Výrazy, operace, příkazy

Karel Richta a kol.

katedra počítačů FEL ČVUT v Praze

Přednášky byly připraveny s pomocí materiálů, které vyrobili Ladislav Vágner, Pavel Strnad

Programování v C++, B6B36PJC

02/2017, Lekce 2

https://cw.fel.cvut.cz/wiki/courses/b6b36pjc/start

Základní datové typy

- ***celočíselné typy:***

  `char` (nejmenší adresovatelná buňka)

  `char, unsigned char, signed char`

  `int` (šířka je závislá na procesoru a systému)

  `short = short int = signed short = signed short int`

  `int = signed = signed int`

  `unsigned = unsigned int`

  `long = long int = signed long = signed long int`

  `unsigned long = unsigned long int`

  `long long = long long int`

  `unsigned long long = unsigned long long int`

  `short <= int <= long`

- ***s pohyblivou řádovou čárkou:***

  `float, double, long double`

- ***logické hodnoty (chybí v C):***

  `true, false`

- `void` – typ s prázdnou množinou hodnot

Základní datové typy: příklad reprezentace

Vnitřní reprezentace skalárních typů na PC

- celočíselné typy (reprezentace v pevné řádové čárce) v doplňkovém kódu se znaménkem
- reálné typy (reprezentace v pohyblivé řádové čárce) podle normy IEC 60559:1989 (ANSI/IEEE)

Velikost vnitřní reprezentace:	**Clang++3.3, 64bit Linux**	**VC++ 12 64bit**
char, signed char, unsigned char	1B	1B
short, unsigned short	2B	2B
int, unsigned int	4B	4B
long, unsigned long	8B	4B
long long, unsigned long long	8B	8B
float	4B	4B
double	8B	8B
long double	16B	8B
Platforma	*I32LP64*	*IL32P64*

I32LP64: Integer má 32 bitů, Long a Pointer mají 64 bitů

Jak zjistit velikost datového typu?

```
#include <iostream>

int main() {
    std::cout << "char: " << sizeof(char) << '\n'
          << "short: " << sizeof(short) << '\n'
          << "int: " << sizeof(int) << '\n'
          << "long: " << sizeof(long) << '\n'
          << "long long: " << sizeof(long long) << '\n'
          << "float: " << sizeof(float) << '\n'
          << "double: " << sizeof(double) << '\n'
          << "long double: " << sizeof(long double) << '\n';
}
```

Popis jazyka

- Popis syntaxe (**jak** se to píše)
- Popis sémantiky (**co** to znamená)
- Standardní knihovny (standardní služby, které musí každá implementace jazyka poskytovat – základ přenositelnosti programů)

Popis syntaxe a sémantiky je uspořádán takto:

- popis použité paměti (**deklarace**)
- popis výpočtů prováděných nad pamětí (**výrazy**)
- popis posloupnosti provádění výpočtů (**příkazy**)

Lexikální elementy

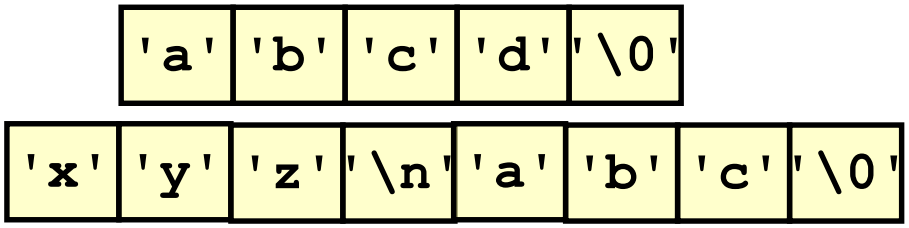
- **identifikátory**
  - písmena (rozlišuje se velikost), číslice, podtržítko
  - délka není omezena, rozlišují se podle úvodních znaků v závislosti na implementaci (ANSI doporučuje nejméně 31 znaků)
  - některé jsou rezervovány jako klíčová slova
- **klíčová slova**
  - `if, while...`
- **oddělovače** a **operátory**
  - `{ } ; + - & && <<=`
- **číselné literály (zápisy čísel)**
  - `125  1'234'567'893'234  125.45e-7  0.12E3`
  - oktalové: `037` hexadecimální: `0x1F` binární: `0b101011`

Lexikální elementy (pokračování)

- **literály řetězců**

  `"abcd"  ""  "xyz\nabc"`

  vnitřní reprezentace: posloupnost znaků zakončená binární nulou `'\0'`



- **znakové literály (zápisy znaků)**

```
'x'  '\n'  '\t'  '\0'  '\\'  '\''  '\014' '\u010C',
wchar_t c = L'\u79c1'; // Unicode řetězec
const char* str = "\xEC\xA4\x91"; //řetězec s UTF-8 znakem
```

- **komentáře**

```
/* toto je komentář */
// toto je komentář v C++ a v novějších normách C (C99+)
```

Deklarace proměnné

paměťová třída kvalifikátor specifikace typu seznam deklarátorů ;

Příklady:

```
int n;
const int x = 2;
extern pole[10];
static long double z;
```

Deklarace proměnné: paměťová třída

paměťová třída ***kvalifikátor*** ***specifikace typu*** ***seznam deklarátorů ;***

Paměťová třída popisuje způsob přidělování paměti:

(nic)	dle kontextu (viz. níže)
static	paměť přidělena staticky během překladu
extern	nepřidělovat paměť

- Implicitně je paměťová třída:

  automatická (ve funkcích, v bloku) nebo
  static (v modulu, globální proměnné)

- Automatické proměnné jsou alokovány na zásobníku, statické zpravidla v datovém segmentu programu. Statické existují po celou dobu výpočtu, automatické jen po dobu zpracování bloku, ve kterém jsou lokální.

Deklarace proměnné: kvalifikátor

paměťová třída kvalifikátor specifikace typu seznam deklarátorů ;

Kvalifikátor popisuje další požadované vlastnosti:

const	konstantní objekt
volatile	objekt, který je sdílen s jiným nezávislým procesem
mutable	vzácné, třídní proměnné, které mohou být modifikovány na const objektu

```
extern const int Max;
...
Max = 10;/* chyba */
volatile int regs;
```

Deklarace proměnné: specifikace typu

paměťová třída kvalifikátor specifikace typu seznam deklarátorů ;

Specifikace typu je:

- **void** zastupuje „žádný“ typ (má prázdnou množinu hodnot)
- *identifikátor typu* (základního nebo uživatelem definovaného)
- popis struktury (**struct**), popis třídy (**class**), popis sjednocení (**union**), nebo výčtového typu (**enum**)
- **auto** typ je doplněn kompilátorem jako typ na pravé straně výrazu

Deklarace proměnné: seznam deklarátorů

paměťová třída kvalifikátor specifikace typu seznam deklarátorů ;

Seznam deklarátorů je výčet deklarátorů oddělený čárkami.
Deklarátor má syntaxi:

identifikátor	
* deklarátor	*ukazatel*
deklarátor [konstantní výraz]	*pole*
deklarátor (parametry)	*funkce*
(deklarátor)	

Pozn.: za * může být kvalifikátor.

Deklarace proměnné

paměťová třída kvalifikátor specifikace typu seznam deklarátorů ;

Příklady:

`int i;`	proměnná typu `int`
`int *ai;`	proměnná typu ukazatel na `int`
`int ia[10];`	proměnná typu pole s `10` prvky typu `int`
`int fi(double);`	proměnná typu funkce z `double` do `int`

Čtení složitějších deklarací

Deklarátory je třeba číst **„zevnitř ven“** postupně podle následujících pravidel:

- najdi deklarovaný identifikátor a zjisti, zda se **vpravo** od něj nachází [nebo (
- interpretuj tyto závorky a pak zjisti, zda se **vlevo** od identifikátoru nachází *
- kdykoliv narazíš na), vrať se zpět a aplikuj předchozí pravidla uvnitř závorek (a)
- nakonec použij specifikaci typu

Příklad čtení deklarace

```
char *( *(*x)() )[10];
```

1. identifikátor **x** je deklarován jako
2. ukazatel na
3. funkci vracející
4. ukazatel na
5. pole 10-ti elementů, kterými jsou
6. ukazatelé na
7. hodnoty typu **char**

Definice vlastních typů

Syntaxe:

using *alias* = *typ*;
typedef *typ alias*

Příklady:

```
using Byte = unsigned char;
typedef unsigned char Byte;

using Word = short int;
typedef unsigned short int Word;

using Matice = int[10][10];
typedef int Matice[10][10];
```

Možné deklarace:

```
static Word *ptr;
const Matice m = {{ /**/ }};
```

```
const Byte b = 0xBE;
extern Word w;
```

Ukazatelé (pointers)

- Ukazatel je datový typ, který obsahuje paměťovou adresu. Říkáme, že ukazatel na tuto adresu ukazuje.
- Navíc rozlišujeme ukazatele podle toho, jaký typ proměnné se na adrese nachází.
  - `int*` ukazuje na adresu, kde se nachází `int`. Čti: ukazatel na `int`.
  - `char*` ukazuje na adresu, kde se nachází `char`. Čti: ukazatel na `char`.
  - `bool**` ukazuje na adresu, kde se nachází `bool*`. Čti: ukazatel na ukazatel na `bool`.

Inicializace

- Datové objekty (proměnné, konstanty) se inicializují v době vzniku
  - statické a globální na začátku programu
  - ostatní v době deklarace (na začátku funkce, bloku), pokud jsou statické, tak jen poprvé
- Implicitní inicializace
  - statické objekty jsou vynulovány
  - ostatní nemají definovanou hodnotu
- Explicitní inicializace
  - *deklarátor* = *výraz* nebo
  - *deklarátor* = { *seznam výrazů* }
- Statické objekty lze inicializovat pouze konstantními výrazy
- Příklady:

```
float A = 10.2f;
int B = 123;
```

Inicializace v C++

- Lze použít stejné tvary jako v C.
- Navíc deklarace s explicitní inicializací může mít tvar:

  typ deklarátor (*výraz*) ; nebo
  typ deklarátor { *výraz* } ; (C++11)

```
int a = 5;              // počáteční hodnota: 5
int b(3);               // počáteční hodnota: 3
int c{ 2 };             // počáteční hodnota: 2
int result;             // počáteční hodnota není určena
```

- V deklaraci s explicitní inicializací můžeme použít automatické odvození typu:

```
int a = 0;
auto b = a;  // totéž jako: int b = a;
```

Výrazy

- Bohatý repertoár **operací**: aritmetické, bitové, logické, relační, operace s ukazateli
- Výraz může označovat **modifikovatelný objekt** (lvalue) nebo **hodnotu** (rvalue)
- Pořadí vyhodnocení operandů není (až na výjimky) specifikováno normou, ale implementací!
- Některé operace mají vedlejší efekt (inkrementace, dekrementace, přiřazení) => v jednom výrazu nad jedním objektem max. jeden!

```
i = i++; // Nedefinované chování
a = ++i + i++; // Nedefinované chování
```

Výrazy (pokračování)

- Operace se provádějí v těchto aritmetikách:

  `int, unsigned, long, unsigned long,`

  `float, double, long double`

- Před provedením operace může dojít k ***implicitní konverzi operandů***:
  - **`char`**, **`short`** (**`signed`**, **`unsigned`**) a **`enum`** jsou před provedením operace konvertovány na **`int`** (nebo **`unsigned int`**) – tzv. ***roztažení*** (***integral promotion***)
  - pole prvků typu *T* je konvertováno na ukazatel na *T*
  - jméno funkce je konvertováno na typ ukazatel na funkci

Výrazy (pokračování)

U většiny operací s aritmetickými operandy se provádějí tzv. ***běžné aritmetické konverze*** (usual arithmetic conversion):

- operandy se konvertují pomocí roztažení, a následně se převedou na „nejširší“ typ

long double
double
float
unsigned long
long
unsigned int
int

Typy číselných literálů

- Typy číselných literálů:

`12`	`int`
`12U`	`unsigned int`
`12L`	`long`
`12ULL`	`unsigned long long`
`3.4`	`double`
`3.4F`	`float`
`3.4L`	`long double`

- Příklady:

`'a' + 'b'`	konverze na `int`, výsledek `int`
`1 / 2`	celočíselné dělení, výsledek `0`
`1.0 / 2.0`	reálné dělení, výsledek `double`
`1 / 2.0L`	`1` je konvertována na `long double`

Přiřazení

- Syntaxe: ***X*** = ***Y***
- Levý operand musí být modifikovatelný (lvalue)
- Přiřazení má hodnotu a vedlejší efekt
- Hodnotou výrazu je hodnota ***X*** po přiřazení
- Pozor: plete se = a ==
- Přípustné typy operandů:
  - levá a pravá strana jsou téhož skalárního typu nebo téhož typu **`struct`**, **`class`** nebo **`union`** (pro pole není přiřazení dovoleno)
  - jsou-li typy levé a pravé strany skalární, avšak různé, musí být hodnota pravé strany konvertibilní na typ levé strany
- Složené přiřazení:

  `+=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=`

- Význam:
  - *X op= Y* je zkratkou za *X = X op Y*

Operace

- Aritmetické:
  - Unární: `a++ a-- ++a --a`
  - Binární: `a+b a-b a*b a/b a%b`
- Relační:
  - Binární: `a<b a>b a<=b a>=b a==b a!=b`
- Logické:
  - Unární: `!a`
  - Binární: `a&&b a||b`
- Bitové:
  - Unární: `~a`
  - Binární: `a&b a|b a^b a<<b a>>b`
- Další:
  - Binární: `a=b a,b`
  - Ternární: `a?b:c`

Konverze při přiřazení

Následující tabulka udává možné konverze při přiřazení:

typ pravé strany	typ levé strany	poznámka ke konverzi
reálný	kratší reálný	zaokrouhlení mantisy
reálný	delší reálný	doplnění mantisy nulami
reálný	celočíselný	odseknutí necelé části
celočíselný	reálný	možná ztráta přesnosti
celočíselný	kratší celočíselný	odseknutí vyšších bitů
celočíselný unsgn.	delší celočíselný	doplnění nulových bitů
celočíselný sgn.	delší celočíselný	rozšíření znaménka
nullptr	T *	
T * nebo nullptr	void *	opačné přiřazení musí být explicitní

Bitové operace

- Jen pro celočíselné operandy
- Unární:
  - `~` negace jednotlivých bitů
- Binární:
  - `&` logický součin jednotlivých bitů
  - `|` logický součet jednotlivých bitů
  - `^` logický *xor* jednotlivých bitů
  - `<<` posun bitové reprezentace levého operandu vlevo
  - `>>` posun bitové reprezentace levého operandu vpravo
- U operací posunu pravý operand udává délku posunu v bitech
- Pozor, plete se `&` a `&&`, `|` a `||` :

```
x = 1; y = 2;
x & y == 0
(x && y) == 1
```

Operace s ukazateli

- Pokud ukazatel u ukazuje na adresu, kde se nachází proměnná p, říkáme, že **u ukazuje na p**.
- Operátor & (address-of) vytvoří ukazatel z proměnné.

```
int promA = 31;
int promB = 13;
int* ukaz = &promA; // ukaz ukazuje na promA
ukaz = &promB;      // ukaz ukazuje na promB
```

- Operátor * (dereference) vytvoří proměnnou z ukazatele.

```
int promC = *ukaz;  // promC je nyní 13
promA = *ukaz;      // promA je nyní 13
*ukaz = 42;         // promB je nyní 42
```

Příkazy

- Obecně:
  - příkazy se dělí na jednoduché a strukturované
  - všechny jednoduché jsou zakončeny středníkem
  - podmínka v cyklech a podmíněném příkazu je dána výrazem skalárního typu; je splněna, je-li hodnota výrazu různá od nuly
- Výrazový příkaz:

  výraz **;**
  - smysl mají jen výrazy s vedlejším efektem:

```
x = y + 2; y++; f(a, 2); g();
g; // příkaz bez vedlejšího efektu
```

- Prázdný příkaz:

```
;
```

  - použití např. jako prázdné tělo cyklu:

```
for (i = 0; i < 10; a[i++] = 0);
```

Příkazy (pokračování)

- Příkaz bloku:

  { ***posloupnost příkazů a deklarací*** }

  - deklarace jsou v bloku lokální,
  - v bloku platí též deklarace z nadřazeného kontextu,
  - lokální deklarace zastíní deklaraci stejného identifikátoru z nadřazeného kontextu,
  - deklarace mohou chybět (složený příkaz).

```
{
        int a, b;
        a = 10;
        {       int c = a; /* c = 10 */
                int a = 20;
                c = a; /* c = 20 */
                a = 0;
        }
        cout << a; /* vypíše se 10 */
}
```

Příkazy (pokračování)

- Podmíněný příkaz:

  `if` (*podmínka*) *příkaz*

  `if` (*podmínka*) *příkaz1* **`else`** *příkaz2*

- Příkaz **`while`**:

  `while` (*podmínka*) *příkaz*

- Příkaz **`do`**:

  `do` *příkaz* **`while`** (*podmínka*) ;

- Příkaz **`for`**:

  `for` (*výraz1* ; *výraz2* ; *výraz3*) *příkaz*

- Příkaz **`for`** řízený rozsahem (C++11):

  `for` (*deklarace* : *rozsah*) *příkaz*

Příklad

```
// Cyklus for řízený rozsahem (range-based)
#include <iostream>
#include <string>

int main()
{
	std::string str{ "Hello!" };
	for (char c : str)
	{
		std::cout << "[" << c << "]";
	}
	std::cout << '\n';
}

// typ proměnné v deklaraci lze nechat odvodit
	for (auto c : str)
```

Příkazy (pokračování)

- Příkaz **`break`**:

  ukončí bezprostředně nadřazený cyklus nebo přepínač

- Příkaz **`continue`**:

  přejde na nové vyhodnocení podmínky cyklu

- Příkaz **`switch`** (přepínač):

  `switch` (***výraz***) ***příkaz***

  - návěští v přepínači:

    `case` ***konst. výraz*****`:`** nebo **`default:`**

Příklad: použití přepínače

```
int zn = 2;

switch (zn) {
case 1: printf("výborně");
case 2: printf("velmi ");
case 3: printf("dobře");
default: printf("nedostatečně");
}
```

Chybně

```
int zn = 2;

switch (zn) {
case 1: printf("výborně"); break;
case 2: printf("velmi ");
case 3: printf("dobře"); break;
default: printf("nedostatečně");
}
```

Správně

Konec