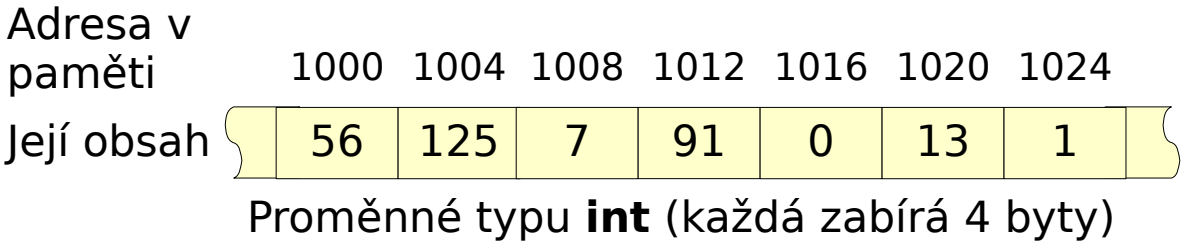
Programování v jazyce C pro chemiky
(C2160)

7. Ukazatele, funkce pro práci s řetězci

Ukazatele

- Každá proměnná je umístěna na konkrétním místě v paměti
- Paměť je organizována lineárně jako posloupnost bytů
- Každému bytu přísluší adresa, tj. pořadí bytu v paměti
- V jazyce C někdy potřebujeme zjistit adresu v paměti, kde je uložen obsah proměnné, tuto adresu ukládáme do proměnných nazývaných ukazatel (anglicky pointer)



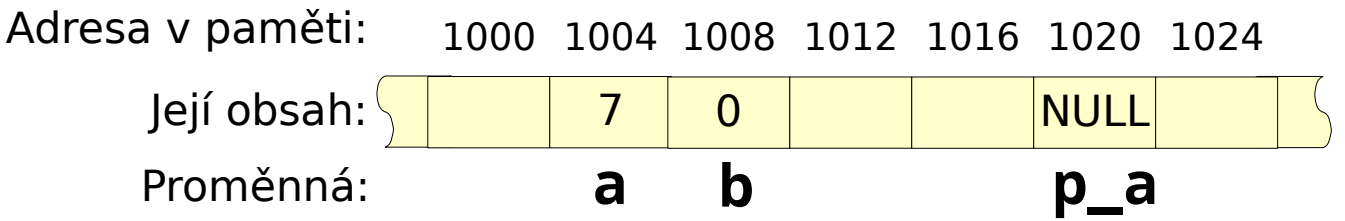
Definice ukazatele

- Rozlišujeme různé typy ukazatelů podle toho, zda ukazují na proměnnou typu **`int`**, **`float`**, **`char`** a podobně
- Všechny ukazatele obsahují adresu v paměti, kde se proměnná nachází, rozlišení typu určuje jen způsob interpretace cílové proměnné
- Ukazatel definujeme podobným způsobem jako běžnou proměnnou, před název proměnné však uvedeme hvězdičku *
- Ukazatele je vhodné inicializovat hodnotou NULL, která představuje neplatnou adresu (stejný efekt má číselná konstanta 0)
- Názvy ukazatelů někdy začínají na p nebo p_, aby bylo možné snadno rozpoznat, že se jedná o ukazatel
- Příklad definice ukazatelů:

```
int *p_n = NULL;
float *p_a = NULL, *p_b = NULL;
```

Referenční a dereferenční operátor

- Pro zjištění adresy proměnné používáme referenční operátor **&** uvedený před názvem proměnné jejíž adresu chceme zjistit; tuto adresu zpravidla přiřadíme do ukazatelové proměnné
- Dereferenční operátor * provádí opačnou akci než referenční operátor. Uvádíme jej před názvem ukazatelové proměnné, abychom přistoupili k obsahu proměnné nacházející se na adrese uložené v ukazateli.
- POZOR: Znak * v definici ukazatele nepředstavuje dereferenční operátor, ale pouze poskytuje informaci, že jde o ukazatelovou proměnnou

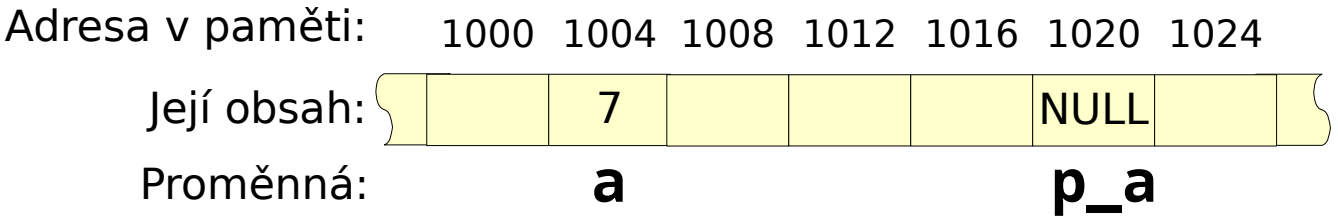



```
int a = 7, b = 0;  // Definice promennych typu int
int *p_a = NULL;   // Definice ukazatele na promennou typu int

p_a = &a;          // Do ukazatele p_a priradime adresu promenne a

b = *p_a;          // Do promenne b se priradi hodnota promenne,
                   // ktera se nachazi na adrese zapsane v p_a
                   // coz je promenna a, do b se tedy priradi 7
```

a	b	p_a
7	0	NULL
7	0	1004
7	7	1004

Práce s ukazateli

- Na spojení dereferenčního operátoru * se jménem ukazatele nahlížíme, jako by to byla proměnná. Můžeme mu tedy přiřadit hodnotu, používat jej při předání hodnoty odkazované proměnné do volané funkce a podobně




```
int a = 7;
int *p_a = NULL;

p_a = &a;          // Do ukazatele p_a priradime
                   // adresu promenne a
*p_a = 8;          // Zpusobi, ze do promenne a se priradi 8

printf("%i %i", a, *p_a); // Vypise hodnotu a, take hodnotu
                          // z adresy na niz ukazuje p_a
                          // (coz je take a)
                          // V obou pripadech se vypise 8
```

a	p_a
7	NULL
7	1004
8	1004

Ukazatele jako parametry funkcí

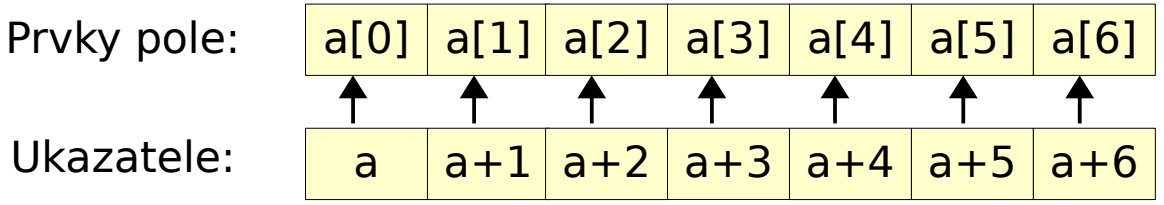
- Funkce v jazyce C mohou vrátit pouze jednu hodnotu
- Chceme-li z funkce vrátit více hodnot, používáme ukazatele
- Pro proměnné, do nichž chceme uložit výstupní hodnoty, předáme funkci jejich adresy (ukazatele). Uvnitř funkce tyto ukazatele dereferencujeme a přiřadíme jim potřebné hodnoty.
- Stejný mechanismus využívají funkce `scanf()` a `fscanf()`

```
// Funkce spocita soucet a rozdil prvnich dvou parametru (tj. a, b)
// a vysledky priradi do promennych, na nez ukazuji dva predavane
// ukazatele (tj. p_res1 a p_res2)
void soucet_rozdil(int a, int b, int *p_res1, int *p_res2)
{
  *p_res1 = a + b;
  *p_res2 = a - b;
}

int main()
{
  int x = 8, y = 3;
  int result1 = 0, result2 = 0;

  soucet_rozdil(x, y, &result1, &result2);
  printf("Soucet %i a %i je %i\n", x, y, result1);
  printf("Rozdil %i a %i je %i\n", x, y, result2);
}
```

Pole a ukazatele

- V jazyce C používáme obvykle jméno pole ve spojení s hranatými závorkami [], v nichž uvádíme index (tj. pořadí) prvku pole, s nímž chceme pracovat
- Pokud použijeme jméno pole bez uvedení hranatých závorek, je toto považováno za ukazatel na začátek pole, tj. ukazatel na první prvek pole
- Pokud k názvu pole přičteme číselnou hodnotu *n*, získáme ukazatel na *n*-tý prvek pole (tj. prvek pole s indexem *n*)

```
int n = 0;
int *p = NULL;
int a[6] = {4, 12, 9, 4, 25, 18};

p = a;      // Do ukazatele p se ulozi adresa zacatku pole
p = &a[0];  // Totez zapsane jinak

n = *a;       // Do n se ulozi hodnota prvku pole a[0] (tj. 4)
n = *(a + 2); // Do n se ulozi hodnota prvku pole a[2] (tj. 9)
```



Řetězce a ukazatele

- Řetězce jsou pole typu `char []` obsahující zakončovací znak \0
- Pro řetězce platí stejná pravidla jako pro pole, tj. jméno řetězcové proměnné použité bez [] je považováno za ukazatel na začátek pole (a při přičtení číselné hodnoty ukazuje na příslušný prvek pole)

```
char s[20] = "HELLO";
char *p_s = NULL;

p_s = s;     // Promenna p_s bude ukazovat na zacatek retezce s
printf("Retezec: %s\n", p_s);   // Vypise HELLO

p_s = s+2;   // Promenna p_s bude ukazovat na s[2]
printf("Retezec: %s\n", p_s);   // Vypise LLO
```

Funkce pro manipulaci s textovými řetězci

- Pro zjednodušení práce s textovými řetězci jsou v jazyce C k dispozici různé základní funkce
- Chceme-li používat funkce pro práci s řetězci, je třeba vložit na začátek programu `#include <string.h>`
- Seznam nejdůležitějších funkcí (pouze jména):
  - strlen() – zjištění délky řetězce
  - strcpy(), strncpy() – kopírování řetězců
  - strcat(), strncat() – spojování dvou řetězců
  - strchr(), strrchr() – hledání znaku v řetězci
  - strstr(), strcasestr() – hledání podřetězce v řetězci
  - strcmp(), strncmp() – porovnání dvou řetězců
- Podrobnější informace ke každé funkci lze získat z manuálových stránek UNIXu (*man jmeno_funkce* případně *man 3 jmeno_funkce*). Seznam všech funkcí je v *man 3 string*

Zjištění délky řetězce

int strlen(const char *s);

- Funkce `strlen()` vrací délku řetězce (tj. počet znaků nacházejících se před zakončovacím znakem '\0')
- Funkce pro práci s řetězci používají při definici řetězce jako parametru zápis char *s místo char s[], z praktického hlediska je funkčnost obou zápisů stejná (POZOR, to platí jen v seznamech parametrů funkcí, ne však při definici proměnných!)
- Použití slova `const` znamená, že předávaný řetězec bude uvnitř funkce pouze čten, nebude do něj zapisováno

```
// Na zacatku musime vlozit #include <string.h>

char s[] = "Tady je nejaky text";
int length = 0;

length = strlen(s);
printf("Delka retezce je: %i\n", length);
```

Kopírování řetězce

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, int size);
```

- Funkce `strcpy()` kopíruje řetězec `src` (z angl. *source*) do `dest` (z angl. *destination*), kopírují se všechny znaky, dokud se nenarazí na '\0' (v řetězci `src`), nakonec se do `dest` zkopíruje i znak '\0'
- Funkce `strncpy()` pracuje podobně ale kopíruje se maximálně `size` znaků. Pokud je řetězec `src` kratší než `size`, zkopíruje se i znak '\0', v opačném případě se '\0' nezkopíruje a řetězec `dest` zůstane nezakončený! Proto po použití funkce `strncpy()` vždy přiřadíme '\0' do posledního znaku `dest`
- Návratová hodnota je ukazatel na řetězec `dest`
- POZOR: v praxi upřednostňujeme funkci `strncpy()`, protože v případě použití `strcpy()` hrozí riziko překročení mezí řetězce `dest`, pokud je délka `src` větší než paměť alokovaná pro `dest`

Kopírování řetězce - příklad

```
// Ukazka pouziti funkce strcpy()

char s1[] = "Tady je nejaky text";
char s2[100] = "";

// Do retezce s2 bude zkopirovan obsah retezce s1
strcpy(s2, s1);
```

```
// Ukazka pouziti funkce strncpy()

char s1[] = "Tady je nejaky text";
char s2[30] = "";

// Do retezce s2 bude zkopirovan obsah retezce s1,
// ale max. 29 znaku
strncpy(s2, s1, 29);

// Do posledniho znaku retezce s2 priradime '\0',
// jinak by mohl zustat nezakonceny
s2[29] = '\0';
```

Spojení řetězců

```
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, int size);
```

- Funkce `strcat()` přidá řetězec `src` na konec řetězce `dest`, a na konec přidá znak '\0' (znak '\0', který se původně nacházel na konci `dest`, je přepsán)
- Funkce `strncat()` pracuje podobně, ale z řetězce `src` přidá do `dst` maximálně `size` znaků + znak '\0' (tedy až `size` + 1 znaků)
- Návratová hodnota je ukazatel na řetězec `dest`
- Při použití těchto funkcí musíme dávat pozor, aby nedošlo k překročení mezí cílového řetězce

```
#define STR_SIZE 100

char s1[STR_SIZE] = "JEDNA ";
char s2[STR_SIZE] = "DVA ";
char s3[STR_SIZE] = "TRI CTYRI PET";

strcat(s1, s2); // Retezec s2 pridame na konec retezce s1
strncat(s1, s3, 3); // Dale do s1 pridame 3 znaky z retezce s3

printf("Retezec: %s", s1);  // Vypise: JEDNA DVA TRI
```

Nalezení znaku v řetězci

```
char *strchr(char *s, int c);
char *strrchr(char *s, int c);
```

- Funkce `strchr()` hledá znak c v řetězci s a vrátí ukazatel na jeho první výskyt. Pokud znak není nalezen, vrací NULL
- Funkce `strrchr()` pracuje podobně, ale vyhledává poslední výskyt znaku (tj. prohledává řetězec odzadu – reverzně)

```
char s[] = "FILE=crambin.pdb";
char *p_s = NULL;

// Hledame znak '=', je-li nalezen, pak je p_str ruzne od NULL
// a bude ukazovat na pozici znaku '='
p_s = strchr(s, '=');
if (p_s != NULL)
  printf("Jmeno souboru: %s\n", p_s+1);  // Vypise crambin.pdb
```

```
char s[] = "/home/uzivatel/data/crambin.pdb";
char *p_s = NULL;

// Hledame znak '/' odzadu, abychom ziskali jmeno souboru
// bez pocatecni adresarove cesty
p_s = strrchr(s, '/');
if (p_s != NULL)
  printf("Jmeno souboru: %s\n", p_s+1);  // Vypise crambin.pdb
```

Nalezení podřetězce v řetězci

char *strstr(const char *haystack, const char *needle);

- Funkce `strstr()` hledá první výskyt řetězce `needle` v řetězci `haystack` a vrátí ukazatel na příslušné místo v řetězci `haystack`. V případě neúspěchu vrací NULL
- Pokud potřebujeme, aby funkce nerozlišovala velká a malá písmena, použijeme `strcasestr()`

```
char s[] = "Jmeno proteinu je crambin";
char *p_s = NULL;

// V retezci s hledame podretzec "crambin",
// je-li nalezen pak je p_str ruzne od NULL
// a bude ukazovat na pozici, kde zacina hledany text
p_s = strstr(s, "crambin");
if (p_s != NULL) {
  printf("Slovo crambin bylo nalezeno\n");
  printf("Text od nalezene pozice: %s\n", p_s); // Vypise: crambin
}

// Hledame zda-li na zacatku retezce s je "Jmeno", pokud ano,
// bude vraceny ukazatel ukazovat na zacatek s a tedy bude roven s
p_s = strstr(s, "Jmeno");
if (p_s == s) {
  printf("Retezec zacina na Jmeno\n");
}
```

Porovnání dvou řetězců

```
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, int size);
```

- Funkce `strcmp()` porovnává řetězce s1 a s2
- Funkce `strncmp()` pracuje podobně, ale porovnává pouze prvních `size` znaků
- Funkce vrací 0, pokud jsou oba řetězce totožné, zápornou hodnotu, pokud s1 je lexikograficky menší než s2, jinak vrací kladnou hodnotu
- Funkce rozlišují malá a velká písmena (malá písmena jsou lexikograficky větší než velká)
- Pro porovnání řetězců bez rozlišení velkých a malých písmen lze použít funkce `strcasecmp()` a `strncasecmp()`

```
char s1[] = "ATOM1", s2[] = "ATOM2";

if (strcmp(s1, s2) == 0)  printf("Retezce jsou totozne\n");
else  printf("Retezce jsou rozdilne\n");

if (strncmp(s1, s2, 4) == 0) printf("Shoda v prvnich 4 znacich.\n");
else  printf("Retezce se lisi v prvnich 4 znacich\n");
```

Předání parametrů příkazového řádku do funkce `main()`

- Funkce `main()` v jazyce C existuje ve dvou verzích, jedna verze je bez parametrů, druhá verze s parametry `argc` a `argv`. Druhou verzi používáme, pokud chceme zpracovat parametry předané programu na příkazovém řádku
- Argument `argc` obsahuje počet parametrů předaných programu na příkazovém řádku, zvětšený o 1
- Proměnná `argv` je pole ukazatelů na řetězce, každý prvek pole odpovídá jednomu předanému parametru
- První hodnota v argv je vždy název a cesta programu tak, jak byly uvedeny na příkazovém řádku (proto je argc vždy >= 1)

```
// Program vypise pocet predavanych parametru a jejich seznam

int main(int argc, char *argv[])
{
  printf("Pocet predavanych parametru: %i\n", argc);
  for (int i = 0; i < argc; i++) {
    printf("Parametr cislo %i je: %s\n", i, argv[i]);
  }
  return 0;
}
```

Dodržujte následující pravidla

- Při definici proměnných nedávejte příliš mnoho proměnných na jeden řádek. Raději definujte každou proměnnou na samostatném řádku (pak je lze snadněji okomentovat). Na jednom řádku definujte pouze proměnné, které spolu souvisí a lze je opatřit jednotným komentářem.
- Používejte komentáře:
  - Na začátek programu umístěte stručný komentář popisující činnost programu
  - Před každou funkci umístěte komentář popisující její účel, význam předávaných parametrů a význam návratové hodnoty
  - Všechny proměnné (globální i lokální) opatřete komentářem, pokud není z názvu proměnné na první pohled zřejmé, k čemu slouží. Proměnné `i`, `j`, `k`, používejte hlavně jako řídící proměnné cyklu (pro jiné účely tyto proměnné raději nepoužívejte), pak je nemusíte opatřovat komentářem.
- Vyzkoušejte kompilaci pomocí `gcc -O2 -Wall -Wextra -o prg nazev.c`, kompilátor provede důkladnější analýzu programu a upozorní vás na různé potenciální problémy.
- K testování přítomnosti argumentů na příkazovém řádku vždy kontrolujte hodnotu proměnné `argc`, nikdy např. neporovnávejte prvky `argv` s NULL.

Úlohy – část 1

1. Upravte program pro řešení kvadratické rovnice (cv. 1, úloha 1) tak, aby kořeny byly spočítány v samostatné funkci volané z `main()`. Do funkce budou předány parametry kvadratické rovnice a, b, c a pomocí ukazatelů budou vráceny hodnoty dvou kořenů. Návratovou hodnotou funkce bude počet nalezených kořenů. **1 bod**

2. Vytvořte jednoduchý program, který bude obsahovat funkci na záměnu hodnot dvou celočíselných proměnných. Program nejdříve načte od uživatele dvě celá čísla do dvou proměnných. Potom zavolá funkci, která prohodí hodnoty těchto proměnných (obě proměnné do ní budou předány jako ukazatele). Nakonec budou vypsány hodnoty obou proměnných. **1 bod**

Úlohy – část 2

3. Vytvořte program pro procvičení práce s řetězci. Program načte od uživatele řetězec. Poté předá řetězec do samostatné funkce, která provede následující:
   - Zjistí délku předaného řetězce a vypíše ji na obrazovku
   - Zkopíruje první 4 znaky do jiného pomocného řetězce s1 a dále zkopíruje 6 znaků od pozice 10. znaku (číslováno od 0) do řetězce s2. Na obrazovku vypíše oba řetězce.
   - Porovná první 4 znaky obou řetězců s1 a s2 a vypíše informaci, zdali se shodují, nebo ne.
   - Řetězce s1 a s2 se spojí do jednoho řetězce a to tak, že se nejdříve zkopíruje obsah s1 do pomocného řetězce s3 a pak se k němu přidá řetězec s2. Výsledný řetězec s3 se vypíše na obrazovku.
   - Určí se délka řetězce s3 a vypíše se na obrazovku.

   Nakonec implementujte do programu možnost zadat vstupní řetězec jako parametr na příkazovém řádku. Pokud uživatel nezadá řetězec na příkazovém řádku, tak si program vyžádá jeho zadání přímo (načte jej pomocí `scanf()`). **2 body**

Úlohy – část 3

4. Upravte program pro načítání zjednodušeného PDB souboru z předchozího cvičení (cv. 6, úloha 3). Do programu přidejte funkci, která na obrazovku vypíše seznam residuí. Na každém řádku bude vždy uvedena zkratka residua, číslo residua, název a číslo prvního atomu residua (tj. číslo uvedené v PDB souboru u atomu, který je v souboru uveden jako první pro dané číslo residua). Program musí správně pracovat i v případě, že čísla residuí v PDB souboru netvoří souvislou posloupnost (tedy například za residuem číslo 1 může následovat residuum číslo 3). První residuum v souboru také nemusí mít číslo 1.

1 bod