Pole a struktury

V předchozí kapitole jsme si ukázali práci s řetězcem jako s polem znaků. Pro získání délky pole můžeme použít operátor `sizeof`, který vrací počet bytů vyhrazených pro proměnnou. Pokud předáme libovolné pole jako parametr funkce, degraduje na konstantní ukazatel. Po předání pole funkci tak už nelze zjistit jeho délku:

```
int data[3];
printf("%d\n", sizeof data); // 12 nebo 24, dle velikosti int
printf("%d\n", sizeof(data) / sizeof(int)); // 3

int getLength(const int* array) {
  return sizeof(array) / sizeof(int);
}

printf("%d\n", getLength(data)); // 1!
```

Ve výše uvedeném je lhostejno, deklarujeme-li array jako `const int[]` nebo `const int[3]`, vždy degraduje na ukazatel. Poslední výpis vrací délku ukazatele, což je adresa, jejíž velikost je prakticky shodná s typem int. Při předání pole jako parametru tedy musíme předat funkci i délku pole:

```
int data[] = {5,3,7,2};

void show(const int* array, int length) {
  if(length<=0) return;
  printf("%d", array[0]);
  for(int i=1; i<length; ++i) printf(", %d", array[i]);
  puts("");
}

show(data, 4); // 5, 3, 7, 2
```

Funkce main má také nepovinné atributy v podobě pole a jeho délky. Tyto atributy obsahují parametry, se kterými byl program spuštěn, první (s indexem 0) je vždy jméno souboru programu:

```
void main(int argc, char** argv) { // char** je pole řetězců
  printf("pocet parametru: %d\n", argc);
  for(int i=0; i<argc; ++i) printf("%s", argv[i]);
}
```

Všimněte si, že všechny parametry jsou typu řetězec. Převod na číslo lze nejintuitivněji funkcí `sscanf`, převod zpět pomocí `sprintf`. Tyto funkce se chovají podobně jako funkce bez onoho `s` na začátku, místo konzole ale mají první parametr řetězec, který konzoli nahrazuje:

převod řetězce na číslo

```
char str[] = "21";
int n; // zde načteme str jako číslo
sscanf(str, "%d", &n);
printf("%d", 2*n); // 42
```

převod čísla na řetězec

```
int n = 42;
char str[3]; // zde načteme n jako řetězec
sprintf(str, "%d", n);
puts(str); // 42
```

Pokud bychom chtěli pole předat včetně délky bez dalšího parametru, můžeme tak učinit pomocí referencí. Ty jsou pouze v jazyce C++:

```
void show(const int(&array)[3]) {
  for(int i=0; i<3; ++i) printf("%d ", array[i]);
}
```

Tato funkce akceptuje pouze pole délky 3. Závorka v typu je důležitá, bez ní by se typ vyhodnotil jako `int& array*`, neboli *pole referencí na int*, zatímco `int(&array)[3]` znamená *reference na pole int délky 3*. Pro zvídavé existuje možnost generického programování v C++:

```
template<int N>
void show(const int(&array)[N]) {
  for(int i=0; i<N; ++i) printf("%d ", array[i]);
}
```

Zde kompilátor vytvoří pro každé volání `show` funkci s odpovídajícím typem argumentu, což je poněkud kontroverzní. Bude lépe na celou věc prozatím zapomenout a vrátit se do říše jazyka C. Zde můžeme vytvořit „pole s délkou“ pomocí struktur, což je složený datový typ:

```
struct intArray {
  const int* array;
  int length;
};

void show(struct intArray x) {
  for(int i=0; i<x.length; ++i) printf("%d ", x.array[i]);
}

void main() {
  int data[] = {5, 3, 7, 2};
  struct intArray x = {data, sizeof(data)/sizeof(int)};
  show(x);
}
```

Všimněte si vytučnělých `struct`. Struktura totiž v C není datový typ, ale tzv. *tag*, datový typ je vytvořen právě slovem `struct` před deklarací typu. V C++ **je** struktura datový typ a `struct` při deklaraci není nutno uvádět. Tomu se lze vyhnout definicí typu konstruktem `typedef`:

```
struct intArray {
  const int* array;
  int length;
};
// definujme „struct intArray" jako datový typ intArray
typedef struct intArray intArray; // v C++ možno přeskočit

void show(intArray x) {
  for(int i=0; i<x.length; ++i) printf("%d ", x.array[i]);
}
```

```
void main() {
  int data[] = {5, 3, 7, 2};
  intArray x = {data, sizeof(data)/sizeof(int)};
  show(x);
}
```

Dále si všimněte středníku za definicí struktury. Ve skutečnosti je to totiž deklarace, kterou můžeme rovnou spojit s proměnnou:

```
struct intArray {
  const int* array;
  int length;
} x;

void show() {
  for(int i=0; i<x.length; ++i) printf("%d ", x.array[i]);
}

void main() {
  int data[] = {5, 3, 7, 2};
  x.array = data;
  x.length = 4;
  show(x);
}
```

Tento zápis je vhodný k deklaraci statických proměnných.

Jazyk C má oproti C++ výhodu v podobě pojmenovaných inicializátorů:

```
struct intArray y = {.length=4, .array=data};
```

Téhož chování lze v C++ docílit pomocí preprocesoru (viz zde), ale je to v podstatě hack: jazyk C nahlíží na struktury jako *plain old data* (POD), tedy jako na složený datový typ. C++ je má za třídy (rozdíl oproti `class` je v C++ minimální), které by se měly vytvářet pomocí konstruktoru, proto Cčkovou syntaxi potlačuje a není rozumné pokoušet se ji používat.

Cvičení

Vytvořte program `avg`, který spočítá průměr čísel zadaných jako parametry (na 2 desetinná místa). Např.:

```
C:\ avg 1 3 1
1.66
```

Použijte následující strukturu Point

```
struct Point { int X, Y; };
```

k definici funkce `double distance(struct Point A, struct Point B)`, která vrátí vzdálenost dvou bodů v euklidovském prostoru.