Grammar Guide(finished 50%)

Contents

Quick tour（You need to master a C-style language）

Basic type: **int, real, char, string, bool**

real is equivalent to **double** in C

Branch statements and loop statements must be enclosed in curly brackets
switch supports string matching such as case "DAMN": ...
Support for pointers(not supporting pointer’s pointer), use new to allocate objects on the heap (basically the same as C ++)

```
    int *ptr=new int(123);
```

What to need to do is new some objects without deleting, the objects will be released automatically
Define a struct, don't add a semicolon after it, initialize the struct object with list initialization, for example

```
    struct Coor
    {
      int x;
      int y;
    }
    Coor co{x:1,y:1};
```

Supports defining member functions. Defining a function declares that the function must be preceded by a function for instance:

```
  function void show_nothing()
  {
  }
```

Functions support overloading, that is, the function names can be the same, and the interpreter selects different same-name function by distinguishing their arguments. At the same time, the type of the actual parameter must be exactly same as parameter, otherwise an error will be reported. such as

```
  function void int print_int(int x)
  {
    std.cout(x);
  }
```

call function

`print_int (1.2);` will report an error as a result the argument should be transformed.

`print_int (cast <int> (1.2));`

Note that member function does not support overload

Pre-input

```
  $pre_input
  hello 123 $end
  program main
  {
    string str=std.input_string();
    int tmp=std.input_int();
  }
```

We can use the `$pre_input` to implement pre-input

IO

```
    std.cout(args);
```

args->expr1,expr2,expr3..,expr_n;

function:input expr1,expr2,expr3...,expr_n

```
  std.input_int();
  std.input_char();
  std.input_string();
  std.input_real();
```

I reckon you can learn the usage merely by their name.

Containers

string

merdog string supporting operations as follows

1. random visit
2. +=, + add a string to the string’s back

   for example:

   ```
   string tmp=”123”;
   string tmp2=”.334”;
   string v=tmp+tmp2;
   ```

tmp+=tmp2;

3. size() // return the characters count of a string
4. substr (startPos,length); //cut a string from startPos and return the string.

Vector

before using the vector, ensure you have added the “using vector” in the front of your program.

vector<Type>: create a vector container to contain Type’s elements

Suppose you have defined a vector variable called vec

- initialize a vector
  - ➔ vector<Type> vec={…}; //list initialize
  - ➔ vector<Type> vec(n); //initialize a vector with n elements, all the elements are initialized with default value
  - ➔ vector<Type> vec(n,v); // initialize a vector with n elements, all the elements are initialized with v
- insert and erase
  - ➔ vec.push_back(v); // push v to the back of the vector;
  - ➔ vec.pop_back(); // pop the back of the vector;
  - ➔ vec.insert(n,v);// Not recommended: it will be low-performance, insert v at vec[n], after that, vec[n] is v;
  - ➔ vec.clear(); // erase all elements of vec;
- others

  vec[n] // random visit

  .resize(n);// you can know the function from its name;

  .size();// obtain the count of elements

  .back(); //obtain the back elements’ value

Deque

deque has all operations that vector has, and in addition.

- ➔ front(); // obtain the front elements’ value
- ➔ push_front(v); //push v to the front the deque
- ➔ pop_front(); // …