8_函数重载分析_上

- 函数重载(Function Overload)
  - `用同一个函数名定义不同的函数`
  - 当函数名和`不同的参数搭配`时，函数的含义不同
  - 满足以下条件其中一条则构成函数重载
    - `参数个数`不同
    - `参数类型`不同
    - `参数顺序`不同
  - example lesson_8-1.cpp
  - 当`函数默认值`遇到`函数重载`时会发生什么?

    `lesson_8-2.cpp`

```cpp
#include <stdio.h>

int func(int, int);  // ambiguous
int func(int, int, int c = 0);  // ambiguous

int main(void)
{

    printf("%d\n", func(1, 2));  // ambiguous

    return 0;
}

int func(int a, int b)  // ambiguous
{
    return a + b;
}

int func(int a, int b, int c)  // ambiguous
{
    return a + b + c;
}
```

    输出

```
lesson_8-2.cpp:9:29: error: call of overloaded ‘func(int, int)’ is ambiguous
    9 |     printf("%d\n", func(1, 2));
      |                             ^
```

```
lesson_8-2.cpp:3:5: note: candidate: ‘int func(int, int)’
    3 | int func(int, int);
      |     ^~~~
lesson_8-2.cpp:4:5: note: candidate: ‘int func(int, int, int)’
    4 | int func(int, int, int c = 0);
      |     ^~~~
```

编译器调用重载函数的准则

- 将所有同名的函数作为候选者
- 尝试寻找可行的候选函数
  - 精确匹配实参
  - 通过默认参数能够匹配实参
  - 通过默认类型转换匹配实参
- 匹配失败
  - 最终寻找到的候选函数不唯一，则出现二义性，编译失败。`lesson_8-2` 就是二义性编译失败，因为有两个 `candidate`
  - 无法匹配所有候选者，函数未定义，编译器失败

函数重载的注意事项

- 重载函数在 `本质` 上是相互独立的不同函数
- 重载函数的 `函数类型不同`，如：`int(*)(char, int)`
- 函数 `返回值` 不能作为函数重载的依据

函数重载是由 函数名 和 参数列表 决定

函数重载的本质

- 如何证明程序中同名的 `add` 函数是独立的？
  `lesson_8-3.cpp`

```cpp
#include <stdio.h>

int add(int, int);
int add(int, int, int);


int main(void)
{

```

```
    printf("%d\n", add(1, 2));
    printf("%d\n", add(1, 2, 3));


    return 0;
}

int add(int a, int b)
{
    return a + b;
}

int add(int a, int b, int c)
{
    return a + b + c;
}
```

i. 第一种证明方法：输出两个函数类型不同的 `add` 函数的地址

`lesson_8-4.cpp`

```
#include <stdio.h>

int add(int, int);
int add(int, int, int);

int main(void)
{

    //printf("%d\n", add(1, 2));
    //printf("%d\n", add(1, 2, 3));

    printf("%p\n", (int(*)(int, int))add);
    printf("%p\n", (int(*)(int, int, int))add);


    return 0;
}

int add(int a, int b)
{
    return a + b;
}

int add(int a, int b, int c)
{
    return a + b + c;
}
```

输出

```
root@ubuntu:~/exp/DT_CPP/part01# ./a.out
0x55eebf5af188
0x55eebf5af1a0
```

> 通过观察函数的地址，这两个同名函数的地址值是不同的，所以可以证明重载是相互独立的函数

ii. 第二种证明方法：根据C++符号修饰的规则

`lesson_8-5.cpp`

```cpp
#include <stdio.h>

int add(int, int);
int add(int, int, int);

extern "C" int(*_Z3addii)(int, int);
extern "C" int(*_Z3addiii)(int, int, int);

int main(void)
{
    printf("%p\n", &_Z3addii);
    printf("%p\n", &_Z3addiii);

    return 0;
}

int add(int a, int b)
{
    return a + b;
}

int add(int a, int b, int c)
{
    return a + b + c;
}
```

输出

```
root@ubuntu:~/exp/DT_CPP/part01# ./a.out
0x5626b094d188
0x5626b094d1a0
```

> 先将源码放到 Compiler Explorer，语法选择 `c++`
> 之后 `g++` 编译器会将函数签名进行修饰
> 如：
> `int add(int, int)` 修饰后 `_Z3addii`

> `int add(int, int, int)` 修饰后 `_Z3addiii`
> 通过观察函数的地址，这两个同名函数的地址值是不同的，所以可以证明重载是相互独立的函数

以上两种方式可证明，重载是相互独立的函数

小结

- 函数重载是C++中引入的概念
- 函数重载的 `本质为相互独立的不同函数`
- C++中通过 `函数名` 和 `函数参数` 确定函数调用