AdvPT Exercise 3

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Function resolving

- What is function overloading and function resolving?

```
void print(int a) { std::cout << "int" << std::endl; }
void print(float b) { std::cout << "float" << std::endl; }
void print(char c) { std::cout << "char" << std::endl; }

int main() {
        float c = 1u;
        print(c);

}
```

- Which function will be called?

Function resolving

1. Gather *viable functions*
   1. Gather all functions in the current scope that have the same name as the function called (-> *candidate functions*)
   2. Filter all functions with a non-matching number of parameters
2. Check the number of functions
   1. 0 -> compiler error
   2. 1 -> call that function
   3. >1 -> select best match; if there is no best match the compiler will report on an ambiguous function call -> compiler error

Function resolving


```
void print(int a) { std::cout << "1" << std::endl; }
void print(int a, int b) { std::cout << "2" << std::endl; }
void print(int a, int b, int c = 0) { std::cout << "3" << std::endl; }

int main( ) {
        print(1);
        print(1, 1);
        print(1, 1, 1);

}
```

Function resolving

- Best match
  - Each parameter type is matched against the types passed in the call
  - In decreasing order of ‘goodness’:
    - An exact match (e.g. double -> double)
    - A promotion (e.g. float -> double)
    - A standard type conversion (e.g. int -> float)
    - A constructor or user-defined type conversion (e.g. int -> class A)
- Choosing a winner
  - Candidates are as strong as their weakest match
  - Candidates with an equivalent number and type of weakest match are compared on their next-weakest (and so on)

Function resolving


```
void print(char c) { std::cout << "char" << std::endl; }
void print(int a) { std::cout << "int" << std::endl; }
void print(double b) { std::cout << "double" << std::endl; }

int main( ) {
        char c = 1;
        print(c);

        short s = 1;
        print(s);
        float f = 1;
        print(f);

        long long ll = 1;
        print(ll);
        long double ld = 1;
        print(ld);

}
```

Function resolving


```
void print(long a, int b = 0) { std::cout << "long" << std::endl; }
void print(double a, int b = 0) { std::cout << "double" << std::endl; }
void print(float a, int b) { std::cout << "float" << std::endl; }

int main() {
        print(1.f);
        print(1.f, 'c');

}
```

Function resolving

- Member functions
  - Candidate functions that are member functions are treated as if they had an extra parameter which represents the object for which they are called; it appears before the first of the actual parameters
  - ⇒ Functions discarding qualifiers (e.g. const functions called on a non-const object) don’t count as valid candidate functions
  - Static functions are handled just as non-static functions for the overload resolution

Function resolving


```
struct A {
        static void print(long a, int b = 0) { std::cout << "static long" << std::endl; }
        static void print(float a, int b) { std::cout << "static float" << std::endl; }

        void print(double a, int b = 0) { std::cout << "double" << std::endl; }
        void print(float a, int b) { std::cout << "float" << std::endl; }
};

int main( ) {
        A::print(1.f);
        A().print(1.f, 'c');

}
```

Function resolving


```
struct A {
        void print(long a, int b = 0) const { std::cout << "const long" << std::endl; }
        void print(double a, int b = 0) const { std::cout << "const double" << std::endl; }

        void print(double a, int b = 0) { std::cout << "double" << std::endl; }
        void print(float a, int b) { std::cout << "float" << std::endl; }
};

int main( ) {
        A().print(1.f);
        A().print(1.f, 'c');
        A().print(1l, 1);
        const A cA;
        cA.print(1.f, 'c');

}
```

Function resolving


```
struct A {
        void print(int a) { std::cout << "A int" << std::endl; }
        virtual void print(char c) { std::cout << "A char" << std::endl; }
        void print(float b) { std::cout << "A float" << std::endl; }
};

struct B : A {
        void print(int a) { std::cout << "B int" << std::endl; }
        void print(char c) { std::cout << "B char" << std::endl; }
        virtual void print(float b) { std::cout << "B float" << std::endl; }
};

int main( ) {
        A a; A* ap = &a;
        B b; B* bp = &b;

        ap->print(1);
        bp->print(1);

        A* ba = bp;
        ba->print(1);
        ba->print('1');
        ba->print(1.f);

}
```

Function resolving

- Template functions
  - Works basically just like ‘regular’ functions
  - Template arguments are deduced automatically if not provided explicitly
  - In case of ambiguity, the most specialized version is chosen
  - If there is more than one ‘most specialized’ version -> compiler error

Function resolving

```
template<typename T1, typename T2> void print (T1 a, T2 b) { std::cout << "generic" << std::endl; }
template<typename T2> void print (char a, T2 b)            { std::cout << "specialized" << std::endl; }
template<> void print (int a, int b)                       { std::cout << "int" << std::endl; }
template<> void print (float a, float b)                   { std::cout << "float" << std::endl; }
template<> void print (char a, char b)                     { std::cout << "char" << std::endl; }

int main( ) {
        print('1', '1');
        print('1', 1.f);
        print(1, '1');

}
```

Function resolving


```
template<class T> struct A {
        static void print (T a) { std::cout << "base" << std::endl; }
};

template<> struct A<int> {
        using T = int;
        static void print (T a) { std::cout << "int" << std::endl; }
};

template<> void A<double>::print (double a) { std::cout << "double" << std::endl; }

int main( ) {
        A<char>::print('1');
        A<int>::print('1');
        A<double>::print('1');

}
```

Function resolving

- Short summary
- More in-depth resources available online, e.g.

  http://en.cppreference.com/w/cpp/language/overload_resolution

  http://en.cppreference.com/w/cpp/language/virtual

  http://en.cppreference.com/w/cpp/language/template_argument_deduction

The End

Questions ?

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT