Programování v jazyce C pro chemiky
(C2160)

9. Datové typy, konverze, práce s PDB soubory

Celočíselné datové typy

- Hodnoty v proměnných mohou nabývat pouze určité velikosti, která je dána počtem bitů použitých pro uložení hodnoty v paměti
- Počet bitů použitý pro daný typ není v jazyce C přesně dán, je závislý na použitém hardwaru a překladači
- Pro uložení proměnných typu `int` bývá zpravidla použito 32 bitů, rozsah možných hodnot tak je tj. $-2^{31}-1$ až $2^{31}-1$, což je

  –2 147 483 648 až 2 147 483 647

- Rozsah hodnot dalších typů na současných počítačích (x86_64):

char	8 bit	–128 až 127
short (int)	16 bit	–32768 až 32767
long (int)	64 bit	$-9{,}2 \cdot 10^{18}$ až $9{,}2 \cdot 10^{18}$

- Garantováno je ale jen 16 bitů pro `int` a 32 bitů pro `long` (na Windows má `long` právě jen 32 bitů). Minimálně 64 bitů garantuje jen `long long int`
- Klíčové slovo `int` v názvech typů lze vynechat
- Přidáním klíčového slova `unsigned` lze získat bezznaménkovou variantu každého typu s dvojnásobným kladným rozsahem (např. `unsigned char` pro hodnoty 0 až 255)

Datové typy pro reálná čísla

- Kromě typu `float` je dostupný i typ `double` s dvojitou přesností
- Pro načítání do proměnné typu `double` (ve funkcích `scanf()` a pod.) musíme použít modifikátor `l` (malé L) tj. `%lf`
- Pro zápis proměnné `double` (ve funkcích `printf()` a pod.) používáme konverzi `%f` (bez modifikátoru `l`!)
- Pro vědecké výpočty vždy používáme místo `float` typ `double`, abychom snížili zaokrouhlovací chybu při aritmetických operacích
- Rozsah kladných hodnot reálných typů na současných počítačích:

float	32 bit	$1{,}2 \cdot 10^{-38}$ až $3{,}4 \cdot 10^{38}$	7 plat. číslic
double	64 bit	$2{,}2 \cdot 10^{-308}$ až $1{,}8 \cdot 10^{308}$	15 plat. číslic

- Hodnoty mezi nulou a uvedeným minimem se ukládají se sníženou přesností nebo zaokrouhlují na nulu
- Reálné datové typy dále umí uložit speciální hodnoty ±nekonečno a NaN (neplatná hodnota, například 0/0)

Typy char

- **char** je celočíselný typ, proměnné tohoto typu se tedy chovají podobně jako **int**, v paměti však zabírají pouze 8 bitů. Lze jim přiřadit hodnoty 0 až 127 (ve skutečnosti -128 až 127 nebo 0 až 255 v závislosti na překladači)
- Číselná hodnota proměnné typu **char** udává pořadí znaku v tabulce znaků (standardně se používá tabulka ASCII – zkratka z American Standard Code for Information Interchange)
- Znaky s hodnotou < 32 v ASCII tabulce se nazývají řídící znaky které se netisknou na obrazovku ale představují instrukce pro výstupní terminál (např. přechod na nový řádek, tabulátor, atd.)
- Hodnoty < 0 či > 127 se na znaky překládají dle kódování daného použitým operačním systémem (dnes typicky UTF-8 Unicode, dříve ISO8859-2 či Windows CP1250)

ASCII tabulka

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Konstanty

- Konstantou v jazyce C označujeme číselnou, znakovou nebo textovou hodnotu zapsanou v textu programu
- Celočíselné konstanty zapisujeme v jedné ze tří číselných soustav:
  - desítkové (dekadické), např.: 16, 47
  - osmičkové (oktalové) – jejich zápis začíná nulou, nesmí se v nich použít číslice 8 nebo 9, např.: 0644, 0100
  - šestnáctkové (hexadecimální) – začínají znaky 0x, případně 0X, kromě číslic se v nich užívají naky A-F, např.: 0x12, 0xB7, 0xAC
- Reálná konstanta vždy obsahuje desetinnou tečku, lze ji zapsat v přímém tvaru (15.4788, .458, -1.) nebo semilogaritmickém tvaru např. 23.45e6 (což znamená $23.45 * 10^6$), .09E34, 2e-5
- Zápis číselných konstant nesmí obsahovat mezeru
- Příklad konstant:

celočíselné:	5, 7, 0x83
reálné:	3.45, 23.45E6, 2e-5
znakové:	'A', 'k', '\n'
řetězcové:	"Nejaky text"

Řídící znaky

- Znakové konstanty jsou tvořeny znakem uvedeným v apostrofech, např.: 'A', 'g', '5'
- Kromě toho existují řídící ("neviditelné") znaky, které zapisujeme pomocí zpětného lomítka a odpovídajícího písmene
- Řídící znaky můžeme použít ve znakových konstantách (např. '\n', '\0', '\t', '\'') nebo řetězcových konstantách ("Bude nasledovat tabulator \t, potom \"text v uvozovkach\", zpetne lomitko \\ a konec radku\n")

\a	Zvukový signál, písknutí, *alert*	BELL
\n	Nový řádek, *line feed*	LF
\r	Návrat na začátek řádku, *carriage return*	CR
\t	Horizontální tabulátor, *horizontal tab*	HT
\\	Zpětné lomítko, *backslash*	\
\'	Apostrof, *single quote*	'
\"	Uvozovky, *double quote*	"
\0	Nulový znak, *null character*	NULL

Logický (Booleovský) datový typ

- Jazyk C před příchodem C99 neměl logický datový typ, proto se ve starším kódu místo toho používá typ **int**, za nepravdivou je typicky považována hodnota 0, za pravdivou je považována jakákoliv hodnota různá od 0
- Počínaje **C99** je k dispozici typ `bool` a konstanty `true` a `false`, definované v hlavičkovém souboru `stdbool.h`
- Nechceme-li použít stdbool.h, je tentýž logický datový typ k dispozici pod názvem `_Bool` a místo `true a false` můžeme používat `1 a 0`
- Výhodou bool proti int je, že hodnoty lze spolehlivě porovnávat

```
int a = 1, b = 2; // tyto hodnoty bychom mohli dostat z ruznych funkci

if (a && b) {     // zde diky && dojde k interpretaci logickych hodnot
    printf("a i b jsou pravda\n");
}
if (a == b) {     // tady ale probehne obycejne ciselne porovnani
    printf("tak proc se toto nevytiskne?\n");
}

bool ba = a, bb = b; // bool ma jen jednu moznou hodnotu pro true,
if (ba == bb) {      // takze jej lze snadno porovnavat
    printf("slava, s bool to funguje!\n");
}
```

Typová konverze

- Jazyk C umožňuje při provádění aritmetických operací kombinovat proměnné (popř. konstanty) různých typů (int, float, double)
- Procesory jsou však schopny provádět aritmetické operace pouze s čísly stejného typu
- Překladač musí převést (konvertovat) proměnné v aritmetickém výrazu na stejný typ
- Konverze se provádí automaticky při použití rozdílných typů v následujících případech:
  - v aritmetických výrazech
  - při přiřazení hodnoty proměnné, pokud typ proměnné a přiřazovaná hodnota jsou odlišného typu
  - při předávání hodnoty volané funkci
  - při vracení hodnoty funkce příkazem **return**
- Pokud přiřazujeme reálnou hodnotu do celočíselné, je hodnota konvertována tak, že se ořízne desetinná část, tj. nedochází k zaokrouhlení podle standardních pravidel! (např.: `int n = 3.7;` v proměnné n bude hodnota 3)
- Při kombinaci celočíselných proměnných s reálnými v aritmetických výrazech se celočíselné konvertují na reálné

Celočíselné vs. neceločíselné dělení

- Operátor dělení / provádí jeden ze dvou typů dělení:
- Celočíslené dělení – pokud jsou oba operandy celá čísla (tj. proměnné typu int nebo celočíselné konstanty). Při celočíselném dělení je vráceno celé číslo, zbytek po celočíselném dělení lze získat pomocí operátoru modulo %
- Reálné dělení je prováděno, pokud je minimálně jeden z operandů reálné číslo (tj. proměnné typu float, double nebo reálná konstanta). Při reálném dělení je vráceno reálné číslo
- Chceme-li při dělení celočíselných hodnot zajistit aby došlo k reálnému dělení, musíme jednu z hodnot přetypovat na reálný typ

```
int a = 7, b = 2, c = 0;
double x = 2.25, y = 3.14, z = 0.0;
// Priklady celociselneho deleni
c = a / b;   // Vysledkem bude 3, zbytek po deleni tj. a % b by byl 1
c = 17 / a;  // Vysledek bude 2, zbytek by byl 3
c = 23 / 10; // Vysledek bude 2, zbytek by byl 3
// Priklady realneho deleni
z = x / y;     // Vysledek bude priblizne 0.71656
z = a / y;     // Druhy z operandu (tj. y) je realna hodnota
z = 45.2 / y;  // Oba operandy maji realnou hodnotu
z = x / 20;    // Prvni operand ma realnou hodnotu
z = a / 2.0;   // Druhy operand ma realnou hodnotu
z = a / (double) b;  // Druhy operand jsme pretypovali na realny
z = (double) a / (double) b;  // Muzeme pretypovat i oba operandy
```

Rozšířený přiřazovací operátor

- Rozšířený přiřazovací operátor vzniklý spojením aritmetického a přiřazovacího operátoru:
  - a += b; je totéž jako a = a + b;
  - a -= b; je totéž jako a = a - b;
  - a *= b; je totéž jako a = a * b;
  - a /= b; je totéž jako a = a / b;
  - a %= b; je totéž jako a = a % b;
- POZOR: příslušná operace je provedena s celým výrazem na pravé straně:
  - a /= b + 5; **neznamená** a = a / b + 5; ale a = a / (b + 5);

Bitové operátory

- Bitové operátory zpracovávají každý bit operandu zvlášť
- Bitové operátory

		1 1	1 0	0 1	0 0
&	bitový součin	1	0	0	0
|	bitový součet	1	1	1	0
^	bitový exkluzivní součet	0	1	1	0
~	bitová negace				
<<	bitový posun doleva				
>>	bitový posun doprava				

- Příklady:

  21 & 56 = 00010101 & 00111000 = 00010000
  21 | 56 = 00010101 | 00111000 = 00111101
  21 ^ 56 = 00010101 ^ 00111000 = 00101101
  ~21 = ~00010101 = 11101010
  21 << 2 = 00010101 << 2 = 01010100
  21 >> 1 = 00010101 >> 2 = 00001010

- Bitové operátory lze aplikovat pouze na hodnoty celočíselného typu nebo typu **char**
- Lze použít i zkrácené operátory **&=**, **|=**, **^=**, **<<=**, **>>=**

Priorita operátorů

Priorita operátorů od nejvyšší po nejnižší:

priorita	operátory	směr vyhodnocování	pozn.
1	() [] -> .	zleva doprava	1)
2	! ~ ++ -- - * & přetypování, sizeof	zprava doleva	2) 3) 4)
3	* / %	zleva doprava	
4	+ -	zleva doprava	
5	<< >>	zleva doprava	
6	< <= > >=	zleva doprava	
7	== !=	zleva doprava	
8	&	zleva doprava	
9	^	zleva doprava	
10	|	zleva doprava	
11	&&	zleva doprava	
12	||	zleva doprava	
13	?:	zprava doleva	
14	= += -= *= /= atd.	zprava doleva	
15	,	zleva doprava	

1) Závorky mají nejvyšší prioritu, používají se pro upřesnění priority

2) **&** a * jsou v tomto případě referenční a dereferenční operátory

3) **–** zde znamená unární minus (mění znaménko operandu)

4) Pro ++ a - toto platí pokud jsou použity jako prefixové operátory

Operátor podmíněného výrazu

- Operátor podmíněného výrazu obsahuje tři operandy (tzv. ternární operátor) a dva znaky

  logicky_vyraz **?** vyraz1 **:** vyraz2

- Logický výraz se vytváří podobně jako u podmínek **if**, zpravidla ho uzavíráme do kulatých závorek
- Je-li hodnota logického výrazu pravdivá, je výsledná hodnota určena vyhodnocením *vyraz1*, v opačném případě je určena vyhodnocením *vyraz2*
- Příkaz se používá pro konstrukci jednoduchých podmínek
- Výhodou oproti použití příkazu **if** je kratší forma zápisu, v praxi je však často přehlednější použití **if**

```
int a = 5, absolutni_hodnota = 0;
absolutni_hodnota = (a >= 0) ? a : -a;

//Totez s pouzitim klasicke podminky

if (a >= 0)
  absolutni_hodnota = a;
else
  absolutni_hodnota = -a;
```

```
int i = 2, j = 8;
printf("Vetsi hodnota: %i\n",
            (i > j) ? i : j);

//Totez s pouzitim klasicke podminky

if (i > j)
  printf("Vetsi hodnota: %i\n", i);
else
  printf("Vetsi hodnota: %i\n", j);
```

Prázdný příkaz

- Prázdný příkaz je tvořen samostatným středníkem
- V praxi se používá málo
- Příklad:

```
while (is_message() == 0)
   ;
```

  cyklus běží stále dokola, dokud nevrátí funkce `is_message()` pravdivou hodnotu
- Častou chybou je neúmyslné použití prázdného příkazu v cyklech a podmínkách

```
// Zamysleny kod
int i = 0, a = 1;

for(i = 0; i < 10; i++)
  a += 2;

printf("%i", a); // Vypise 21


if (a > 1)
  printf("Je vetsi nez 1\n");
```

```
// Chybny kod
int a = 1;

// Neumyslne pouzity strednik:
for(i = 0; i < 10; i++);
  a += 2;

printf("%i", a); // Vypise 3


if (a > 1);    //Neumyslny strednik
  printf("Je vetsi nez 1\n");
// Uvedeny text se vypise vzdy bez
// ohledu na hodnotu a
```

Konverze %n

- Ve funkci `sscanf()` lze použít konverzi %n, která zapíše počet dosud načtených znaků do proměnné typu `int` (ta musí být předána jako ukazatel, tj. použijeme &)
- Tuto konverzi používáme, pokud chceme provést další načítání od pozice, kde jsme naposledy skončili (např. pokud neznáme počet načítaných prvků)

```
char s[] = "12 458 30 2 78 98";  // Z retezce budeme nacitat hodnoty
int value = 0;    // Do teto promenne budeme nacitat hodnoty
int nitems = 0;   // pocet polozek nactenych funckci sscanf()
int nchars = 0;
int delta = 0;
char s[] = "12 458 30 2 78 98";

while (1) { // Nekonecny cyklus (dokud z nej nevyskocime pomoci break)
  nitems = sscanf(s+delta, "%d %n", &value, &nchars);
  if (nitems < 1)  // Nebyla-li nactena ani 1 polozka vyskocime z cyklu
    break;
  printf("%i, ", value);
  delta += nchars;
}
printf("\n");
```

Konverze %n - příklad

```
// Program demonstruje nacitani hodnot, ktere jsou oddeleny carkou,
// ale mohou byt tez zapsany jako interval hodnot
char s[] = "12, 15-20,30, 40 - 45, 70 , 98";
int value = 0, value_previous = 0;
int nitems = 0, nchars = 0;
int delta = 0;
char c = ' ', c_previous = ',';

while (1) {
  nitems = sscanf(s+delta, "%d %c %n", &value, &c, &nchars);
  if (nitems < 1) {
    break;
  }

  if ((c_previous == ',') && (c != '-')) {
    printf("Samostatna hodnota: %i\n", value);
  } else if (c_previous == '-') {
    printf("Interval hodnot: %i - %i\n", value_previous, value);
  }

  value_previous = value;
  c_previous = c;
  delta += nchars;
}
```

Dodržujte následující pravidla

- Pro reálné proměnné používejte typ `double` místo `float`. Nezapomeňte změnit konverze `%f` na `%lf` (malé L) ve volání `scanf()`, ne však `printf()`
- Pokud v podmínkách `if` používáte složitější výrazy, nespoléhejte se na prioritu operátorů, ale používejte závorky. Podmínka bude přehlednější a sníží se tak pravděpodobnost chyby způsobené nepozorností nebo špatnou znalostí priorit operátorů
- Ačkoli se v jednotlivých úlohách upravuje stále tentýž program, odevzdejte řešení každé úlohy v samostatném souboru. Každý odevzdaný program musí vypisovat jen to, co je požadováno v zadání dané úlohy

Úlohy – část 1

1. Rozšiřte program pro načítání PDB souboru (cv. 8, úloha 1) tak, že do něj implementujete funkci, která bude obsahovat pole residuí, tj. pole struktur RESIDUE (velikost pole zvolte 2000). Struktura RESIDUE bude obsahovat dvě celočíselné proměnné (např. *first_atom* a *last_atom*). Těmto proměnným přiřaďte index prvního a posledního atomu residua. Do struktury RESIDUE dále přidejte proměnné obsahující číslo residua (jak je uvedeno v PDB souboru) a název residua. Na konci funkce vypíše seznam residuí, pro každé residuum vypíše číslo a název residua a index prvního a posledního atomu residua (Začátek by měl vypadat jako ve žlutém rámečku níže). Pro testování použijte soubor *crambin_noal.pdb* nacházející se v adresáři */home/tootea/C2160/data/* **3 body**

```
Residuum: 1 THR, atomy: 0 - 14
Residuum: 2 THR, atomy: 15 - 27
Residuum: 3 CYS, atomy: 28 - 37
Residuum: 4 CYS, atomy: 38 - 47
Residuum: 5 PRO, atomy: 48 - 61
....
```

Úlohy – část 2

2. Do programu z předchozí úlohy implementujte funkci která spočítá pro každé residuum geometrický střed jeho atomů a vypíše tuto hodnotu na obrazovku (společně se jménem a číslem residua). Funkci dále modifikujte tak, aby se pro výpočet použily pouze nevodíkové atomy. Začátek výpisu (pro nevodíkové atomy) bude vypadat tak, jak je uvedeno ve žlutém rámečku. **2 body**

```
Residuum 1 THR, stred: 17.14, 12.94, 4.90
Residuum 2 THR, stred: 13.78, 10.69, 5.66
Residuum 3 CYS, stred: 13.46, 11.24, 9.80
Residuum 4 CYS, stred: 10.83, 8.46, 11.34
Residuum 5 PRO, stred: 8.85, 8.91, 14.73
```

3. Do programu z předchozí úlohy implementujte funkci, která vyhledá pro každé residuum atomy, které tvoří peptidovou páteř proteinu (jmenují se `" N  "`, `" CA "`, `" C  "`, `" O  "`, je třeba respektovat mezery v názvech). Pro každé residuum vypište čísla těchto atomů (tak, jak jsou uvedena v PDB souboru) společně se jménem a číslem residua. **2 body**

Úlohy – část 3

4. Do předchozího programu přidejte funkci, která vyhledá všechny dvojice residuí, jejichž vzdálenost C-alpha (tj. CA) atomů je větší než 5 a zároveň menší než 8 Å.

   nepovinná, 3 body

5. Do předchozího programu přidejte funkci, která načte konfigurační soubor uvedený níže (ve žlutém rámečku). Jméno konfiguračního souboru bude specifikováno na příkazovém řádku. Řádky v souboru mohou být uvedeny v libovolném pořadí. Program nejdříve načte konfigurační soubor a pak načte PDB soubor, přičemž jako jméno PDB souboru se použije jméno uvedené na řádku INPUT_FILE. Ze souboru se načte také seznam residuí uvedený v RESIDUE_LIST (čísla odpovídají hodnotám uvedeným v PDB souboru!). Do výstupního PDB souboru se zapíší pouze ta residua, která jsou uvedena v tomto seznamu.

   nepovinná, 2 body

```
INPUT_FILE=/home/username/data/filename.pdb
WINDOW_SIZE = 300, 500
DISTANCE = 3.5
RESIDUE_LIST=3,4,7 ,15,16, 17,30, 34,40,45
```