FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ústav teoretické a experimentální elektrotechniky

Seminář C++

4. přednáška

Autor: doc. Ing. Jan Mikulka, Ph.D.

2023

Obsah přednášky

- Objekty
  - Přiřazování objektů
  - Předávání objektů funkcím hodnotou
  - Předávání objektů funkcím adresou
  - Předávání objektů funkcím referencí
  - Vracení objektů funkcemi návratovou hodnotou
  - Vracení objektů funkcemi parametrem
- Spřátelené funkce (**friend**)
- Kopírovací a konverzní konstruktory

Objekty

- Co je objekt
  - Třída (definovaná klíčovým slovem **class**)
  - Instance – proměnná typu třída

Přiřazování objektů

- Objekty přiřazujeme stejně, jako proměnné nativních typů
- Jak to bude s pointery?

Příklad přiřazení objektů

```
int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6);

    cout << "A: ";
    A.tisk();
    cout << endl;

    cout << "B: ";
    B.tisk();
    cout << endl;

    B = A;

    cout << "B: ";
    B.tisk();
    cout << endl;

    system("pause");
    return 0;
}
```

Předávání objektů funkcím hodnotou

- Kopírují se všechna data do dočasné lokální instance funkce
  - Plýtvá časem – dochází ke kopírování všech zapouzdřených dat
  - Plýtvá pamětí – v době vykonávání funkce jsou data objektu v paměti 2x

```
void tisk(komplex K)
{
    K.tisk();
}


int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6);

    A.tisk();
    cout << endl;
    tisk(B);
    cout << endl;

    system("pause");
    return 0;
}
```

Objekt jako parametr funkce

S objektem pracujeme jako s hodnotou

Objekt předáváme jako hodnotu

Předávání objektů funkcím adresou

- Nekopírují se žádná data, předává se pouze adresa objektu
  - Šetří čas
  - Šetří paměť

```
void tisk(komplex *K)
{
    K->tisk();
}


int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6);

    A.tisk();
    cout << endl;
    tisk(&B);
    cout << endl;

    system("pause");
    return 0;
}
```

Pointer na objekt jako parametr funkce

S objektem pracujeme jako s pointerem

Předáváme adresu objektu

Předávání objektů funkcím referencí

- Chová se stejně jako u pointeru, ale vytváří se zástupné jméno instance, čili s objektem pracujeme jako s hodnotou.

```
void tisk(komplex &K)
{
    K.tisk();
}


int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6);

    A.tisk();
    cout << endl;
    tisk(B);
    cout << endl;

    system("pause");
    return 0;
}
```

Reference na objekt jako parametr funkce

S objektem pracujeme jako s hodnotou

Předáváme hodnotu objektu

Rozdíl mezi pointerem a referencí

- Reference, podobně jako pointer, slouží k odkazování se na data v paměti.
- S referencí pracujeme stejně jako s instancí (hodnotou).
- Reference je jakýmsi zástupcem instance a vždy zastupuje pouze jednu instanci; nikdy ji nelze přesměrovat na jinou instanci.

Vracení objektů funkcemi návratovou hodnotou

- Nevýhoda – musíme ve funkci vytvořit lokální instanci – nároky na paměť.

```
komplex soucet(komplex &A, komplex &B)
{
    komplex C(0,0);
    C.setr(A.getr() + B.getr());
    C.seti(A.geti() + B.geti());
    return C;
}

int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6), C(0,0);

    C = soucet(A, B);
    C.tisk();

    cout << endl;

    system("pause");
    return 0;
}
```

Objektový typ návratové hodnoty

Pomocná lokální instance, do které uložíme výsledek

Vracení instance

Přiřazení návratové hodnoty

Vracení objektů funkcemi parametrem

- Výhoda – není třeba pomocné lokální proměnné – šetří čas a paměť.

```
void soucet(komplex &A, komplex &B, komplex &C)
{
    C.setr(A.getr() + B.getr());
    C.seti(A.geti() + B.geti());
}

int _tmain(int argc, _TCHAR* argv[])
{
    komplex A(2,-3), B(5,6), C(0,0);

    soucet(A, B, C);
    C.tisk();

    cout << endl;

    system("pause");
    return 0;
}
```

Nepotřebujeme vracet návratovou hodnotu

Zapisujeme přímo do reference na instanci zadanou parametrem

Instanci, do které chceme uložit výsledek předáme parametrem – musí být reference nebo pointer!!!

Spřátelené funkce

- Spřátelením funkce s určitým objektem se rozumí předání veškerých práv a přístupů k zapouzdřeným prvkům třídy dané funkci, která není členskou funkcí této třídy.
- V objektu se funkce pouze deklaruje a klíčovým slovem **friend**.
- Funkce je definována mimo třídu a není jejím členem; pouze se tak chová.

Spřátelené funkce

```
class komplex
{
    double re, im;
public:
    komplex(double re, double im)
    {
        this->re = re;
        this->im = im;
    }

    friend void soucet(komplex &A, komplex &B, komplex &C);
};
```

```
void soucet(komplex &A, komplex &B, komplex &C)
{
    C.re = A.re + B.re;
    C.im = A.im + B.im;
}
```

Deklarace spřátelené funkce

Definice spřátelené funkce

Přístup nečlenské funkce k privátním proměnným

Kopírovací konstruktor

- Slouží k tzv. hluboké kopii objektu.

- Druhy kopií objektů
  - Mělká kopie
    - V případě, kdy nedefinujeme kopírovací konstruktor a např. předáme funkci parametr hodnotou – vytváří se automaticky mělká kopie této předávané instance na lokální proměnnou funkce. Všem zapouzdřeným proměnným je postupně přiřazena hodnota z proměnných instance předané parametrem.
    - Problém nastává u zapouzdřených pointerů
      - Kdo bude alokovat paměť?
      - Kdo bude kopírovat data?
      - Přiřazením pointer = pointer říkáme, že pouze ukazují na stejnou adresu v paměti -> změnou hodnoty uvnitř jedné instance dochází ke změně hodnoty uvnitř zkopírované instance. PROBLÉM!

Kopírovací konstruktor

- Druhy kopií objektů
  - Hluboká kopie
    - Řeší skutečné a správné kopírování dat.
    - Jednotlivé statické proměnné jednoduše přiřadí.
    - V případě pointerů
      - Alokuje paměť přesně podle velikosti alokace ve vzorové instanci.
      - Následně kopíruje prvek po prvku ze vzorové instance do nové instance.
    - V případě zapouzdření alespoň jednoho pointeru musíme vždy implementovat kopírovací konstruktor a řešit hlubokou kopii zapouzdřených dat.

Kopírovací konstruktor

- Kopírovací konstruktor je funkce, která má stejný název jako objekt a má jediný parametr typu reference na tentýž objekt.

Kopírovací konstruktor

```
class pole
{
    int *p, s;

public:
    pole(int s) {
        p = new int[this->s = s];
    }

    pole(pole &x) {
        p = new int[s = x.s];
        for (int i = 0; i < 10; i++) p[i] = x.p[i];
    }

    void set(int i, int c) {
        p[i] = c;
    }

    void tisk(void) {
        for (int i = 0; i < s; i++) cout << p[i] << ", ";
    }

    ~pole() {
        delete [] p;
    }
};
```

Zapouzdřený pointer; potřebujeme KK

Kopírovací konstruktor

Alokace paměti podle velikosti vzorové instance

Kopírovaní dat ze vzorové instance

Kopírovací konstruktor

```
int _tmain(int argc, _TCHAR* argv[])
{
    pole p(10);

    for (int i = 0; i < 10; i++) p.set(i, i);

    pole r(p);

    r.set(5, 100);

    p.tisk();
    cout << endl;
    r.tisk();

    cout << endl;

    system("pause");
    return 0;
}
```

Objekt p vzniká konstruktorem s jedním parametrem

Objekt r vzniká kopírovacím konstruktorem

Bez kopírovacího konstruktoru by se změnila i hodnota v objektu p

Při rušení instancí by bez KK nastal problém s dvojí dealokací stejné paměti

Konverzní konstruktor

- Konverzní konstruktor slouží k inicializaci objektu hodnotou, která není stejného objektového typu.

- Jako příklad lze uvést inicializaci objektu komplexního čísla jednou racionální hodnotou, kterou má být nastavena reálná i imaginární část současně. V tomto případě jde o konverzi double na komplex.

Konverzní konstruktor

```
class komplex
{
    double re, im;
public:
    komplex(double re, double im)
    {
        this->re = re;
        this->im = im;
    }

    komplex(double c)
    {
        this->re = this->im = c;
    }

    void tisk(void)
    {
        cout << noshowpos << re << showpos << im << "i" << noshowpos;
    }
};
```

Definice konverzního konstruktoru; konverze double -> komplex

Použití konverzního konstruktoru

```
int _tmain(int argc, _TCHAR* argv[])
{
    komplex k(3);

    k.tisk();
    cout << endl;

    system("pause");
    return 0;
}
```

Děkuji Vám
za pozornost.

mikulka@vut.cz
www.utee.fekt.vut.cz