Znaky

Znakový datový typ v C

- Neexistuje, jednotlivé znaky v paměti = celá čísla dle ASCII – jedno zda v `char, int, signed` či `unsigned` atd.
- pouze ASCII kódování
- Pouze pro I/O lze vynutit práci se znaky:

```
unsigned char c;

scanf("%d", &c);  // zadam 9, hodnota promenne = 9
scanf("%c", &c);  // zadam 9, hodnota promenne = 57
printf("%d", c);  // tisk 57
printf("%c", c);  // tisk 9
```

jen idea, nelze!!!

- lze přímo provádět aritmetické operace – převod malé abecedy na velkou

```
c = c - 32        // ASCII (a) = 97, ASCII (A) = 65
```

Literály

- vždy uzavřeny mezi apostrofy`''`
- hodnota konstanty odvozena od odpovídajícího čísla v ASCII tabulce

Možnosti zápisu

- přímý znak

```
'4', 'a', '*'
```

- `'\0xHH'` – ASCII hodnota v hexa soustavě

```
'0xA'       // NewLine
'0x41'      // = 'A'
```

Konstanta uvozená `\` = **escape sekvence**

- znakové escape sekvence

	ASCII	Název	Význam
`\0`	0x00	Nul	Nula
`\a`	0x07	Alert (Bell)	pípnutí
`\b`	0x08	Backspace	návrat o jeden znak zpět
`\f`	0x0C	Formfeed	nová stránka nebo obrazovka
`\n`	0x10	Newline	přesun na začátek nového řádku
`\r`	0x0D	Carriage return	přesun na začátek aktuálního řádku
`\t`	0x09	Horizontal tab	přesun na následující tabelační pozici
`\\`	0x5C	Backslash	obrácené lomítko
`\'`	0x2C	Single quote	apostrof
`\"`	0x22	Double quote	uvozovky

Příklady:

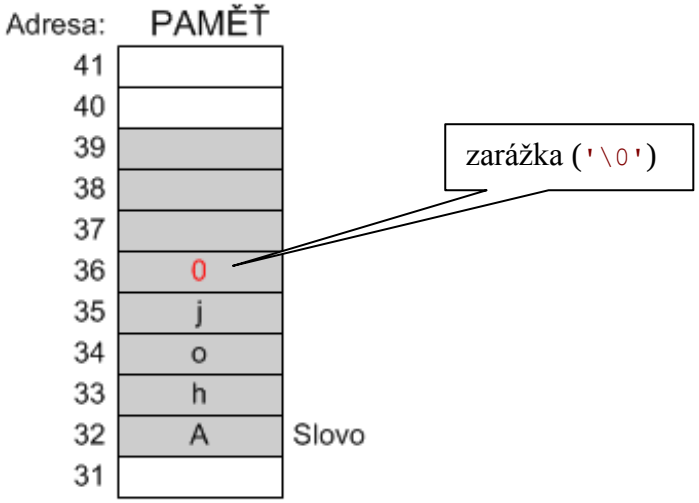
Převod malé abecedy na velkou

```
c = c - ('a'-'A');      // zavorky kolem a-A zajisti jako první vypocet 32
```

Řetězce (strings)

- = jednorozměrné pole prvků typu `char`, zakončené zarážkou; určuje aktuální délku řetězce
- Příklad:

`char Slovo[8];` obsah = `"Ahoj"`



- manipulaci se zarážkou v C automaticky (v drtivé většině případů)

Literály

- `"Ahoj"`
- platí vše, co pro znakové konstanty (escape sekvence, spec. symboly, atd.)
- pozor:
  - `"Ahoj"` = délka 5 B
  - `"A"` = 2 B
  - `'A'` = 1 B

Definice proměnné

- jako pole prvků typu `char`

```
#define MAX_DELKA 50
char Slovo[MAX_DELKA];
```

  - maximální počet znaků = `MAX_DELKA - 1`
  - index posledního znaku = `MAX_DELKA - 2`
- deklarace s inicializací – několik možností

```
char Slovo[MAX_DELKA] = "Ahoj";     // překladač doplní zarážku
char Slovo[] = "Ahoj";  // překladač doplní zarážku a určí délku pole
char Slovo[] = {'A','h','o','j',0};
char Slovo[] = {'A','h','o','j','\0'};
```

Vstup a výstup řetězce

- %s

- existuje několik různých funkcí (viz string.h)

Vstup řetězce z klávesnice

- lze použít `scanf()`:

```
char Slovo[MAX_DELKA];
scanf("%s", Slovo);
```

  - načítání se zastaví na prvním bílém znaku → po zadání `Ahoj mami` bude v paměti pouze `'A','h','o','j',0`

Výstup řetězce na obrazovku

- lze použít `printf()`:

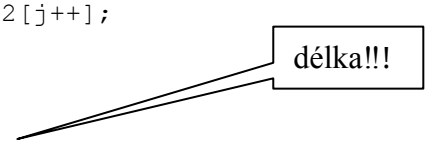
```
char Slovo1[] = "Ahoj";
char Slovo2[] = "mami";
printf("%s %s.", Slovo1, Slovo2);
```

  - `printf()` tiskne znaky až po `'\0'`

Práce s řetězcem

- možnosti
  - „ručně“
  - string.h
- práce s řetězcem jako s celkem → vždy až po `'\0'`
- příklad: Napište funkci `Append()`, která připojí na konec řetězce `s1` řetězec `s2`

```
#include <stdio.h>
#define MAX_DELKA 50
void Append(char s1[], char s2[])
{
	int i=0, j=0;
	// nalezeni konce s1
	while (s1[i] != '\0')
		i++;
	// kopie s2 na konec s1
	while (s2[j] != '\0')
		s1[i++] = s2[j++];
}
int main(void)
{
	char Slovo1[MAX_DELKA] = "Ahoj ";
	char Slovo2[] = "mami";
	Append(Slovo1, Slovo2);
	return 0;
}
```



Struktury

- struktura = datový typ:
  - strukturovaný (obsahuje více prvků)

- **heterogenní** → prvky (položky) libovolných **různých** datových typů
- realizace zapouzdření

Definice

- syntaxe obecně:

```
struct Tag {
      typ item1;
      typ item2;
            |
      typ itemN;
} identifikator;
```

nepovinná deklarace nového datového typu

nepovinná definice konkrétní proměnné (lze i později s využitím `Tag`)

- 1. nezávislá deklarace struktury a definice proměnných

```
struct Osoba {
      char jmeno[100];
      int vek;
};
int main(void)
{
      struct Osoba ja;
      struct Osoba ty, on;
      return 0;
}
```

  - nelze pouze `Osoba ty, on; // ale v C++ OK`
  - deklarace uvnitř `main()` → nelze využít ve funkcích

- 2. nezávislá deklarace struktury přes `typedef` a definice proměnných (přednostně)

```
#include <stdio.h>
typedef struct {
      char jmeno[100];
      int vek;
} tOsoba;
int main(void)
{
      tOsoba ja;
      tOsoba ty, on;
      return 0;
}
```

- 3. (viz. později spojový seznam)

Práce se strukturou

- práce s prvky struktury

```
identifikatorStruktury.identifikatorPolozky
```

- Příklad:

```
tOsoba pracovnik;
scanf("%d", &pracovnik.vek);
pracovnik.vek += 20;    // je to hornik
printf("Jmeno: %s\n", pracovnik.jmeno);
}
```

- práce se strukturou jako celkem – lze

```
tOsoba pracovnik, jinyPracovnik;
jinyPracovnik = pracovnik;
```

prvkem struktury je pole → přesto funkční!!!

  - matematické, logické, relační atd. operátory nejsou definovány!!!

inicializace struktur

```
tOsoba nekdo = {"Hilda Novakova", 10000}
```

Pointer na strukturu

```
tOsoba ja, *pJa;
pJa = &Ja;         // adresa Ja do pJa
(*pJa).Vek = 10;   // zavorky nutne!!!
pJa->Vek = 10;     // totez co (*pJa).Vek
```

Struktury a funkce

Struktura jako parametr funkce

- volání hodnotou i odkazem
- způsob definice formálního parametru závisí na způsobu definice struktury

```
struct tOsoba {
      // prvky
};
void Tisk(struct tOsoba Ktera)
```

```
typedef struct {
      // prvky
} tOsoba;
void Tisk(tOsoba Ktera)
```

- struktura = velký objem dat → přednostně volání odkazem

Struktura jako návratová hodnota funkce

- lze, avšak přednostně odkazem

Dynamická alokace struktur

- transparentní → pozor na memory alignment – zásadně přes:

```
sizeof(struct Tag)
sizeof(typedefStruct)
sizeof(identifStruct)
```

Příklad

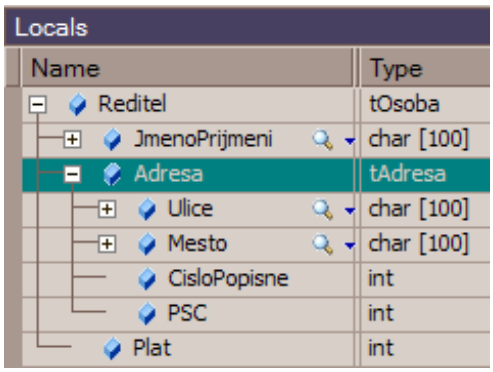
- napište funkce pro
  - naplnění struktury `tOsoba` hodnotami zadanými z klávesnice (vrátí strukturu),
  - tisk proměnné typu `tOsoba` na obrazovku,

- naplnění struktury `tOsoba` hodnotami zadanými z klávesnice (přes volání „odkazem“)

```
#include <stdio.h>
#define MAX_DELKA 50

typedef struct {
	char jmeno[MAX_DELKA];
	int plat;
} tOsoba;

void TiskOsoby(tOsoba ktera);
tOsoba NactiOsobu(void);
void NactiOsobuOdkazem(tOsoba *pOsoba);

int main(void)
{
	tOsoba osoba;
	osoba = NactiOsobu();
	TiskOsoby(osoba);
	NactiOsobuOdkazem(&osoba);
	return 0;
}
void TiskOsoby(tOsoba ktera)
{
	printf("%s\n", ktera.jmeno);
	printf("%d\n", ktera.plat);
}
tOsoba NactiOsobu(void)
{
	tOsoba pom;
	scanf("%s", &pom.jmeno);
	scanf("%d", &pom.plat);
	while (getchar() != '\n');    // vyprazdneni buff klavesnice
	return pom;
}
void NactiOsobuOdkazem(tOsoba *pOsoba)
{
	scanf("%s", pOsoba->jmeno);   // bez &
	scanf("%d", &(pOsoba->plat)); // zavorky nutne, priorita operatoru
	while (getchar() != '\n');    // vyprazdneni buff klavesnice
}
```

Struktura uvnitř struktur

```
#include <stdio.h>
// definice struktury

typedef struct {
	char ulice[100];
	char mesto[100];
	int cisloPopisne, PSC;
} tAdresa;

typedef struct {
	char jmeno [100];
	tAdresa adresa;
	int plat;
} tOsoba;

void TiskOdkazem(tOsoba *ktera);

int main(void)
{
	tOsoba reditel;
	scanf("%d", &(reditel.adresa.PSC));
	TiskOdkazem(&Reditel);
	return 0;
}

void TiskOdkazem(tOsoba *Ktera)
{
	printf("PSC: %d", Ktera->Adresa.PSC)
}
```

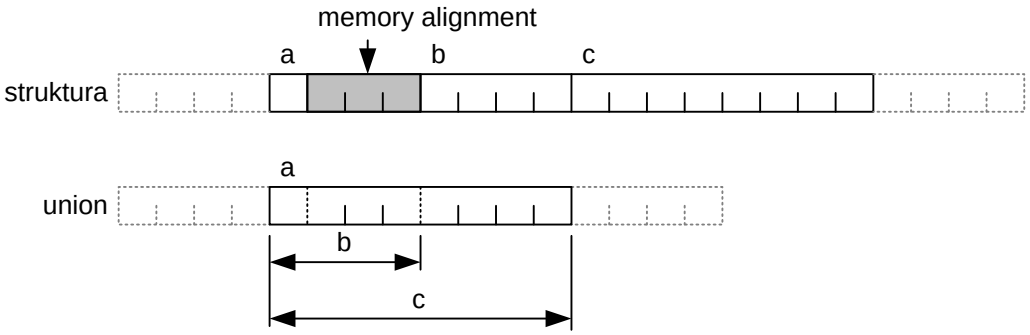


Uniony

- = struktura – položky se překrývají!!!
- definice, práce → stejné možnosti jako struktury
- velikost = největší položka

```
struct {
	char a;
	int b;
	double c;
} promStruct;
```

```
union {
	char a;
	int b;
	double c;
} promUnion;
```



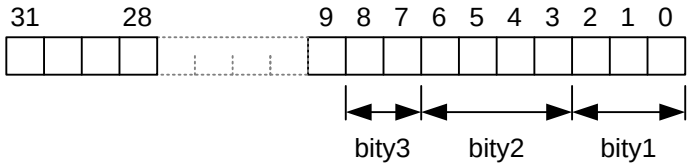
- využití:
  - šetření místem
  - nestandardní operace (např. výpis čísla `double` po bytech)

Bitová pole

- speciální případ struktur
- velikost pevně `int` → prvky = „pole bitů“ v jednom `int` = více čísel v jednom
- definice

```
struct {
      unsigned bity1: 3;
      unsigned bity2: 4;
      unsigned bity3: 2;
} bitovePole;
```

  - lze i `signed` nebo kombinace
  - obsazení bitů od LSB



- využití:
  - „zapouzdření“ bitových operací
  - šetření místem

- př.: uložení datumu v MS-DOS

```
typedef struct {
      unsigned den      : 5;
      unsigned mesic    : 4;
      unsigned rok      : 7;  // 7 bitu = 128 let!!!
} tDatum;
int main(void)
{
      tDatum datum;
      datum.den = 13;
      datum.mesic = 2;
      datum.rok = 2006 - 1980;
      printf("datum: %d.%d.%d\n", datum.den, datum.mesic,
            datum.rok + 1980);
      return 0;
}
```

Výčtový typ

- „zabalení“ konstant do struktury
- možnosti definice = struktura, v praxi většinou přes `typedef`

```
typedef enum {
      konstanta1 [ = inicializace],
      konstanta2 [ = inicializace],
            |
      konstantaN [ = inicializace]
} typEnum;
typEnum promenna;
```

  - vnitřní reprezentace = celé bezznaménkové číslo
  - inicializace nepovinná (možnosti: žádná, několik i napřeskáčku, všechny)
  - pokud není inicializace, 1. konstanta = 0
  - prvky nejsou nijak s proměnnou svázány
- využití: několik logicky svázaných konstant
  - Příklad 1

```
enum boolean {
      FALSE,
      TRUE
};
int isOK;
if (isOK == TRUE) {
      // akce
}
```

- Příklad 2 – definice přes `typedef`

```c
#include <stdio.h>

typedef enum {
	LEFT = 'L',
	RIGHT = 'R',
	CENTER = 'C'
} Zarovnani;

void tiskni(char *co, Zarovnani zr)
{
	switch (zr) {
		case LEFT:
			// printf vlevo
			break;
		case RIGHT:
			// printf vpravo
			break;
		case CENTER:
			// printf doprostred
			break;
		default:
			break;
	}
}
int main(void)
{
	tiskni("Ahoj", RIGHT);
	return 0;
}
```