scanf("%hu%99s", &u, rest);
  printf("n=%d,u=%u,rest=%s\n", n, u, rest);

  return 0;
}

vstup:
17 hello world
výstup:
n=2,u=17,rest=hello
```