西安电子科技大学
XIDIAN UNIVERSITY

计算机科学与技术学院
SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY
国家示范性软件学院
NATIONAL PILOT SCHOOL OF SOFTWARE ENGINEERING

面向对象设计与构造

Object-Oriented Design and Construction

课程案例—桌面计算器

6.1 A Desk Calculator

main.cpp

```
1   //#include <cstdlib>
2   #include <iostream>
3   #include <map>
4   #include <string>
5   #include <cctype>
6
7   using namespace std;
8
9   enum Token_value {
10    NAME, NUMBER, END, PLUS = '+', MINUS = '-', MUL = '*',
11    DIV = '/', PRINT = ';', ASSIGN = '=', LP = '(',
12    RP = ')'
13  };
14  Token_value curr_tok = PRINT;
15  double      number_value;
16  string      string_value;
17  map<string, double>    table;   // 符号表
18  int     no_of_errors;
19
20  double expr(bool get);
21
22  double error(const string& s)  // 打印出错信息，并统计出错次数
23  {
24    no_of_errors++;
25    cerr << "error: " << s << "\n";
26    return 1;
27  }
28
29  Token_value get_token()
30  {
31    char ch = 0;
32    cin >> ch;
33  //  cout << "ch in get_token = " << ch << endl;
34  //  do {  // skip whitespace except '\n'
35  //      if (!cin.get(ch)) return curr_tok = END;
36  //  } while(ch != '\n' && isspace(ch));
37    switch (ch) {
38      case 0:
39        return curr_tok = END;
40      case ';':
41      case '\n':
```

```
D:\02-教学工作\01-我教的课\C
(3.5+2)*3/(18-5)=
1.26923

r=3.5;
3.5
r*4=
14

pi*3=
9.42478
```

A Desk Calculator

- 怎样来读这个程序？可以分以下步骤阅读：
  - 背景材料；
  - 基础性定义：“桌面计算器”所使用的语言的文法（Grammar）；
  - 数据结构定义：主要是自定义类型；
  - 状态设置与隐涵的协同：全局变量；
  - 明显的协同：通过函数调用表现出来的程序结构；
  - 处理过程：通过语句或函数调用表现出来的控制结构；
  - 其他。

A Desk Calculator—背景资料

- “桌面计算器”从标准输入设备读入一个（数值计算）表达式，计算它的值后从标准输出设备输出；读入的也可以是一个赋值语句：左端是一个符号名，右端是表达式。
- 表达式中可以有四则运算符、括号、整数/实数值、已经赋值的符号名和预定义的符号常量（pi 和 e），也可以只有单个的整数/实数值。
- 发现输入内容与文法不符或将导致非法计算时，则从标准输出设备输出出错提示，并计算出错次数。

A Desk Calculator—背景资料

- The calculator consists of four main parts: a parser（解析程序）, an input function, a symbol table（符号表）, and a driver（驱动程序）.
- It is a miniature（小规模的）compiler in which the parser does the syntactic analysis（语法分析）, the input function handles input and lexical analysis（词法分析）, the symbol table holds permanent information（固定信息）, and the driver handles initialization, output, and errors.

A Desk Calculator—语言的文法

- 文法是关于这个语言要素的一个递归（Recursive）定义。
- 这个定义中有两类符号：*非终结符*（Nonterminal symbols）和终结符（Terminal symbols）。

```
program:
    END
    expr_list END
expr_list:
    expression PRINT
    expression PRINT expr_list
expression:
    expression + term
    expression - term
    term
```

```
term:
    term / primary
    term * primary
    primary
primary:
    NUMBER
    NAME
    NAME = expression
    - primary
    ( expression )
```

A Desk Calculator—自定义类型

- 有一个自定义类型 Token_value，它定义了代表各种终结符的标记（Token）值。

```
enum Token_value {
    NAME,           NUMBER,         END,
    PLUS = '+',     MINUS = '-',    MUL = '*',      DIV = '/',
    PRINT = ';',    ASSIGN = '=',   LP = '(',       RP = ')'
};
```

- 想一想：为什么定义成枚举类型？

A Desk Calculator—全局变量

- 定义了5个全局变量，先列出来，以后再确定它们的涵义和作用（注意它们在各个函数之间所起的、传递状态或值的协同作用）：

```
Token_value      curr_tok = PRINT;
double           number_value;
string           string_value;
map<string, double>    table;
int              no_of_errors;
```

A Desk Calculator—程序的结构

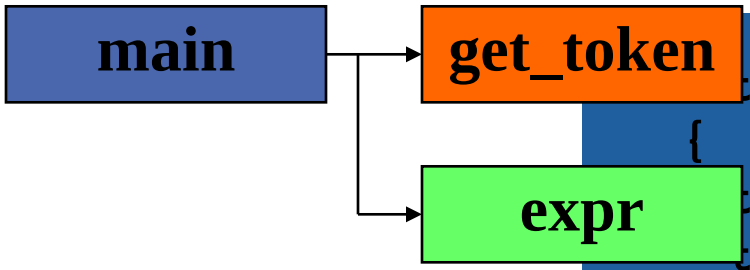
- 从 main 开始，寻找函数调用，得出程序的结构：

main

```
int main()
{
  table["pi"] = 3.14159265358979323385;
  table["e"]  = 2.71828182845904523354;
  while (cin) {
    get_token();
    if (curr_tok == END) break;
    if (curr_tok == PRINT) continue;
    cout << expr(false) << '\n';
  }
  return no_of_errors;
}
```

A Desk Calculator—程序的结构

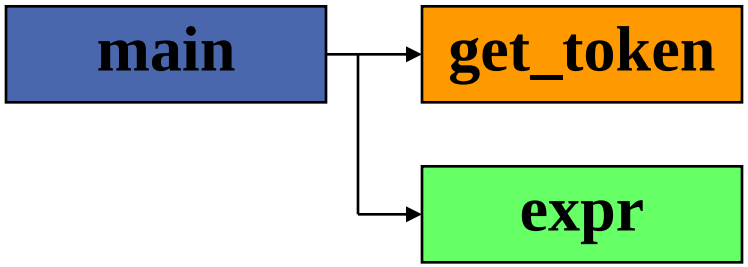
- 从 main 开始，寻找函数调用，得出程序的结构：




```
int main()
{
    table["pi"] = 3.14159265358979323385;
    table["e"]  = 2.71828182845904523354;
    while (cin) {
      get_token();
      if (curr_tok == END) break;
      if (curr_tok == PRINT) continue;
      cout << expr(false) << '\n';
    }
    return no_of_errors;
}
```

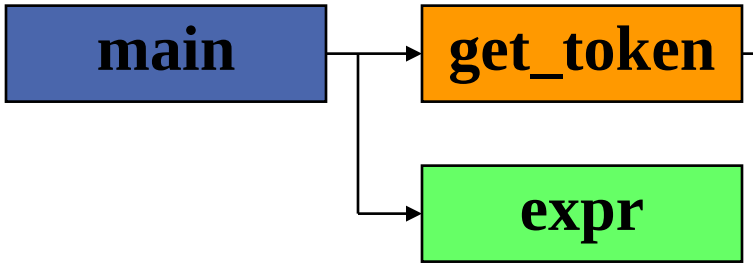
6.

- 从 main 开始，寻




```
Token_value get_token()
{
  char ch = 0;
  cin >> ch;
  switch (ch) {
    case 0:
      return curr_tok = END;
    case ';': case '*': case '/': case '+':
    case '-': case '(': case ')': case '=':
      return curr_tok = Token_value(ch);
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
    case '.':
      cin.putback(ch);  cin >> number_value;
      return curr_tok = NUMBER;
    default:                    // NAME, NAME = , or error
      if (isalpha(ch)) {
        cin.putback(ch); cin >> string_value;
        return curr_tok = NAME;
      }
      error("bad token");
      return curr_tok = PRINT;
    }
}
```

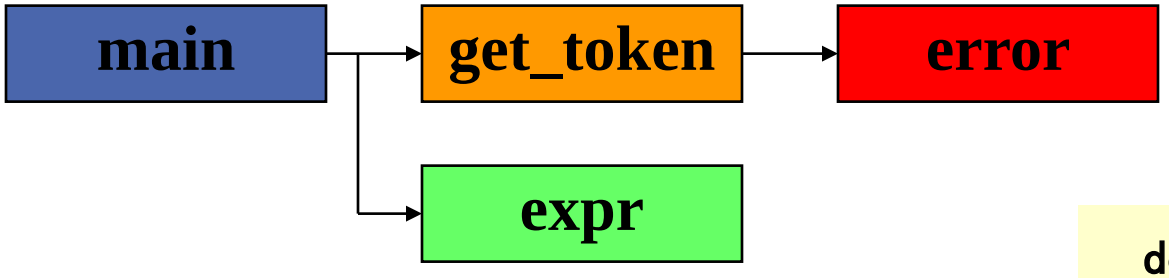
- 从 main 开始，寻




```
Token_value get_token()
{
  char ch = 0;
  cin >> ch;
  switch (ch) {
    case 0:
      return curr_tok = END;
    case ';': case '*': case '/': case '+':
    case '-': case '(': case ')': case '=':
      curr_tok = Token_value(ch);
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
    case '.':
      cin.putback(ch);  cin >> number_value;
      return curr_tok = NUMBER;
    default:                    // NAME, NAME = , or error
      if (isalpha(ch)) {
        cin.putback(ch); cin >> string_value;
        return curr_tok = NAME;
      }
      error("bad token");
      return curr_tok = PRINT;
    }
}
```

A Desk Calculator—程序的结构

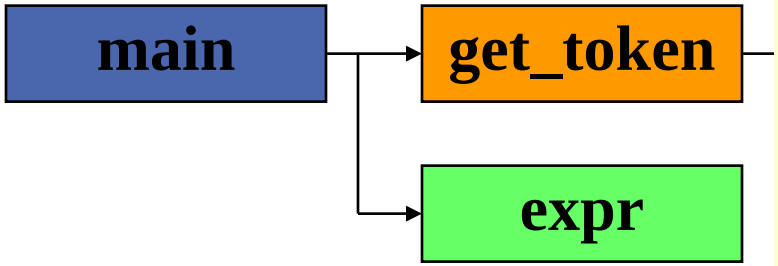
- 从 main 开始，寻找函数调用，得出程序的结构：




```
double error(const string& s)
{
  no_of_errors++;
  cerr << "error: " << s << "\n";
  return 1;
}
```

A Desk Calculator—程序的结构

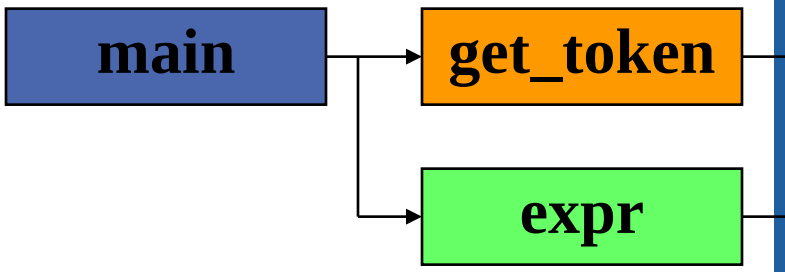
- 从 main 开始，寻找函数调用，得出程序的结构：




```
double expr(bool get) // add and subtract
{
  double left = term(get);
  for (;;)              // “forever”
    switch (curr_tok) {
      case PLUS:
        left += term(true);
        break;
      case MINUS:
        left -= term(true);
        break;
      default:
        return left;
    }
}
```

A Desk Calculator—程序的结构

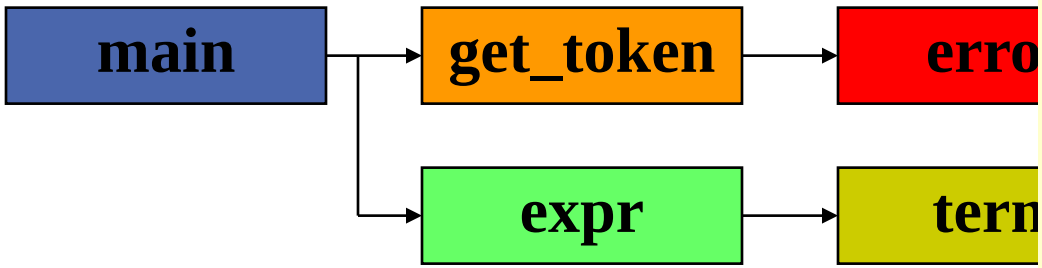
- 从 main 开始，寻找函数调用，得出程序的结构：




```
double expr(bool get) // add and subtract
{
    t = term(get);
  for (;;)                  // "forever"
    switch (curr_tok) {
    LUS:
        left += term(true);
        break;
      case MINUS:
        left -= term(true);
        break;
      default:
        return left;
    }
}
```

A Desk Calculator—程序的结构

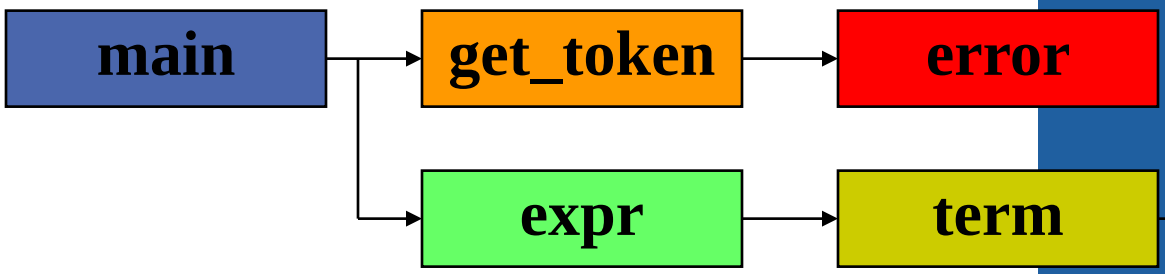
- 从 main 开始，寻找函数调




```
// multiply and divide
double term(bool get)
{
  double left = prim(get);
  for (;;)
    switch (curr_tok) {
      case MUL:
        left *= prim(true);
        break;
      case DIV:
        if (double d = prim(true)) {
          left /= d;
          break;
        }
        return error("divide by 0");
      default:
        return left;
    }
}
```

A Desk Calculator—程序的结构

- 从 main 开始，寻找函数调




```
// multiply and divide
double term(bool get)
{
  double left = prim(get);
  for (;;)
    switch (curr_tok) {
      case MUL:
        left *= prim(true);
        if (double d = prim(true)) {
          ;
          }
          return error("divide by 0");
      default:
        return left;
    }
}
```

lator—程序的结构

程序的结构：

```
double prim(bool get)
{
  if (get) get_token();
  switch (curr_tok) {
    case NUMBER:
      { double v = number_value;
        get_token();
        return v;
      }
    case NAME:
      { double& v = table[string_value];
        if (get_token() == ASSIGN) v = expr(true);
        return v;
      }
    case MINUS:    // unary minus
      return - prim(true);
    case LP:
      { double e = expr(true);
        if (curr_tok != RP) return error(" ) expected");
        get_token();  // eat ")"
        return e;
      }
    default:
      return error("primary expected");
    }
}
```

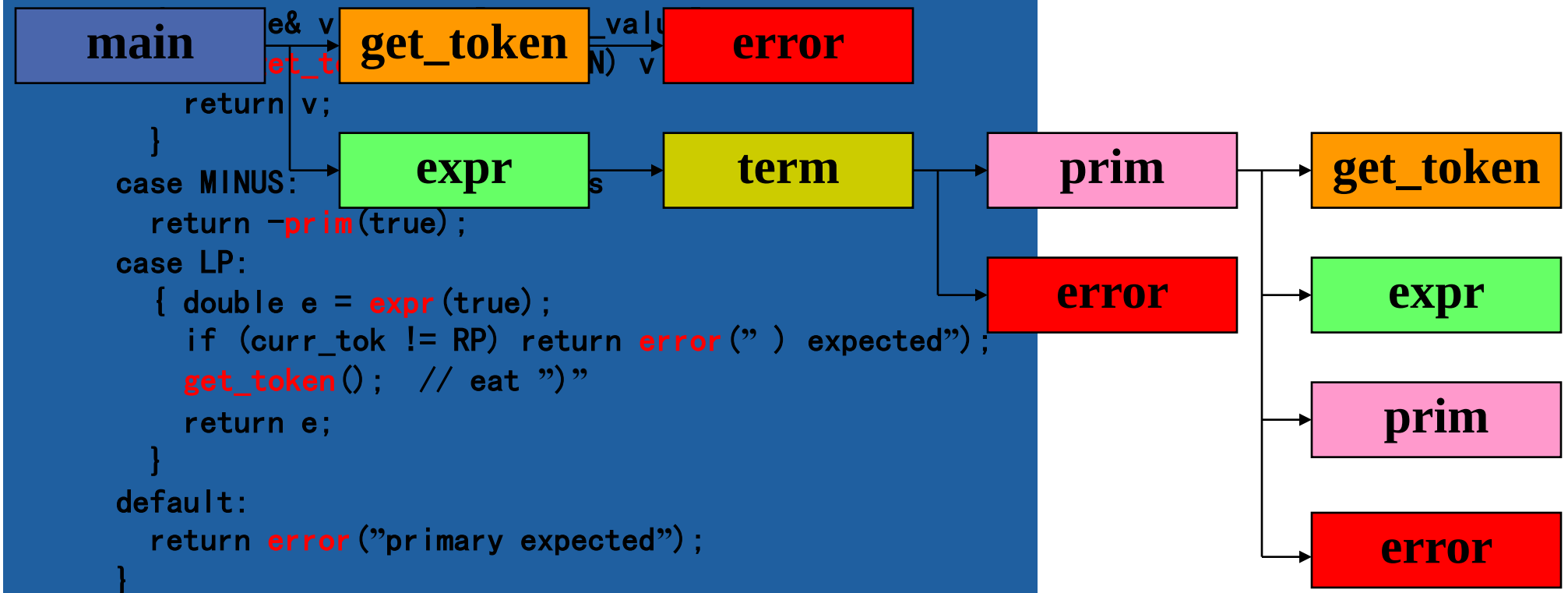
prim

error

A Desk Calculator—程序的结构

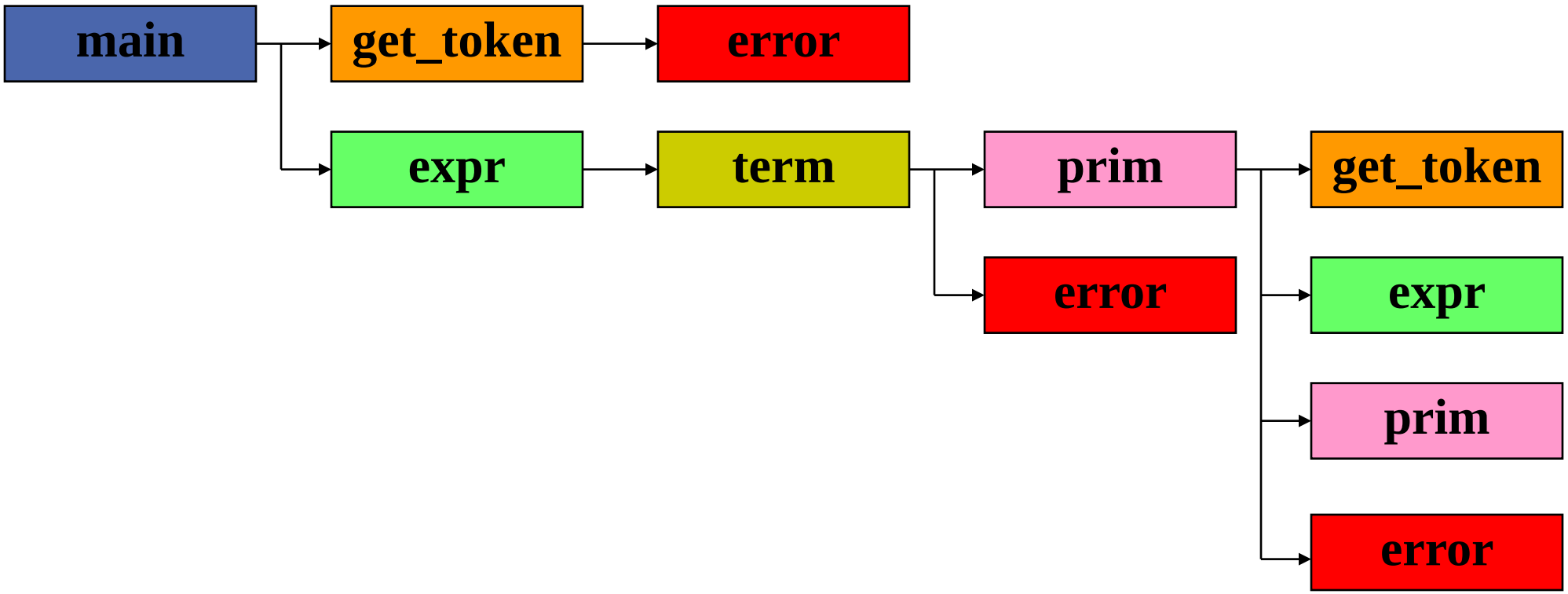
```
double prim(bool get)
{
  if (get) get_token();
  switch (curr_tok) {
    case NUMBER:
      double v = number_value;
      get_token();
      return v;
    }
    case NAME:
      e& v
      et_t
      return v;
    }
    case MINUS:
      return -prim(true);
    case LP:
      { double e = expr(true);
        if (curr_tok != RP) return error(" ) expected");
        get_token();  // eat ")"
        return e;
      }
    default:
      return error("primary expected");
    }
}
```

- 从main开始，寻找函数调用，得出程序的结构：



A Desk Calculator—程序的结构

- 从 main 开始，寻找函数调用，得出程序的结构：



- 选择语句 switch：若不 break和return，将走到后面的case；多个case可以复选；要养成写default的习惯。
- 用类型名可以求值。
- 选择语句 if：条件值非 0 时为真。
- 返回语句 return：出现了先赋值后返回。

```
Token_value get_token()
{
  char ch = 0;
  cin >> ch;
  switch (ch) {
    case 0:
      return curr_tok = END;
    case ';': case '*': case '/': case '+':
    case '-': case '(': case ')': case '=':
      return curr_tok = Token_value(ch);
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
    case '.':
      cin.putback(ch);  cin >> number_value;
      return curr_tok = NUMBER;
    default:                    // NAME, NAME = , or error
      if (isalpha(ch)) {
        cin.putback(ch); cin >> string_value;
        return curr_tok = NAME;
      }
      error("bad token");
      return curr_tok = PRINT;
    }
}
```

A Desk Calculator—处理过程

- 二元算符与赋值运算符相结合：+=；-=；*=；/=……

  （x @= y means x = x @ y）

- 循环（注意结束条件）。
- break 的作用区域只有一层，所以只是跳出 switch，而没有跳出 for 循环。
- 在 if 语句的条件中可定义变量、调用函数和赋值。

```
// multiply and divide
double term(bool get)
{
  double left = prim(get);
  for (;;)
    switch (curr_tok) {
      case MUL:
        left *= prim(true);
        break;
      case DIV:
        if (double d = prim(true)) {
          left /= d;
          break;
        }
        return error("divide by 0");
      default:
        return left;
    }
}
```

A

- 这里用string_value当索引来取出给符号名赋的值。那么，如果这个符号名还没有赋值，怎样赋值？
- 为什么两个 case 中的变量 v 有不同的类型？

```
double prim(bool get)
{
  if (get) get_token();
  switch (curr_tok) {
    case NUMBER:
      { double v = number_value;
        get_token();
        return v;
      }
    case NAME:
      { double& v = table[string_value];
        if (get_token() == ASSIGN) v = expr(true);
        return v;
      }
    case MINUS:    // unary minus
      return -prim(true);
    case LP:
      { double e = expr(true);
        if (curr_tok != RP) return error(" ) expected");
        get_token();  // eat ")"
        return e;
      }
    default:
      return error("primary expected");
    }
}
```

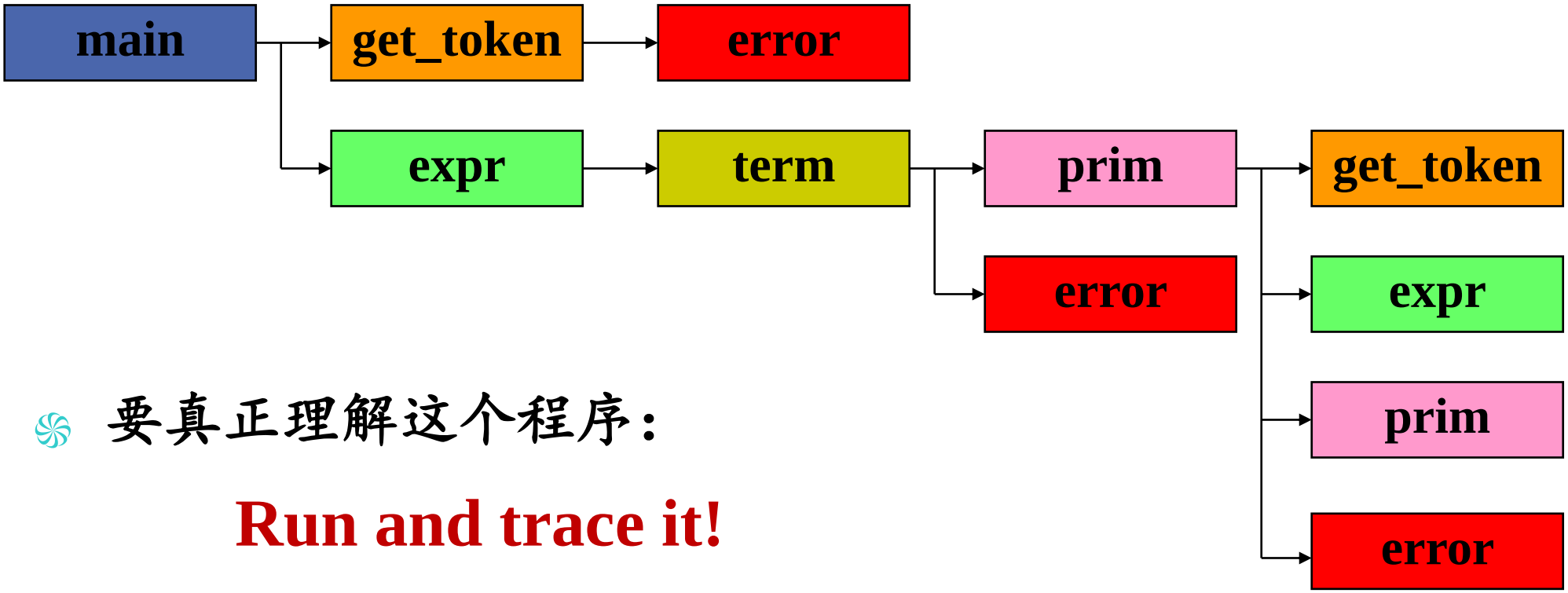
A Desk Calculator—其它

```
Token_value      curr_tok = PRINT;
double           number_value;
string           string_value;
map<string, double>     table;
int              no_of_errors;
```

- 这时再确定全局变量的作用：
  - **curr_tok**：在 get_token 中设置；在 expr、term、prim 中的 switch 中使用。它表示的是当前读入的标记的类别，用来控制分类别的求值及其他处理。
  - **number_value**：在 get_token 中设置；在 prim 中使用。它表示的是当前读入的数值字面值。
  - **string_value**：在 get_token 中设置；在 prim 中使用。它表示的是当前读入的符号名，用来在 table 中查找对应的数值。
  - **table**：在 prim 中设置；在 prim 中使用。它表示的是已经读入的符号名与对应数值，符号名可以增加，对应数值通过引用类型隐涵地赋值。
  - **no_of_errors**：在 error 中设置；在 main 中使用。它表示的是已经发生的错误数量。

A Desk Calculator—其它

- 这个程序出现了直接递归调用（在 prim 中调用了 prim）和间接递归调用（在 prim 中调用了 expr），要注意递归终止条件。



- 要真正理解这个程序：

Run and trace it!