Úvod: C a C++

Karel Richta a kol.

Katedra technických studií
Vysoká škola polytechnická Jihlava

Objektově-orientované programování, OOP
03/2023, Lekce 1

C versus C++

C – Brian Kernighan, Dennis Ritchie 1972 AT&T Bell Laboratories

- 1978 Kernighan, Ritchie: The C Programming Language

C++ – Bjarne Stroustrup 1983 AT&T Bell Laboratories

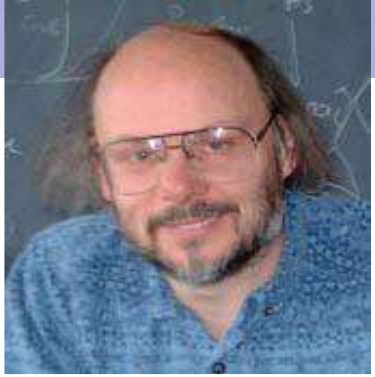
- 1985 Bjarne Stroustrup: The C++ Programming Language
- 1990 Bjarne Stroustrup: The Annotated C++ Reference Manual

C++ = C - nedovolené konstrukce

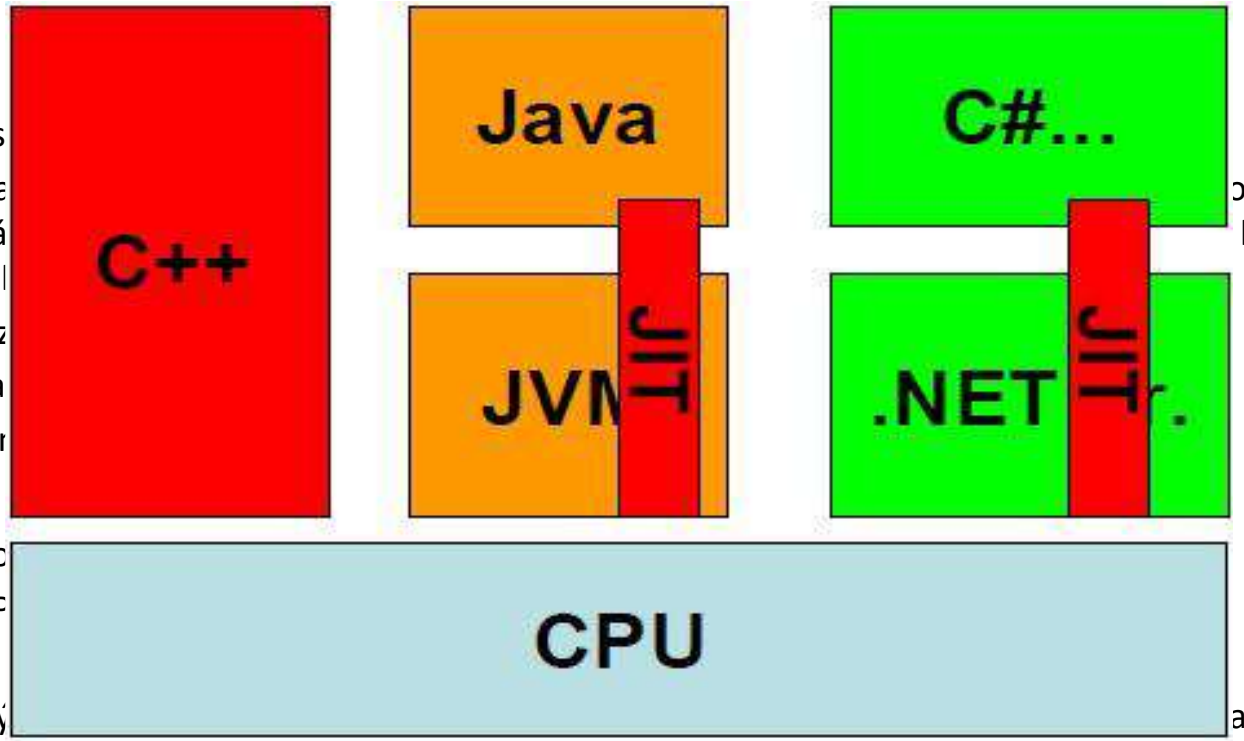
+ navíc neobjektová rozšíření

+ navíc objektová rozšíření

Jazyk C++



- Historie
  - 1979 – C s
  - Nadstavba kompilová všechny pl ový kód je byte code pro
  - Standardiz
  - OOP – z ja
  - Lze psát i r
- Vlastnosti
  - Vysokoúrc
  - Stabilní - F
  - Podpora -
  - Udržovaný adače.
  - Podpůrné knihovny - počínaje databázemi, přes mobilní frameworky, grafické knihovny atd.
  - Budoucnost - bez ohledu na různé trendy, C++ je rychlý jazyk, který je významným (ne však jediným) jazykem pro operační systémy, grafické aplikace a „embedded“ zařízení.
  - Není tak pohodlný, rychlejší jazyky pro vývoj aplikace, nevhodné pro web.



Přehled neobjektových rysů C++

- C++ je rozšíření proti C, ale C není podmnožinou C++

- změny ve funkcích:
  - zrušena možnost funkcí s libovolným počtem parametrů
  - nelze použít nedefinovanou funkci
  - `int f()` znamená `int f(void)`
  - funkce lze přetěžovat (jméno funkce nemusí být jednoznačné)
  - implicitní hodnoty parametrů
  - inline funkce
  - navíc parametry nahrazované odkazem (referencí)

Přetěžování (overloading) funkcí

- Funkce se rozlišují nejen jménem, ale i typy parametrů a počtem parametrů (profilem, aritou)

```
void f(char x) { printf("char %c\n", x); }

void f(int x) { printf("int %d\n", x); }

int main(void)
{
   f('a');
   f(10);
}
```

Přetěžování funkcí (pokračování)

- Při volání přetížené funkce se vyvolá ta, jejíž parametry se nejlépe "spárují" se skutečnými parametry
- Párování (matching) parametrů:
  1. Přesná shoda typů
  2. shoda typů po roztažení (promotion)

     `char` → `int`, `short` → `int`, `enum` → `int`, `float` → `double`
  3. shoda typů po standardní konverzi

     `int` → `float`, `float` → `int`, `int` → `unsigned`, …, `int` → `long`
  4. shoda typů po uživatelské konverzi
- Pokud nejlepší párování má více než jedna funkce, ohlásí se chyba

Příklady přetěžování funkcí

```
void f(float, int);
void f(int, float);
f(1,1);                          // chyba
f(1.2,1.3);                      // chyba

void f(long);
void f(float);
f(1.1);                          // chyba
f(1.1F);                         // f(float)

void f(unsigned);
void f(int);
void f(char);
unsigned char uc;
f(uc);                           // f(int)
```

Implicitní hodnoty parametrů

- V hlavičce funkce lze definovat implicitní (default) hodnotu parametru, skutečný parametr pak lze vynechat
- Skutečné parametry lze vynechávat od konce

```
void f(int x, int y=1, int z=3)
{ printf("x=%d, y=%d, z=%d\n", x, y, z); }
main() {
   f(10, 20, 30);
   f(10, 20);
   f(10);
}
```

- Co není dovoleno

```
f(int x=1, int y);        // chyba
f(int x, int y=10);
f(int x);
f(20);                    // nejednoznačnost
```

parametr s implicitní hodnotou musí být na konci

Inline funkce

- Volání ”**inline**” funkce nemusí být skok do podprogramu, ale může být nahrazeno tělem funkce (překlad jako makro) - posoudí překladač

```
inline int max(int a, int b)
{ return (a>b)?a:b; }

int main()
{
   int a,b=3,c=4;
   . . .
   a=max(b,c);
}
```

Typ reference

- Proměnná typu reference je synonymum jiné proměnné

```
int  i = 0;
int& ir = i;                    // ir ~ i
ir = 2;                         // i = 2
```

Synonymum pro něco, co už musí existovat.

- Proměnná typu reference musí být inicializována

```
int &ir;                        //chyba
```

- Proměnná typu reference může být inicializována konstantou nebo proměnnou jiného typu (vytvoří se odkaz na pomocnou proměnnou)

```
int&  ir  = 6;
float f   = 3.14;
int&  ir2 = f;
ir2 = 10;                       // f se nezmění
```

Parametr typu reference

- Podobný efekt, jako parametr nahrazovaný odkazem

```
void swap1(int *x, int *y)
{ int z; z = *x; *x = *y; *y = z; }

void swap2(int& x, int& y)
{ int z; z = x; x = y; y = z; }

int main()
{
   int a = 10, b = 20;
   …
   swap1(&a,&b);
   swap2(a,b);
   …
}
```

Parametr typu reference (pokračování)

- Pro parametr typu *T&* je přípustným skutečným parametrem, kromě proměnné typu *T*, též:
  - konstanta typu *T*
  - proměnná nebo konstanta typu *T1*, který je kompatibilní vzhledem k přiřazení s *T*
- V těchto případech se vytvoří pomocná dočasná proměnná, do které se zkopíruje (s případnou konverzí) hodnota skutečného parametru, skutečný parametr se nezmění

```
float x=10.2;
const int b=10;
void g(int& x);
g(x);
g(b);
```

```
void f(int *x);
f(&x);  // chyba!
f(&b);  // chyba!
```

Konstantní parametr typu reference

- Pokud funkce nemění hodnotu skutečného parametru typu reference, je vhodné specifikovat parametr jako konstantní

```
struct Complex { float re, im; };
Complex plus(const Complex& x, const Complex& y)
{   Complex z;
    z.re = x.re + y.re;
    z.im = x.im + y.im;
    return z;
}
```

- Je-li skutečným parametrem konstanta, nevytvoří se dočasná proměnná

```
Complex a, b, const i = {0,1};
a = plus(b, i);
```

Vstup a výstup

- Kromě standardních prostředků C lze vstup a výstup v C++ provádět pomocí datových proudů - objektů typu *stream*

```
cin       ~ stdin              cin  >> proměnná
cout      ~ stdout             cout << výraz
cerr      ~ stderr             cerr << výraz
```

```
#include <iostream>
int main()
{   int i;
    std::cout << "zadej i:";
    std::cin >> i;
    std::cout << "i=";
    std::cout << i;
    std::cout << '\n';
}
```

- možno zkrátit na: `std::cout << "i=" << i << '\n';`

Vstup a výstup

- Kromě standardních prostředků C lze vstup a výstup v C++ provádět pomocí datových proudů - objektů typu *stream*

```
#include <iostream>
using namespace std;
int main()
{   int i;
    cout << "zadej i:";
    cin >> i;
    cout << "i=";
    cout << i;
    cout << '\n';
}
```

- možno zkrátit na: `cout << "i=" << i << '\n';`

Přípustné typy ve vstupu/výstupu

Vstup:

```
cin >>   char
         short
         int
         long
         char *
         float
         double
         long double
```

Výstup:

```
cout <<  char
         short
         int
         long
         char *
         float
         double
         long double
         void *
```

Přeskočí počáteční mezery!

Manipulátory

- Slouží pro řízení vstupní a výstupní konverze
- Jsou definovány v souboru **<iomanip>**

```
int n = 36;
cout << n << ',' << hex << n << ',';
cout << oct << n << endl;
cout << dec << setw(6) << n << endl;
```

- Některé manipulátory:

dec	dekadická konverze
hex	šestnáctková konverze
oct	osmičková konverze
endl	konec řádku + ”flush”
setw(*int n*)	šířka položky n znaků
setfill(*int c*)	plnící znak c
setprecision(*int n*)	*n desetinných míst*

Deklarace je příkaz

- Deklarace se může vyskytovat mezi příkazy
- Deklarace se může vyskytovat v příkazu **`for`**

```
int main()
{
   for (int i=0; i<10; i++)
   { … }
   for (int i=10; i>0; i--) // chyba! druhá deklarace
   { … }
}
```

Skok za deklaraci

- Nelze přeskočit deklaraci s inicializací

```
    goto lab2;
    …
lab1:
    int a;
    …
lab2:
    …
    goto lab1;
```

```
    goto lab2;  // chyba!
    …
lab1:
    int a = 10;
    …
lab2:
    …
    goto lab1;
```

Dynamické proměnné

- Pro vytvoření dynamické proměnné se nepoužívá funkce **malloc**, ale operátor **new**

```
int *p = new int;
struct S {int a; char b;};
S *q = new S;
```

- Operátorem **new** lze vytvořit pole

```
int Pocet = 10;
int *pa = new int[Pocet];
```

- Pro zrušení dynamické proměnné slouží operátor **delete**

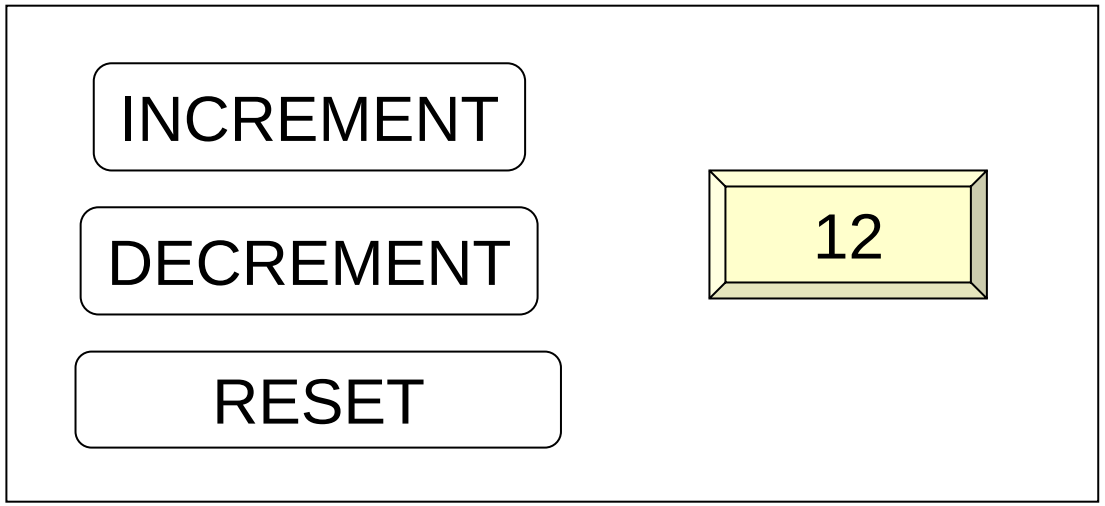
```
delete p;
delete [] pa;
```

- Operátorem **delete** lze zrušit jen to, co bylo vytvořeno operátorem **new**

```
p = pa + 1; delete p;          // chyba
```

Programovací styly

Příklad: Čítač

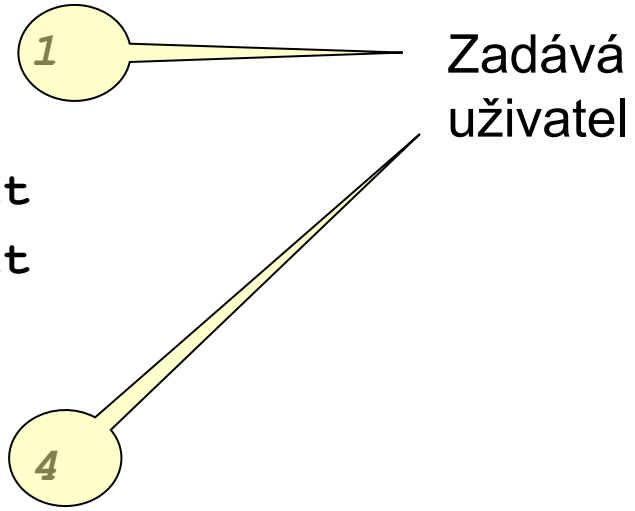


Řešení v různých stylech

- **naivní**
- **procedurální**
- **modulární**
- **objektově orientovaný**

Návrh komunikace s programem

```
Hodnota=0
1. Increment
2. Decrement
3. Reset
4. Konec
Vase volba: 1
Hodnota=1
1. Increment
2. Decrement
3. Reset
4. Konec
Vase volba: 4
Konec
```



Naivní styl

Z jakýchkoli (strukturovaných) konstrukcí sestav řešení

```
#include <iostream>
 using namespace std;
 const int PocHodn = 0;
 int main() {
   int Hodn; char Volba;
   Hodn = PocHodn;
   do {
      cout << "Hodnota = " << Hodn << endl << endl;
      cout << "1. Increment\n";
      cout << "2. Decrement\n";
      cout << "3. Reset\n";
      cout << "4. Konec\n";
      cout << "Vase volba: ";
      cin >> Volba;
      cin.ignore(100, '\n');
      switch (Volba) {
       case '1': Hodn++; break;
       case '2': Hodn--; break;
       case '3': Hodn = PocHodn; break;
       case '4': break;
       default: cout << "Nedovolena volba\n";
      }
   } while (Volba != '4');
   cout << "Konec\n";
}
```

Procedurální styl

Rozlož problém na podproblémy. Navrhni řešení podproblémů jako procedury (funkce). S jejich pomocí sestav řešení.

```
const int PocHodn;
int Hodn;
int Menu()
   /* vypíše nabídku a Přečte volbu
      vrátí: 0=konec, 1,2,3=operace s čítačem
   */
   { … }
void Citac(int op)
   /* provede operaci op s čítačem Hodn */
   { … }
int main()
{  int Volba;
      Hodn = PocHodn;
      do {
         cout << "Hodnota=" << Hodn << endl;
         Volba = Menu();
         if (Volba>0) Citac(Volba);
      } while (Volba>0);
   cout << "Konec\n";
}
```

Objektově orientovaný styl

Rozlož problém na podproblémy. Navrhni potřebné třídy objektů a realizuj všechny operace tříd.

```
/* menu.h */
/* Objektove orientovana specifikace
   typu Menu */
const int MaxPrvku = 9;
class Menu {
   char *Prvky[MaxPrvku];
   char PocetPrvku;
   char *Vyzva;
   int  Volba;
public:
   void AddItem(char *txt);
   void AddPrompt(char *txt);
   int GetUserSelection();
   int GetSelection();
   Menu();
   ~Menu();
};
```

Specifikace třídy (uživatelského typu) Čítač

```
/* citac.h */
/* Objektove orientovana specifikace
   a implementace tridy Citac */
class Citac {
   int   Hodn;
   int   PocHodn;
public:
   Citac(int ph)     {PocHodn = ph; Reset();}
   ~Citac()          {} // lze vypustit, udělá se sám
   void Increment()  {Hodn++;}
   void Decrement()  {Hodn--;}
   void Reset()      {Hodn = PocHodn;}
   int GetValue()    {return Hodn;}
};
```

Deklarace třídy (`class`)

- Typ třída = datové položky + metody + konstruktory + destruktor

```
class T {
  deklarace položek
  prototypy metod              typ jméno( ... );
  prototypy konstruktorů       T( ... );
  prototyp destruktoru         ~T();
};
```

- Definice metody:

```
typ T::jméno( ... ) { ... }
```

- Definice konstruktoru:

```
T::T( ... ) { ... }
```

- V těle metody (konstruktoru, destruktoru) jsou přímo vidět položky (metody) daného objektu

Program s objekty třídy Čítač a Menu

```
/* main1.cpp */
#include "menu.h”
#include "citac.h"
void main()
{  Citac citac(0); Menu menu;
   cout << "\n\nCITAC - Objektove orientovany styl\n";
   menu.AddItem("1. Increment"); menu.AddItem("2. Decrement");
   menu.AddItem("3. Reset"); menu.AddItem("4. Konec");
   menu.AddPrompt("Vase volba: ");
   do { cout << "Hodnota = " << citac.GetValue() << endl;
        switch (menu.GetUserSelection()) {
          case 1: citac.Increment(); break;
          case 2: citac.Decrement(); break;
          case 3: citac.Reset(); break;
        }
   } while (menu.GetSelection() != 0);
   cout << "Konec\n";
}
```

The End