Programovací jazyk C++

Miroslav Melicherčík

Vlastnosti programu

- správna funkcia
  schopnosť vykonávať úlohy podľa zadania
- robustnosť
  schopnosť rozumne fungovať v abnormálnych podmienkach (nesprávny vstup – chybové hlásenie)
- rozšíriteľnosť
  možnosť jednoduchého prispôsobenia zmenám
- opakovateľnosť použitia
  možnosť použitia celého programu alebo časti v novom programe
- zlúčiteľnosť
  možnosť kombinovať s inými produktmi (výmena dát)
- prenositeľnosť
  možnosť preniesť na inú architektúru
- bezpečnosť
- jednoduchosť použitia

História jazyka C++

- autor myšlienky Bjarne Stroustrup (AT&T) 1980
- nazov C++ Rick Mascitti 1983 (predtým C with classes)
- prebral prvky z rôznych jazykov
  Simula67 – triedy
  Algol68 – ľubovoľné umiestnenie deklarácie
  Ada – makrá, resp. šablóny
  Clu – parametrizované moduly
- snaha o spätnú kompatibilitu s jazykom C (väčšinu programov napísaných v jazyku C je možne bez väčších zmien použiť aj v C++)
- C++ je „prísnejší“ ako C

Rozdiely medzi C++ a C

- komentáre

```
i=s//* komentár */ d;
```

- v C `i=s/d`
- v C++ `i=s`

Rozdiely medzi C++ a C

- premenné typu struct a union

```
struct complex {
   float Re;
   float Im;
};
```

- v C `struct complex A;`
- v C++ `struct complex A;` alebo `complex A;`

Rozdiely medzi C++ a C

- operátor prístupu ::

```
::i;
```

- v C++ znamená použitie globálnej premennej
  prístup k inštanciám objektov

Rozdiely medzi C++ a C

- typové prispôsobenie

```
int a;
long b;
```

- v C a C++ `a=(int)b;`
- v C++ `a=int(b);`

Rozdiely medzi C++ a C

- funkčné prototypy

```
int funkcia();
```

- v C **chyba**
- v C++ chapané ako `int funkcia(void);`

Rozdiely medzi C++ a C

- definícia funkcií – hlavička musí obsahovať typ argumentov

```
int fcia (x,y,z)
int x,y;
double z;
{
   …
}
```

- v C++ `int fcia(int x, int y, double z);`

Rozdiely medzi C++ a C

- inline funkcie (zväčša musia byť na jednom riadku)

```
inline int na_druhu(int x) {return x*x;}
```

- v C++ funkcia nie je volaná ako podprogram ale je priamo vsunutá do programu pri kompilácií

Rozdiely medzi C++ a C

- implicitné hodnoty parametrov

```
int sucet(int a=1, int b=5, int c=10)
{return a+b+c;}
```

- v C++

```
sucet(0,1,2);        0+1+2=3
sucet(1,2);          1+2+10=13
sucet(3);            3+5+10=18
sucet();             1+5+10=16
```

```
int sucet(int a, int b=5, int c=10)
```

musí mať aspoň 1 parameter

Rozdiely medzi C++ a C

- preťažovanie funkcií (podľa typu argumentov)

```
int abs(int a) {return a<0?-a:a;}
float abs(float a) {return a<0?-a:a;}
```

- v C `abs, cabs, fabs, labs`
- v C++ `abs`

Rozdiely medzi C++ a C

- generický smerník (void*)

```
void *abc;
char *buffer;
```

- v C++ `abc=&buffer[6];` OK
  `buffer=abc;` chyba
  `buffer=(char*)abc;` OK

Rozdiely medzi C++ a C

- referenica, odkaz na typ (alias)

```
int i=0;
int &ir=i;
```

- v C++ `ir=2;` to isté ako `i=2`
  `&ir` to isté ako `&i`

Rozdiely medzi C++ a C

- definícia premenných na mieste

```
void xyz(void)
{
   int i=8;
   if(1) {
      int jj;
      for(int i=0;i<10;i++)
      {
         …
      }
   }
}
```

Rozdiely medzi C++ a C

- operátory new a delete

```
int *i;
```

- v C
```
i=(int*) malloc(sizeof(int));
free(i);
```
- v C++
```
i=new int;
delete i;

i=new int(2);
```
pri inicializácií priradí do `i 2`

Rozdiely medzi C++ a C

- deklarácia a definícia premnnej

```
extern int i;
```

- v C môže byť deklarácia alebo definícia
- v C++ je deklarácia

Rozdiely medzi C++ a C

- definovanie operátorov

```
int operator == (CAS a, CAS b)
```

…

- okrem ?: . .* ::

Rozdiely medzi C++ a C

- dátové prúdy

  cin, cout, cerr, clog (buffered cerr)

```
stream << data    //vystup
stream >> data    //vstup
```

  napr.

```
cout << "\nTEXT\n";
int i;
cin >> i;
```

Rozdiely medzi C++ a C

- modifikátor const

```
const int k=80;
char line[k];
```

- v C **chyba**
- v C++ **OK**

Rozdiely medzi C++ a C

- znakové konštanty

- v C **su typu int (16, 32 bitov)**
- v C++ **su typu char (8 bitov)**

Rozdiely medzi C++ a C

- inicializácia poľa char

```
char abc[5]=”ahoj”;
char xyz[4]=”ahoj”;
char ppp[4]=('a','h','o','j')
```

- v C `abc,xyz,ppp - OK`
- v C++ `xyz - chyba, abc,ppp - OK`

reťazec je ukončený znakom `\0`
`ahoj\0`

Vlastnosti objektovo orientovaného programovania

- **zapúzdrenie**
  uzatvorenie dát a metód do jedného celku
  sústredenie implementácie dátového typu na jedno miesto a skrytie pre zvyškom programu
- **dedičnosť**
  nové objekty sa dajú definovať na už definovaných
- **polymorfizmus**
  ten istý operátor/metóda môžu mať odlišné správanie pre rôzne dátové typy

Štruktúra

```
struct ryba
   {
        Tcolor farba;
        float hmotnost;
        int dlzka;
        int rychlost_plavania;
        int hlad;
        float x, y, z;
   }

   void plavaj(ryba r);
   int jedz(ryba r, int kolko);
   …
```

Použitie:

```
ryba R;
plavaj(R);
```

Trieda - class

```
class ryba
   {
   Tcolor farba;
   float hmotnost;
   int dlzka;
   int rychlost_plavania;
   int hlad;
   float x, y, z;

   void plavaj(void);
   int jedz(int kolko);
   …
   }
```

Použitie:

```
ryba R;
R.plavaj();
```

Trieda:

inštancie triedy

atribúty - vlastnosti
metódy - funkcie

Trieda - class

- definícia metód triedy

```
void ryba::plavaj(void)
{
    ...
}
```

Trieda - class

- prístup v triede

  public – prístup majú všetci
  protected – prístup majú len metódy vlastnej triedy a dedených potomkov
  private – prístup majú len metódy vlastnej triedy

- v štruktúrach je všetko public
- v triedach je štandardne všetko private

Trieda - class

- atribúty
- metódy
- konštruktor
- deštruktor
- priatelia
- statické metódy a atribúty
- preťažovanie konštruktorov, metód, operátorov

Atribúty triedy

```
class ryba
{
    Tcolor farba;
    float hmotnost;
    int dlzka;
    …
}
```

Metódy triedy

```
class ryba
{
    …
    public:
    void plavaj(void);
    void zmen_farbu(Tcolor farba);
    …
}
```

```
void ryba::plavaj(void)
{
    …
}
```

```
void ryba::zmen_farbu(Tcolor farba)
{
    …
}
```

Konštruktor triedy

- slúži na vytvorenie objektu triedy
- musí byť public
- konštruktor – implicitný (vytvorí miesto v pamäti)
  – explicitný – bez parametrov
  – s parametrami

```
class ryba
{
    ryba(void);
}

ryba::ryba(void) {
    farba = clRed;
}
```

```
...
ryba(Tcolor, int);
...

ryba::ryba(Tcolor f, int d) {
    farba = f; dlzka = d;
}
```

Konštruktor triedy

- použitie bezparametrického konštruktora

```
ryba r;
```

```
ryba *p_r;
p_r = new ryba;
```

- použitie parametrického konštruktora

```
ryba r(clRed, 30);
```

```
ryba *p_r;
p_r = new ryba(clRed, 30);
```

Konštruktor triedy

- kopírovací konštruktor

```
class ryba
{
    ryba(ryba &);
}

ryba::ryba(ryba &x)
{
    farba = x.farba;
    dlzka = x.dlzka;
}

ryba r(s);                    ryba r = s; (implicitný)
```

Deštruktor triedy

- slúži na zrušenie objektu triedy
- nemá parametre a nič nevracia
- musí byť public
- deštruktor – implicitný
  – explicitný

```
class ryba
{
    ~ryba();
}
```

```
ryba::~ryba()
{
    …
}
```

Priatelia triedy - friend

```
class akvarium
{
    float x, y, z;
    float objem;
    int cistota_vody;
    …
    friend class ryba;
}
```

```
class ryba
{
    …
}
```

- ryba môže narábať s akvariom

Statické metódy a atribúty

- statické atribúty sú spoločné pre všetky inštancie triedy
- statické metódy môžeme volať aj bez vytvorenie objektu

```
class ryba
{
    Tcolor farba;
    float hmotnost;
    static int pocet;
    static void plavaj(void);
}

int ryba::pocet = 0;

ryba::pocet = 10;                    ryba::pohni();
```

Preťažovanie

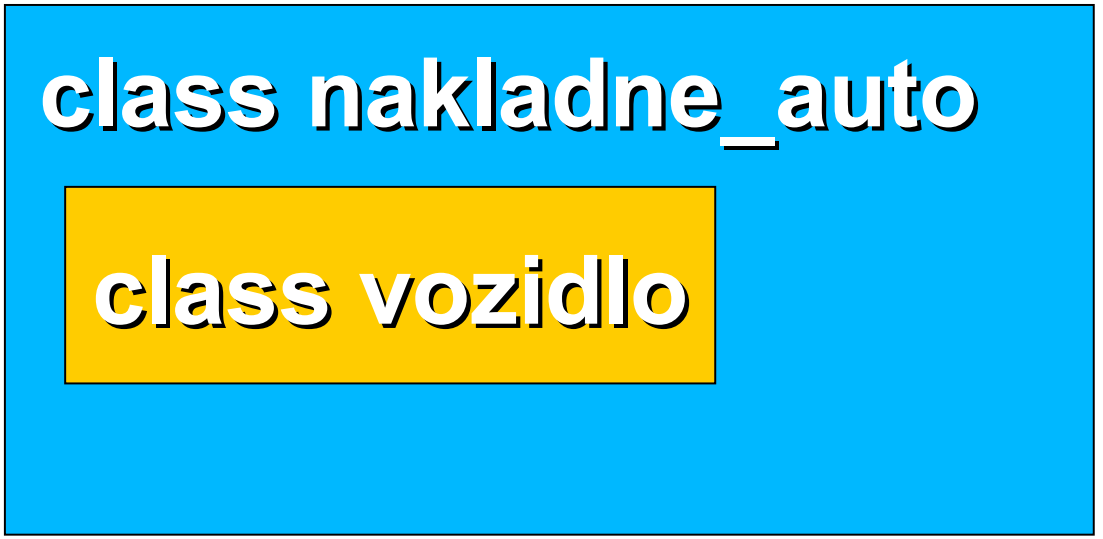
```
class zlomok
{
    int citatel, menovatel;
    …
    const zlomok operator +(const zlomok);
}

const zlomok::operator +(const zlomok x) {
    zlomok pom;
    pom.citatel = …
    pom.menovatel = …
    return (pom);
}
```

Preťažovanie - homonymá

- zachováva sa priorita operátorov
- operátory zachovávajú paritu
- nesmie sa definovať implicitná hodnota parametrov
- nedajú sa vytvárať vlastné operátory
- nerobiť homonymá za každú cenu
- nemeniť význam operátorov
- ak máme homonimum pre +, neznamená to, že máme homonimum aj pre += alebo ++

Dedičnosť

- jedna trieda môže byť základom (podmnožinou) pre inú triedu
- trieda nakladné auto zdedila triedu vozidlo




```
class vozidlo   class nakladne_auto:public vozidlo
{   {
}   }
```

Spôsob dedenia

		spôsob dedenia		
		PRIVATE	**PROTECTED**	**PUBLIC**
prístupové práva rodičov	**PRIVATE**	**PRIVATE**	**PRIVATE**	**PRIVATE**
	PROTECTED	**PRIVATE**	**PROTECTED**	**PROTECTED**
	PUBLIC	**PRIVATE**	**PROTECTED**	**PUBLIC**