Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

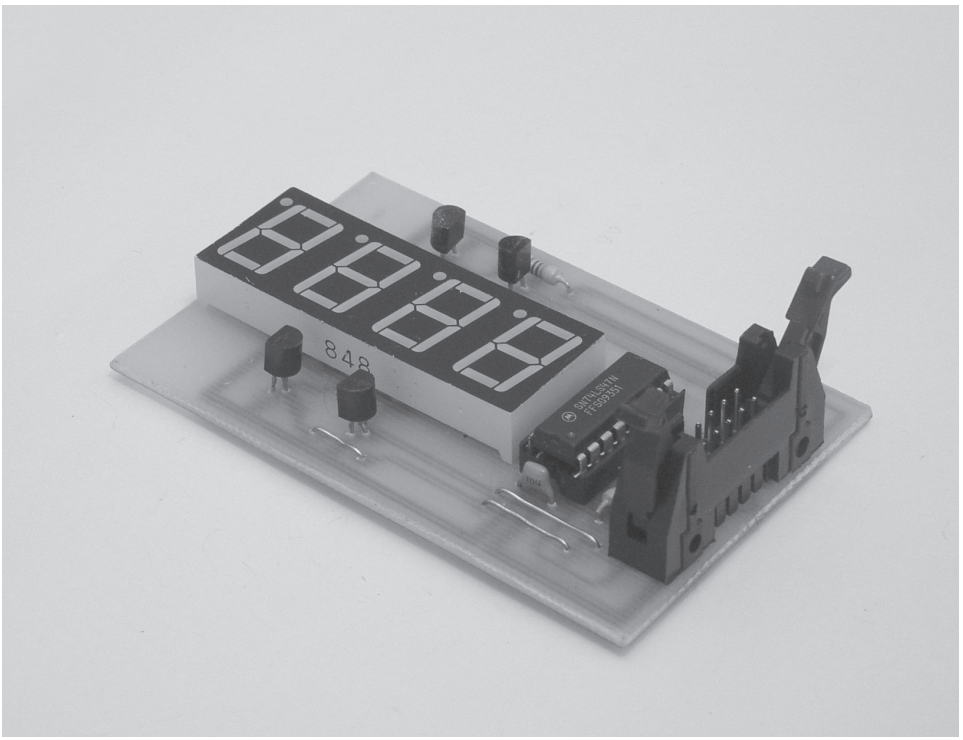
To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

14

UKAZATELE A ŘETĚZCE



V této kapitole budou probrány zbývající datové typy jazyka C: ukazatel a řetězec.

14.1 UKAZATEL (POINTER)

Obsahem proměnné typu ukazatel není hodnota čísla, znaku nebo něco podobného, ale adresa jiné proměnné.

Definice ukazatele

Pro definici ukazatele se používá symbol *. Níže je uveden příklad definice proměnných **i**, **j**, **p**.

Proměnné **i**, **j** jsou obyčejná celá čísla (typ **int**), **p** je typu `int*` (ukazatel na celočíselnou proměnnou). Také můžeme definovat nový typ ukazatel (zde je označen **pint**):

```
int i,*p,j;
.
.
typedef int * pint;
```

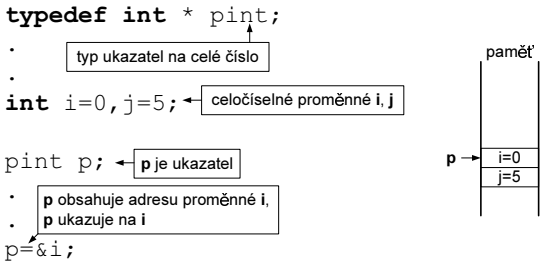
Obr. 14.1 Definice ukazatele

Unární operátor & (reference)

Unární operátor & získá adresu proměnné. Zapisuje se v prefixové podobě, tedy **&x**. Tento operátor můžeme použít na libovolnou proměnnou (každá proměnná má svou adresu v paměti).

Zápis **p = &i** získá adresu proměnné **i** a uloží ji do ukazatele **p**.

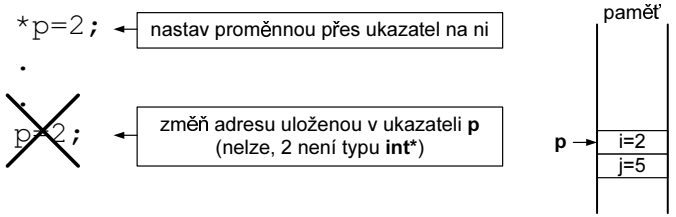
Říkáme: **p** ukazuje na **i**. Viz *obr. 14.2*.



Obr. 14.2 Příklad použití operátoru reference &

*Unární operátor * (dereference)*

Unární operátor `*` zajistí přístup k proměnné přes ukazatel. Zapisuje se v prefixové podobě, tedy **`*p`**. Tento operátor lze použít pouze na proměnné typu ukazatel.



*Obr. 14.3 Příklad použití operátoru dereference **

*Typ void**

Obvyklé je používat ukazatel stejného typu, jakého je proměnná, na kterou ukazuje. Pokud ale vyžadujeme adresu bez ohledu na typ dat, můžeme použít tzv. obecný ukazatel typu **void***.

Ovšem vzhledem k tomu, že typ **void** nemá určenu velikost, nelze ukazatel typu **void*** dereferencovat!

Hodnota NULL

Existují situace, kdy potřebujeme stanovit, že ukazatel neobsahuje adresu žádné proměnné. Pro tyto případy je definován symbol **NULL**, který představuje adresu hodnoty **0**.

Nulovou adresu nemůže mít žádná proměnná. Když ukazatel obsahuje hodnotu **NULL**, bereme to tak, že není nastaven na žádnou proměnnou.

Symbol **NULL** je definován například v hlavičkovém souboru **stdlib.h**.

Velikost ukazatelů

Jako každá jiná proměnná, je ukazatel uložen do paměti. Mělo by nás tedy zajímat, kolik bajtů v paměti zabírá.

Zajímavé je, že všechny ukazatele mají stejnou velikost, která není závislá na jejich typu. Tato velikost je určena adresovacími schopnostmi použitého mikrokontroléru. Vzhledem k šíři adresové sběrnice (16 bitů) je jasné, že ukazatele jsou realizovány jako 2bajtové proměnné. Platí to i pro typ **void***.

Volání parametrů funkce přes ukazatel

Jedno ze základních použití ukazatelů spočívá ve volání parametrů přes ukazatel. Takto lze realizovat parametry funkcí, které lze měnit.

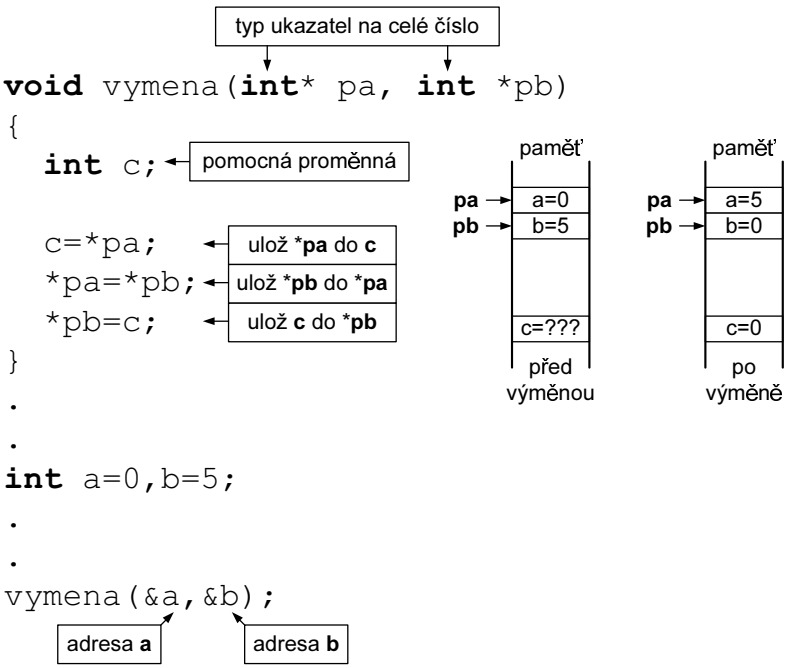
Jako první příklad si uvedeme funkci **vymena**, která má dva parametry typu ukazatel na celé číslo. Jedná se o funkci, která má zajistit výměnu obsahu dvou celočíselných proměnných. Viz *obr. 14.4*.

Pro případ funkce, do které se parametry předávají přes ukazatel, musí být formální parametry typu ukazatel daného typu. V našem příkladu jsou parametry označeny **pa** a **pb** a jsou typu `int*`.

Pro vlastní výměnu je nutné mít jednu pomocnou proměnnou typu celé číslo, je definována jako **c**. Výměna probíhá takto:

- hodnotu první proměnné uložíme do **c**, přístup k první proměnné přes ukazatel **pa** obdržíme pomocí operátoru dereference (tedy *pa),

- nyní již můžeme obsah první proměnné (`*pa`) přepsat hodnotou druhé proměnné (`*pb`),
- nakonec do druhé proměnné (přístup je přes `*pb`) uložíme původní hodnotu první proměnné (**c**).



*Obr. 14.5 Funkce **soucet** a její použití*

Všimněte si, jaký je význam proměnných **pa**, **pb**, **a**, **b**. Proměnné **a**, **b** jsou celá čísla a mají svou adresu v paměti. Proměnné **pa**, **pb** na ně ukazují. Před výměnou ukazuje **pa** na **a**, **pb** na **b**. Po výměně se situace nezmění (**pa** stále ukazuje na **a**, **pb** na **b**), pouze obsahy proměnných **a**, **b** jsou vzájemně vyměněny.

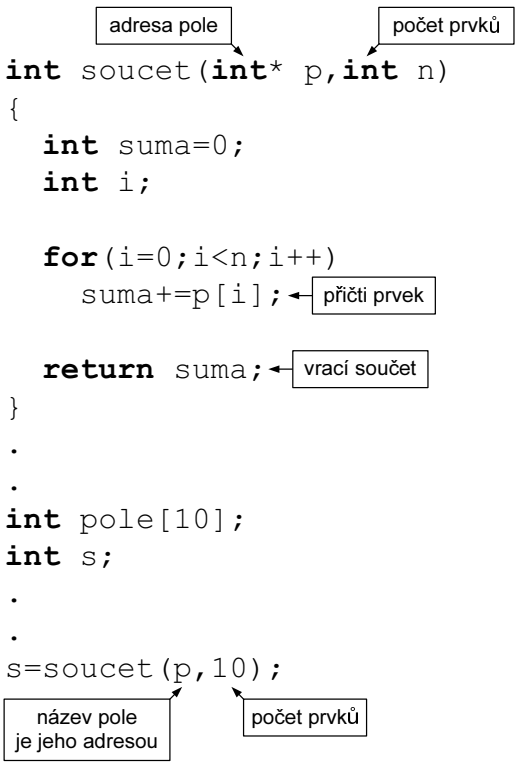
Všimněte si také, že při volání funkce **vymena** musíme předat adresy proměnných **a**, **b** (uloží se do ukazatelů **pa**, **pb**). Adresa proměnné je získána pomocí operátoru `&`.

Souvislost ukazatele s polem

Název pole představuje ukazatel na jeho začátek (prvek s indexem 0). Adresu libovolného prvku lze získat pomocí operátoru reference (`&`).

Výše uvedené informace lze zužitkovat, pokud zapisujeme funkci s parametrem typu pole. Pole bude předáváno pomocí ukazatele. Jelikož uvnitř funkce již nelze velikost pole zjistit (uvnitř pole máme pouze informaci o adrese pole, nikoli o poli samotném; i když můžeme přistupovat k jeho prvkům), musíme ještě předávat maximální počet prvků pole.

Jako příklad si uvedeme funkci **soucet**, která stanoví součet hodnot všech prvků pole celých čísel. Prvním parametrem je ukazatel typu `int*`, druhým parametrem je celé číslo – počet prvků pole. Viz *obr. 14.5*.



*Obr. 14.5 Funkce **soucet** a její použití*

Práce s polem uvnitř funkce je velmi jednoduchá. Dokonce nemusíme používat ani operátor dereference – k prvkům přistupujeme indexováním.

Také při volání funkce **soucet**, je vše také snadné. Neobjeví se dokonce ani zápis operátoru reference (název pole je rovnou jeho adresou).

14.2 ŘETĚZEC

Řetězec je pole znaků, se kterými se pracuje najednou. Řetězec je reprezentován jako jednorozměrné pole znaků, čili jako typ: **char[ ]**.

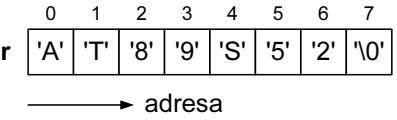
V souvislostmi s pomocnými funkcemi se bude objevovat i zápis **char***. Již ale víme, že název pole odpovídá jeho adrese.

Připomeňme, že pole se v jazyce C++ indexuje vždy od nuly. To znamená, že první znak řetězce je uložen v prvku s indexem 0. Další znaky jsou pak uloženy v následujících prvcích.

Délka řetězce není stanovena přímo, ale pomocí tzv. zarážky. **Zarážka** je znak s ASCII kódem 0, tedy **'\0'**. Jedná se o znak, který nelze zobrazit na výstupním zařízení nebo jej přečíst ze vstupního zařízení.

Při definici řetězce je třeba uvážit, že zarážka samotná zabírá také jednu pozici.

Pokud chceme, aby proměnná **r** typu řetězec obsahovala text „AT89S52“, musíme ji definovat jako alespoň 8prvkové pole znaků. Viz *obr. 14.6*.



Obr. 14.6 Jednotlivé znaky řetězce ***r***

Řetězcový literál a inicializace řetězce

Řetězcový literál se zapisuje mezi uvozovky (připomeňme, že znakový literál se zapisuje mezi apostrofy). Literál je chápán jako celý řetězec, to znamená, že za poslední znak je automaticky připojena zarážka.

Inicializaci řetězce při definici je možno provádět podobně, jako u prostého pole:

```
char r[20]="AT89S52";
```

Také můžeme využít skutečnosti, že překladač počet znaků spočítá (nyní je fyzická délka 8 prvků):

```
char r[]="AT89S52";
```

Poslední variantou je využití souvislosti pole s ukazatelem. Níže uvedený zápis je rovněž možný, nepreferujeme jej však:

```
char *r="AT89S52";
```

Operace přiřazení (=) není pro řetězec definována. Takže přiřadit hodnotu lze pouze v rámci inicializace. Jinak musíme používat funkci **strcpy**.

Podobně operátory <, <=, >, >= také nejsou definovány. Jejich použití vede k porovnání adres řetězců. Korektní porovnání musíme provést pomocí funkce **strcmp**.

Pomocné funkce pro práci s řetězci

Pomocné funkce pro práci s řetězci jsou k dispozici v podobě hlavičkového souboru **string.h**. Zde je seznam nejpoužívanějších funkcí včetně krátkého popisu:

- unsigned **strlen**(const char ***s**)

  – vrátí délku řetezce **s**,

- char* **strcpy**(char ***dest**, const char ***src**)

  – zkopíruje znaky řetězce **src** do řetězce **dest**,

- char* **strcat**(char ***dest**, const char ***src**)

  – spojí dva řetězce, tedy připojí řetězec **src** za řetězec **dest**,

- char* **strchr**(const char ***s**, int **c**)
  - vyhledá první výskyt znaku **c** v řetězci **s**,
  - pokud je znak nalezen, vrátí jeho adresu,
  - při nenalezení vrátí **NULL**,
- char* **strstr**(const char ***s1**, const char ***s2**)
  - vyhledá první výskyt podřetězce **s2** v řetězci **s1**,
  - pokud je podřetězec nalezen, vrátí jeho adresu,
  - při nenalezení vrátí **NULL**,
- int **strcmp**(const char ***s1**, const char ***s2**)
  - porovná abecedně dva řetězce:
  - je-li s1<s2, vrátí výsledek <0,
  - je-li s1==s2, vrátí výsledek 0,
  - je-li s1>s2, vrátí výsledek >0.

Ukázka použití řetězcových funkcí:

```
#include <string.h>          ← hlavičkový soubor string.h

void main()
{
  char r1[20];
  char r2[20];               ← definice tří řetězců
  char r3[40];
  int d;

  strcpy(r1,"mikrokontroler");   ← r1 obsahuje mikrokotroler
  strcpy(r2,"AT89S52");          ← r2 obsahuje AT89S52

  d=strlen(r1);              ← d je délka řetězce r1, tedy 14

  strcpy(r3,r1);             ← r3 obsahuje mikrokontroler
  strcat(r3," ");            ← r3 obsahuje mikrokontroler + mezera
  strcat(r3,r2);             ← r3 obsahuje mikrokontroler AT89S52

  d=strcmp(r1,r2);           ← d je výsledek porovnání r1, r2
                               tedy d<0 protože r1 je abecedně před r2
}
```