

4. Proseminář

Funkce

Funkce – Opakování

Deklarace funkce:

<typ> name (<parameters>);

- Deklarace funkcí se většinou umísťují do hlavičkových souborů.
- Deklarace informuje překladač, že funkce existuje, jak se jmenuje, jaké má parametry a jaký je typ návratové hodnoty.

Funkce – Opakování

Definice funkce:

<typ> name (<parametry>) { <tělo> }

- Definice funkce je deklarace, která navíc nese vlastní implementaci.
- Definice se zpravidla neumisťují do hlavičkových souborů (patří do .c, .cc či .cpp souborů).
- Deklarací téže funkce může v daném programu existovat více, definice právě jedna.
- Definice funkcí nelze vnořovat do sebe (funkce nemůže být definována uvnitř jiné funkce).

Předávání parametrů - Opakování

- Parametry jsou inicializovány stejně jako proměnné.
- Pokud předávaný parametr není reference pak je zkopírován!
- Rozlišujeme tedy předávání parametrů hodnotou a odkazem.
- Jak je to s ukazateli? Předávají se hodnotou nebo odkazem?

Předávání parametrů - Ukazatelů

- Pokud je předávaným parametrem ukazatel, pak je také zkopírován.

```
void reset(int *ip) {  
    *ip = 0; // změní hodnotu objektu na  
             který ip ukazuje  
    ip = 0; // změní hodnotu lokálního ip  
}
```

```
int i = 10;
```

```
int *g_ip = &i;
```

```
reset(g_ip); // při volání je g_ip zkopírován
```

Předávání parametrů - Ukazatelů

- Pokud chceme funkci znemožnit změnu předávaného objektu, pak použijeme modifikátor **const**.

```
void use_ptr(const int *p) {  
    *p = 10; // nepůjde přeložit, p můžeme  
             pouze číst  
}
```

Předávání parametrů

- Předávání parametrů hodnotou není vhodné:
 - pokud je parametr výstupní (funkce ho mění),
 - pokud je předávaný objekt velký,
 - pokud nelze předávaný parametr zkopírovat (např. speciální objekty reprezentující fyzické rozhraní).
- V těchto případech předáváme parametry ukazatelem nebo odkazem.

Ukazatele na funkce

- Jeho hodnotou je adresa funkce.
- Deklarace:
<typ> (*name) (<parametry>);
- Typ je návratový typ funkce na kterou může ukazatel ukazovat.
- Name je název ukazatele.

Ukazatele na funkce

Příklad

```
int gcd(int a, int b); // deklarace funkce
// ukazatel fp na funkci vracející int, se dvěma
// parametry int
int (*fp)(int, int);
fp = gcd; // do fp uložíme adresu funkce gcd
fp(20, 30); // zavolá gcd(20, 30)
int *abc (int, int); // deklarace funkce vracející
                        int * !!!
```

Ukazatele na funkce

- Pro přehlednost je lépe definovat typ ukazatele na funkci pomocí typedef.

```
typedef int (*FP)(int, int);
```

```
FP function_pointer;
```

```
//FP2 je ukazatel na funkci vracející int *
```

```
typedef int *(*FP2)(int *, int *);
```

Ukazatele na funkce

- Jak definovat funkci vracející ukazatel na funkci?

// toto je jedna možnost

```
int (* get_pf(int a))(int, int) {  
    return gcd;  
}
```

// nebo pomocí typedef

```
FP get_pf(int a) {  
    return gcd;  
}
```

Zadání prosemináře – Příklad 1

- Viz zadani1.cpp
- Implementujte chybějící funkce
- Parametry které jsou konstantní definujte `const`

Zadání prosemináře – Příklad 2

Navrhněte funkci *calculate*, která má dva parametry (*arg1*, *arg2*) typu *double* a třetí parametr(*fun*) je ukazatel na funkci, která má dva argumenty typu *double* a typ návratové hodnota je *double*. Funkce *calculate()* také vrátí hodnotu typu *double*, kterou vrátí *fun(arg1, arg2)*.

Příklad:

```
double add(double x, double y)
{
    return x + y;
}
double q = calculate(2.5, 10.4, add); // q = 12.9
```

Zadání prosemináře – Příklad 2

Upravte funkci `calculate`(`calculate_add_vec`) tak, aby její třetí parametr byl vector funkcí stejného typu jako `fun`. Postupně se zavolá každá funkce ve vektoru s parametry `arg1`, `arg2` a výsledky se sečtou.

Bude tedy platit:

```
calculate(arg1, arg2, add) ==  
calculate_add_vec(arg1, arg2, vec)  
// iff add e vec ^ vec.size==1
```

Zadání prosemináře – Příklad 2

Zobecněte funkci `calculate_add_vec` (*calculate_fun_vec*) tak, aby její čtvrtý parametr (*op*) byl typu jako fun a pátý parametr typu double (*neutral*) - neutrální prvek operace *op*.

Pokud není pátý parametr předán, pak je implicitně 0.0. Funkce `calculate_fun_vec` bude pracovat stejně jako `calculate_add_vec`, ale místo sčítání se použije *op*.

Bude tedy platit:

```
calculate_add_vec(arg1, arg2, vec) ==  
calculate_fun_vec(arg1, arg2, vec, add, 0.0)
```

Zadání domácího úkolu

- Zadání bude zasláno emailem do pátku 5.11.2010 a vystaven na stránkách předmětu.