Programování v jazyce C pro chemiky
(C2160)

8. Načítání a zápis PDB souboru

Načtení řádku ze souboru

```
char *fgets(char *str, int size, FILE *stream);
```

- Funkce `fgets()` načítá jeden řádek ze souboru `stream` do řetězce `str`, dokud nenarazí na znak konce řádku nebo na konec souboru; do `str` se zapíše i znak konce řádku, pokud je načten; načítá se maximálně `size` - 1 znaků, nakonec se přidá '\0'
- Návratová hodnota funkce je v případě úspěchu ukazatel na `str`, v případě neúspěchu NULL

Formátovaný vstup z řetězce

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

- Funkce `sscanf()` pracuje podobně jako `scanf()` a `fscanf()`, ale načítá data z řetězce `str`
- Proměnné, do nichž se načtené hodnoty ukládají, předáváme prostřednictvím ukazatelů, tj. používáme & před jménem proměnných (kromě řetězců)
- Návratovou hodnotou funkce je počet úspěšně načtených položek, v případě neúspěchu vrací 0 nebo zápornou hodnotu

```
char s[] = "12  C  1.5  12.7  18.6";
int a = 0, n = 0;
char c = ' ';
float x = 0.0, y = 0.0, z = 0.0;

// Z retezce s se nactou hodnoty do prislusnych promennych,
// navratovou hodnotou je pocet uspesne nactenych polozek
n = sscanf(s, "%i %c %f %f %f", &a, &c, &x, &y, &z);
printf("Pocet nactenych hodnot: %i\n", n);

// Nasledujici prikaz vypise:
// 12, C, 1.500000, 12.700000, 18.600000
printf("%i, %c, %f, %f, %f\n", a, c, x, y, z);
```

Načítání textových souborů po řádcích

- Textové soubory často načítáme po řádcích pomocí funkce `fgets()` a tyto řádky pak zpracováváme pomocí `sscanf()` nebo pomocí funkcí pro práci s řetězci, popř. po jednotlivých znacích
- Tento přístup poskytuje větší flexibilitu a umožňuje zpracovat soubory, které mají na jednotlivých řádcích různé typy dat

```
// Ukazka nacteni souboru s ruznymi daty na ruznych radcich
#define BUF_SIZE 1000
char buf[BUF_SIZE] = "";
char str[50] = "", filename[100] = "";
int width = 0, height = 0;
float distance = 0.0;
// Zde otevreme soubor s identifikatorem f

while (feof(f) == 0)
{
  if (fgets(buf, BUF_SIZE, f) == NULL)
    break;
  sscanf(buf, " %49[^ =]", str);
  if (strcmp(str, "INPUT_FILE") == 0)
    sscanf(buf, " %*[^ =] = %99s", filename);
  else if (strcmp(str, "WINDOW_SIZE") == 0)
    sscanf(buf, " %*[^ =] = %i , %i", &width, &height);
  else if (strcmp(str, "DISTANCE") == 0)
    sscanf(buf, " %*[^ =] = %f", &distance);
}
```

```
INPUT_FILE = crambin.pdb
WINDOW_SIZE = 300, 500
DISTANCE = 3.5
```

Formátovaný výstup do řetězce

int sprintf(char *str, const char *format, ...);
int snprintf(char *str, int size, const char *format, ...);

- Funkce `sprintf()` pracuje podobně jako `printf()` a `fprintf()` ale zapisuje data do řetězce `str` (na konec vloží '\0')
- Funkce `snprintf()` funguje podobně, ale používá parametr `size`, který specifikuje maximální délku řetězce `str`, tj. maximální počet znaků zapsaných do `str` (vč. '\0')
- V praxi vždy upřednostňujeme `snprintf()`, protože při použití `sprintf()` hrozí riziko překročení mezí pro řetězec `str`
- Funkce vrací počet zapsaných znaků, v případě neúspěchu vrací zápornou hodnotu. Funkce `snprintf()` může vrátit hodnotu větší než `size`, viz. UNIXový manuál *man snprintf*

```
#define BUF_SIZE 1000
char buf[BUF_SIZE] = "";
char residue_name[] = "ASN";
int residue_number = 14;

snprintf(buf, BUF_SIZE, "residue_%.3i%s.pdb",
     residue_number, residue_name);
printf("Soubor: %s\n", buf); // Vypise: residue_014ASN.pdb
```

Formátovací řetězec - specifikace *šířky* a *přesnosti* pomocí parametrů

- Formátovací prvky ve formátovacích řetězcích mohou obsahovat *šířku* a/nebo *přesnost*, které uvádíme jako číselné hodnoty přímo ve formátovacím prvku (např. %4s, %.3f, %6.2f a pod.)
- Pokud místo číselné hodnoty *šířky* nebo *přesnosti* uvedeme hvězdičku *, budou tyto hodnoty určeny argumentem typu `int`, který musí v seznamu argumentů bezprostředně předcházet argument, který se má tisknout (odpovídající příslušné konverzi)
- Použijeme-li dvě hvězdičky pro *šířku* a *přesnost,* pak musí být v seznamu argumentů tyto dva argumenty uvedeny za sebou
- Toto se vztahuje na funkce `printf()`, `fprintf()`, `sprintf()` a `snprintf()`

```
char str[] = "Tady je nejaky text";
int width = 6, precision = 2;
float a = 56.14789;
// Vypise minimalne 6 znaku retezce s, stejne jako konverze %6s
printf("Text: %*s\n", width, str);
// Vypise cislo s presnosti 2, stejne jako %.2f
printf("Hodnota: %.*f\n", precision, a);
// Vypise cislo, min. 6 znaku s presnosti 2, stejne jako %6.2f
printf("Hodnota: %*.*f\n", width, precision, a);
```

Inicializace řetězce pomocí `memset()`

```
void *memset(void *s, int c, int n);
```

- Funkce `memset()` zapíše n znaků c na místo určené ukazatelem s
- Je deklarována ve `<string.h>`
- Funkce se často používá pro pro vyplnění řetězce zakončovacím znakem \0, zejména tehdy, pokud řetězec používáme pro práci s funkcemi, které zakončovací znak automaticky nevkládají

```
#define STRING_SIZE 100

char s[STRING_SIZE] = "";

// Nekdy je vhodne vsechny znaky retezce vyplnit zakoncovacim
// znakem '\0'
memset(s, '\0', STRING_SIZE);

// Do retezce take muzeme nastavit napr. same mezery
memset(s, ' ', STRING_SIZE);
// Pak je ale potreba zakoncit retezec znakem '\0'
s[STRING_SIZE-1] = '\0';
```

Formáty dat v textových souborech

- Textové soubory sloužící pro uložení dat (vědeckých dat, konfiguračních souborů a pod.) mají přesně daný formát, který umožňuje jejich správné načtení
- Formát souboru specifikuje jaký typ dat je v souboru uložen a jak jsou tato data v souboru rozmístěna a oddělena
- Rozlišujeme dva základní typy uspořádání dat v souborech:

  fixed format – data jsou umístěna na konkrétní pozici na řádku; je používán hlavně ve starších programech napsaných ve starších verzích FORTRANu

  Příklad: ATOM137HG12AVALB1.882-2.867-13.3630.333.91

  free format – pozice dat na řádku není striktně dána, je dáno pouze pořadí dat a oddělovač, který data odděluje (zpravidla mezera, tabelátor, čárka, středník, dvojtečka, =); tento formát je vhodný hlavně tam kde se předpokládá manuální editace dat uživatelem (nevadí mu např. nadbytečné mezery a pod.)

  Příklad: ATOMS_LIST = 5, 12, 13,151, 192, 220
  coordinates: 12.36 15.78 -25.3

PDB formát

- PDB formát je používán pro ukládání struktur biomolekul (proteinů, nukleových kyselin …) v PDB databázi www.pdb.org
- PDB formát je široce používaný v počítačové chemii a strukturní bioinformatice
- V PDB souborech jsou data zapsána stylem *fixed format*
- Dokumentace k PDB formátu je dostupná na: https://www.wwpdb.org/documentation/file-format

Soubor v PDB formátu

```
HEADER    PLANT PROTEIN                           10-SEP-01   1JXY
TITLE     CRAMBIN MIXED SEQUENCE FORM AT 220 K. PROTEIN/WATER
TITLE    2 SUBSTATES
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: CRAMBIN;
COMPND   3 CHAIN: A
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: CRAMBE HISPANICA SUBSP. ABYSSINICA;
REMARK   2 RESOLUTION.    0.89 ANGSTROMS.
REMARK   3
REMARK   3 REFINEMENT.
REMARK   3   PROGRAM     : PROLSQ
REMARK   3   AUTHORS     : KONNERT,HENDRICKSON
ATOM      1  N   THR A   1      16.858  14.053   3.518  1.00  8.84           N
ATOM      2  CA  THR A   1      16.901  12.798   4.316  1.00  6.25           C
ATOM      3  C   THR A   1      15.614  12.736   5.075  1.00  5.60           C
ATOM      4  O   THR A   1      15.139  13.784   5.543  1.00  7.89           O
ATOM      5  CB  THR A   1      18.071  12.793   5.272  1.00  7.53           C
ATOM      6  OG1 THR A   1      19.242  12.891   4.481  1.00 10.79           O
ATOM      7  CG2 THR A   1      18.159  11.531   6.124  1.00 10.28           C
ATOM      8  H1  THR A   1      17.272  13.943   2.691  1.00 10.25           H
ATOM      9  H2  THR A   1      15.955  14.297   3.365  1.00 15.73           H
TER     786      ASN A  46
HETATM  787  C1 AEOH A  66      15.823   0.920  12.835  0.50 21.97           C
HETATM  788  C1 BEOH A  66      15.532   0.742  12.262  0.50 26.82           C
CONECT   42  696
CONECT   52  572
CONECT  296  477
MASTER      296    0    1    2    2    0    1    6  791    1   12    4
END
```

Typy záznamů v PDB souboru

- Každý řádek PDB souboru obsahuje na začátku 6 znaků identifikujících typ dat na daném řádku
- Vybrané typy záznamů PDB formátu:

  ATOM - souřadnice atomů standardních residuí (tj. jedné z 20 základních aminokyselin nebo nukleotidů), obsahují také další informace charakterizující atom

  HETATM - podobné jako ATOM, ale pro atomy nestandardních residuí (např. ligandy, ionty, molekuly vody a pod.)

  CONECT - specifikuje vazby mezi atomy (zpravidla pouze pro nestandardní residua)

  REMARK - poznámka, rozlišuje se několik typů poznámek, každému přísluší jiná číslice uvedená za slovem REMARK

Struktura záznamu ATOM a HETATM

```
ATOM      7  CG2 THR A   1      18.159  11.531   6.124  1.00 10.28           C
ATOM  ····7··CG2·THR·A··1·····18.159··11.531···6.124··1.00·10.28·····-····--·C
1   5    10   15   20   25   30   35   40   45   50   55   60   65   70   75
```

1 - 6	Jméno záznamu "ATOM " nebo "HETATM" (6 znaků)
7 - 11	Pořadové číslo atomu (celé číslo)
12	Mezera
13 - 16	Jméno atomu (řetězec max. 4 znaky)
17	Alternativní pozice atomu (znak)
18 - 20	Jméno residua (řetězec max. 3 znaky)
21	Mezera
22	Identifikátor proteinového řetězce (znak)
23 - 26	Pořadové číslo residua v sekvenci (celé číslo)
27	Kód indikující vložení residua (znak)
28 – 30	Tři mezery
31 - 38	Souřadnice x v Å (reálné číslo)
39 - 46	Souřadnice y v Å (reálné číslo)
47 - 54	Souřadnice z v Å (reálné číslo)
55 - 60	Occupancy (reálné číslo)
61 - 66	Teplotní faktor (reálné číslo)
77 - 78	Symbol prvku (řetězec max. 2 znaky zarovnané doprava)
79 - 80	Formální náboj na atomu (2 znaky ve tvaru 2+, 1- a pod.)

Načtení PDB formátu

- PDB soubor obsahuje rozdílná data na jednotlivých řádcích, proto jej vždy načítáme po řádcích (funkcí `fgets()`) a podle jména záznamu uvedeného na začátku řádku (ATOM, HETATM ...) potom řádek rozebereme na jednotlivé prvky
- Data na každém řádku v PDB souboru mají uspořádání *fixed format*, proto pro jejich načítání nelze přímo použít funkce `fscanf()`, `sscanf()`
- Textové položky (např. jméno záznamu, atomu, residua, atd.) zkopírujeme pomocí `strncpy()` - kopírujeme daný počet znaků od příslušné pozice, nakonec musíme zapsat '\0'
- Znakové položky (identifikátor řetězce, atd.) – zkopírujeme přímo jeden znak z požadované pozice do výsledné proměnné
- Číselné položky (číslo atomu, číslo residua, souřadnice, atd.) - nejdříve pomocí `strncpy()` zkopírujeme daný počet znaků od příslušné pozice do pomocné řetězcové proměnné (a zakončíme '\0'). Potom pomocí `sscanf()` načteme příslušnou číselnou hodnotu z pomocného řetězce do cílové proměnné (při načítání celočíselných hodnot dáváme přednost konverzi %d před %i, aby bylo zaručeno, že bude číslo načítáno v desítkové soustavě i v případě, kdyby začínalo na 0)

Načtení PDB formátu - pokračování

- Pokud potřebujeme, aby byly textové položky zbaveny počátečních a koncových mezer (to je žádoucí např. u symbolu prvku), postupujeme podobně jako u načítání čísel, tj. nejdříve zkopírujeme příslušnou část do pomocného řetězce a pak pomocí `sscanf()` a konverze %s načteme řetězec do cílové řetězcové proměnné
- Hodnotu formálního náboje je vhodnější načíst jako řetězec, protože má nestandardní tvar se znaménkem uvedeným až za číslem: 2+, 1-
- Před načtením řádku pomocí `fgets()` je vhodné vyplnit řetězec znaky '\0' (pomocí `memset()`), abychom zabránili načítání neplatných dat v případě, že by byl řádek kratší, než očekáváme
- POZOR: pozice na kterých se data nacházejí jsou v PDB dokumentaci číslována od 1, ale jazyk C čísluje prvky pole od 0

Načtení PDB formátu - příklad

```
ATOM·····7··CG2·THR·A···1·····18.159···11.531····6.124···1.00·10.28··········C
1   5    10   15   20   25   30   35   40   45   50   55   60   65   70   75
```

```
char buf[BUF_SIZE] = "";
char s[30] = "";
FILE *f = NULL;

// Zde musí byt kod pro otevreni
// souboru s identifikatorem f

while (feof(f) == 0)
{
  memset(buf, '\0', BUF_SIZE);
  if (fgets(buf, BUF_SIZE, f) == NULL) break;
  if (strncmp(buf, "ATOM", 4) == 0 ||
      strncmp(buf, "HETATM", 6) == 0) {
    // Jmeno atomu - zkopirujeme 4 znaky od pozice 13
    strncpy(atoms[atoms_count].atom_name, buf+12, 4);
    atoms[atoms_count].atom_name[4] = '\0'; // Retezec musime zakoncit '\0'

    // Nacteni identif. proteinoveho retezce - zkopirujeme 1 znak z pozice 22
    atoms[atoms_count].chain_id = buf[21];

    // Ukazka nacteni cisla residua z pozice 23 – 26 (cislovano od 1)
    strncpy(s, buf+22, 4); // Do s kopirujeme 4 znaky od pozice 22 (cisl. od 0)
    s[4] = '\0';  // Retezec musime zakoncit '\0'
    sscanf(s, "%d", &atoms[atoms_count].residue_number);

    atoms_count++;
  }
}
```

```
// Zacatek programu
#define MAX_ATOMS 10000
#define BUF_SIZE 1000

typedef struct
{
  char atom_name[5];
  char chain_id;
  int residue_number;
  // Dalsi polozky
} ATOM;

ATOM atoms[MAX_ATOMS];
int atoms_count = 0;
```

Zápis PDB formátu

- Pro zápis PDB souboru používáme funkci `fprintf()` a příslušné formátování:

  Jméno záznamu – 6 znaků zarovnaných doleva (%-6.6s)
  Pořadové číslo atomu – 5 znaků zarovnaných doprava (%5d)
  Jméno atomu – 4 znaky (%-4.4s)
  Alternativní pozice atomu – 1 znak (%c)
  Jméno residua – 3 znaky zarovnané doleva (%-3.3s)
  Identifikátor proteinového řetězce – 1 znak (%c)
  Pořadové číslo residua – 4 znaky zarovnané doprava (%4d)
  Kód indikující vložení residua – 1 znak (%c)
  Souřadnice x, y, z – 8 znaků, přesnost na 3 deset. místa (%8.3f)
  Occupancy – 6 znaků, přesnost na 2 desetinná místa (%6.2f)
  Teplotní faktor – 6 znaků, přesnost na 2 desetinná místa (%6.2f)
  Symbol prvku – 2 znaky zarovnané doprava (%2.2s)
  Formální náboj na atomu – 2 znaky (%2.2s)
  Nesmíme zapomenout na mezery na pozicích 12 a 21, tři mezery na pozicich 28-30 a 10 mezer na pozicích 67 - 76 (číslováno od 1)

Dodržujte následující pravidla

- Jako globální definujte pouze ty proměnné, jejichž data sdílí větší počet funkcí. Pokud proměnnou sdílí malý počet funkcí, definujte ji jako lokální a předávejte do volaných funkcí jako parametr. Proměnné sloužící pouze pro účely jedné funkce definujte vždy jako lokální
- Názvy globálních proměnných volte tak, aby z nich byl jasný význam proměnné, volte raději delší názvy. Pro lokální proměnné lze používat kratší názvy

Úlohy

1. Upravte program pro načítání zjednodušeného PDB souboru (cv. 6, úloha 3, resp. použijte program ze cv. 7, úloha 4) tak aby uměl načítat a zapisovat skutečný PDB soubor (otestujte se souborem *crambin.pdb* nacházejícím se v adresáři */home/tootea/C2160/data/*). Jména vstupního a výstupního PDB souboru budou zadána jako parametry na příkazovém řádku. **3 body**
2. Program z úlohy 1 upravte tak, aby nejdříve načetl PDB soubor a potom zapsal několik PDB souborů, pro každé residuum jeden. Každý soubor se bude jmenovat podle čísla a názvu residua (např. *residue_014ASN.pdb*) a budou do něj zapsány pouze atomy daného residua. Jméno vstupního PDB souboru a (existujícího) cílového adresáře bude specifikováno na příkazovém řádku. **nepovinná, 3 body**