VE280_2021SU Midterm RC part 3.

Function pointer, Enum, Program argument

SU Zhenxuan

Overview

- Function pointer
- Enum
- Program taking arguments

Function pointers

- Solve several similar problems
- Writing one function for each problem is boring, and may cause more bugs to appear
- Better way: write a function that takes a function pointer as input
  - By passing different function pointer this function can do different task
  - **Less code, fewer bugs**
  - **Higher level of abstraction** (two major advantages)

Function pointers

- Definition:
  - T0 (*fp)(T1, T2,⋯);
  - T0 is the return type, T1, T2,⋯ are the parameter type
  - Example: int (*fp)(int, int);
  - Recall type signature and function definition
    - int min(int, int) declares a function named “min”
    - Change the function name into (*fp)
    - int (*fp)(int, int) declares a function pointer named “fp”

Function pointers

- Assign a function to a function pointer:
  - fp=min;
- Call function pointer:
  - fp(1, 2);
- Recommended but not only way.
- Note here is different from normal variable pointers.

Function pointers

- Function pointers as function argument
- fi_se(a, b, fp) is a function that takes a function pointer as input

```
#include <iostream>
int fi_se(int a, int b, int (*fp)(int, int)) {
    return fp(a, b);
}
int fi(int a, int b) {
    return a;
}
int se(int a, int b) {
    return b;
}
int main() {
    std::cout << fi_se(2, 3, fi) << '\n' << fi_se(2, 3, se) << '\n';
}
```

Enum

- enum is a **type** whose values are restricted to a set of integer values
- Advantage
  - Use less memory than std::string
  - More readable than const int or char
  - Limit valid value set, so compiler help you find spelling mistakes.

Enum

- Example:

```
#include <iostream>

enum A {
    a, b, c=-1, d, e=5, f, g=a + e, h
};

int main() {
    std::cout << a << ' ' << b << ' ' << c << ' ' << d << ' '
              << e << ' ' << f << ' ' << g << ' ' << h << '\n';
}
```

  - Output is 0 1 -1 0 5 6 5 6
- By default the enum value starts from 0, and increments for each value
  - But you can also assign any integer value to them.
- Values in enum (a, b, c,⋯) can be treated as global const int
  - Can be compared (<, >, ==, !=).

Enum

- Since enum A is a new type, std::cin and std::cout cannot identify them
  - Cast the enum variable to int before print it.

```
#include <iostream>

enum A {
    a, b, c=-1, d, e=5, f, g=a + e, h
};

int main() {
    A A1=a;
    std::cout << A1 << '\n'; //wrong
    std::cout << static_cast<int>(A1) << '\n'; //right
}
```

Enum

- Use const array of char* is a better way to print enum type
  - Enum type can serve as array index (same as const int)

```
enum suit {
    DIAMOND,
    SPADE,
    HEART,
    CLUB
};

const char* suit_name[4] = {"DIAMOND", "SPADE", "HEART", "CLUB"};
#include <iostream>

int main () {
    std::cout << suit_name[DIAMOND] << '\n';
    std::cout << suit_name[SPADE] << '\n';
}
```

Program argument

- Like most Linux command, C++ program can also take argument(s)
- Effect: make more general program

Program argument

- Write a main function that takes program arguments:
  - int main(int argc, char *argv[]) {
    ……
  - }
- Or in a way easier to memorize:
  - int main(int argc, char** argv) {
    ……
  - }
- “arg” for argument, “c” for count, “v” for value or vector.
- argv is a 1-D **array** of c-strings (equivalent to char*), so we need two “*” and get char** argv
  - You can consider argv as a pointer to (pointer to char), or an array of (pointer to char)

Program argument

- ./program_argument
  - argc=1, argv[0]="./program_argument"
- ./program argument 1 2 3 4 > args.txt
  - argc=5, argv[0]="./program_argument", argv[1]="1", argv[2]="2", argv[3]="3", argv[4]="4".
  - spaces between "2" and "3", "3" and "4" do not influence argv

```
//source file name: program_argument.cpp
// g++ -o program_argument program_argument.cpp
#include <iostream>
using std::cout;
int main(int argc, char** argv) {
    cout << argc << '\n';
    for (int i = 0; i < argc; i++) {
        cout << argv[i] << '\n';
    }
}
```