Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 4 Functions in C++

- **Const arguments（常量参数）**：
  - When passing arguments by pointers or reference, to avoid changing the value of real argument by the function, declare an argument as **const**

```
int length(const int &a);
```

- **Function overloading（函数重载）**：
  - Use the same function name to create functions that performs a variety of different tasks

```
int volume(int);

double volume(double, int);

long volume(long, int, int);
```

  - The correct function to be invoked is determined by **checking the number and type of the arguments**, but **not on return type**:
    - First try to **find an exact match** in which the types of arguments are the same
    - If an exact match is not found, try the **integral promotions（整型提升）** such as:
      - `char` to `int`
      - `float` to `double` // 这个似乎不是**整型**提升，但被标注在ppt中
    - If both fails, try the **implicit assignment conversion**. If the conversion is possible to have **multiple matches** the compiler will generate an **error** massage.
    - Example:

```
double abs(double num) {
  return ((num < 0) ? -num : num);
}

long abs(long num) {
   return ((num < 0) ? -num : num);
}

int main(){
  abs(10); // Passed as num (converted to long)
}
```

- **Default arguments（默认/缺省参数）**
  - Allow to call a function without specifying all the arguments. Normally a function can contain much more arguments than commonly needed.
  - Default values are specified when the function is **declared**:

```
void func(int num=10);
```

    - `num`: default argument
    - `0`: default value
    - two ways in calling the function : `fun();` and `fun(23);`
  - We must add default **from right to left**:

```
void f(int i, int j=2, int k=5); // OK

void f(int i=2, int j, int k); // Err

void f(int i=0, int j, int k=2); // Err
```

  - When overloading a function with default arguments, pay attention to the **ambiguity problem （二义性）**
- **Inline functions（内联函数）**
  - An inline function is a function that is **expanded in line** when it is invoked.
  - Eliminate the cost of calls to small functions.
  - The inline functions are defined as follows:

```
inline function_header{
  function_body
}
```

  - Difference between inline function and predefined macro:
    - inline:

```
#include <iostream>

using namespace std;

inline double cube(double a){
  return (a*a*a);
}

int main(){
  cout << cube(3.0) << '\n'
       << cube(1.5+2.5) << endl;
  return 0;
}
```

      The output is:

```
27
64
```

    - macro:

```
#include <iostream>

using namespace std;

#define cube(a) (a*a*a)

int main(){
  cout << cube(3.0) << '\n'
       << cube(1.5+2.5) << endl;
  return 0;
}
```

The output is:

```
27
11.5
```

However, although we add branket to macro like `#define cube(a) ((a)*(a)*(a))`, it **still can be cracked** if we use the code `cube(i++)`

- If an inline function is **too long or too complicated**, the complier may compile the function **as a normal function**.
  - a loop, a `switch` or a `goto` exists
  - contain `static` variables
  - recursive function
- `inline` is a suggestion to compiler