

1. Write the number of the definition on the right next to the term it defines.

- | | |
|---------------------------------|---|
| (a) class <u>3</u> | (1) An operation that makes two objects have values that compare equal.. |
| (b) constructor <u>4</u> | (2) The region of program text (source code) in which a name can be referred to. |
| (c) container <u>12</u> | (3) A user-defined type that may contain data members, function members, and member types. |
| (d) copy <u>1</u> | (4) An operation that initializes an object. Typically establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor). |
| (e) invariant <u>10</u> | (5) (1) a value used to identify a typed object in memory; (2) a variable holding such a value. |
| (f) overload <u>7</u> | (6) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value. |
| (g) reference <u>6</u> | (7) Define two functions or operators with the same name but different argument (operand) types. |
| (h) pointer <u>5</u> | (8) an operation that transfers a value from one object to another, leaving behind a value representing "empty." |
| (i) scope <u>2</u> | (9) Something that defines a set of possible values and a set of operations for an object. |
| (j) byte <u>11</u> | (10) Something that must be always true at a given point (or points) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement. |
| (k) type <u>9</u> | (11) The basic unit of addressing in most computers. |
| (l) move <u>8</u> | (12) An object that holds elements (other objects). |

2. Consider the following code fragment for the user-defined type, `Token`:

```
1 struct Token{
    char kind;
3   double value;
    string varname;
5   Token(char ch) :kind(ch), value(0) { }
    Token(char ch, double val) :kind(ch), value(val) { }
7   Token(char ch, string val) :kind(ch), varname(val) { }
};
```

(a) What kind values are used to declare a variable?

ANS: A variable is created with the `Token(name,s)` constructor, where `name` is a `const char` set to 'a' and `s` is the `string`. Also, the program officially declares a variable with this member function of the `Symbol_table` class:

```
void declare(string s, double d, bool b)
2 {   for (int i = 0; i<int(names.size()); ++i)
        if (names[i].name == s) error("declare:␣existing␣name␣",s);
4     names.push_back(Variable(s,d,b));
}
```

The function searches through the existing variable names and, if it finds `s` among them, throws an error, otherwise it dutifully constructs a new variable name `s` with its value `d` and bool `b` to determine whether its a constant or not, and pushes this onto the `symbol_table`'s vector, `names` or `Variables`.

- (b) Describe in detail the ways in which the constructors for `Token` are used.

ANS: In addition to the rather complex construction of a variable token described above, an operator token is constructed with `Token(ch)` where `ch` is a variable of type `char` and represents an operation like `'+'`, or a keyword for a function operator like `let (let = 'L')` `constant (constant = 'C')`, `sqrt (sqrt = 'S')` and so on. The value of an operator token is given the default value `= 0` and the varname for an operator token is just ignored (the empty string). A number token is constructed with `Token(number, val)` where `number` is the `const char`, `'8'` and `val` is a variable of type `double`, the value of the number token.

- (c) Describe how to change `Token` from a `struct` to a `class`.

ANS: Consider it done (note the use of the word `public`. The default setting for a `class` is `private`. We could make the data members private, but then accessors would likely be needed:

3. Consider the following code fragment for the user-defined type, `Token_stream`

```

1 struct Token_stream {
    bool full;
3     Token buffer;
    istream & str;
5     Token_stream(istream & arg) : str(arg), full(0), buffer('␣') { }
    Token get();
7     void unget(Token t) { buffer=t; full=true; }
    void ignore(char);
9 };

```

- (a) Describe the variable `buffer`. What is it? What is it for? Give an example of how it is used.

ANS: The `token_stream` `buffer` is one of two (or three) member variables of the `Token_stream` class. It is a variable of type `Token` and is for holding a `Token` that can't be processed immediately. For instance, if the `term()` function's call to `get()` (`Token t = str.get()`) gets a `(+,0,"")` token, then that token goes into the `Token_stream` `buffer` and control passes back to `expression()` which will get the next term before retrieving the plus token from the buffer and doing the addition of terms. There are a variety of other circumstances where the `Token_stream` `buffer` is used.

- (b) Describe the variable `str`. What is it? What is it for? Give an example of how it is used.

ANS: The variable `str` is a variable of type `istream&` (a reference to an `istream`). It is a third member variable of the class `Token_stream` that was added to make streaming sources more flexible. For instance, we could create from the start (in `main()`), an input file stream and assign it to an `istream` like so:

```

    ifstream fs("inExpr.txt");
    istream& istrm = fs;

```

and then construct `Token_stream ts`, and pass that by reference each time one function calls another that needs it. Since this is a member variable of `Token_stream ts`, `istrm` gets passed along with `ts`. If it's pointing to a file `inExpr.txt` that contains, say, `"5 + 12;"` then `main()` calls `calculate(ts)`, which calls `statement(ts)` by printing its output to the console (and/or to a file?), then `expression(ts)` is called and its output is assigned to a variable `left` of type `double`, to be returned to `statement()`, and `expression()` then gets the term 5 by calling `primary(ts)` through `term(ts)` and adds to that to the term (stored in the `double left`), replacing `left` now with the sum, 17, at which point `expression()` and encounters the token `(';',0,"")`, which it puts in the `Token_stream` `buffer` with `ts.unget(t)` (aka `put_back()`), then `left=17` is returned to `statement()` which recovers the print token from the buffer and returns 17 to `calculate()`, printing it to the console and looking (with `while(ts.str)`) to see if there's more input or not. If we're at the end of the file (`eof`), then that's all folks, otherwise we start again.

4. Consider the following code fragment for the member function `get()`.

```

1 Token Token_stream::get() {
    if (full) { full=false; return buffer; }
3   char ch;
    str >> ch;
5   if (! str) return(Token(quit));
    switch (ch) {
7   case '(' : case ')' : case '+' : case '-': case '*':
    case '/' : case '%' : case ';' : case '=' : case ',':
9       return Token(ch);
    case '.' : case '0' : case '1' : case '2' : case '3' :
11  case '4' : case '5' : case '6' : case '7' : case '8' :
    case '9' :
13  {   str.unget();
        double val;
15  str >> val;
        if (! str) error("Bad token");
17  return Token(number, val);
    }
19  default:
        if (isalpha(ch) || ch == '_') {
21  string s; s += ch;
            while(str.get(ch) &&
23  (isalpha(ch) || isdigit(ch) || ch == '_'))
                s += ch;
25  str.unget();
            if (! str) error("Bad token");
27  if (s == "let") return Token(let);
            if (s == "const") return Token(constant);
29  if (s == "reset") return Token(reset);
            if (s == "sqrt") return Token(sqroot);
31  if (s == "pow") return Token(power);
            if (s == "help") return Token(help);
33  if (s == "quit" || s == "exit")
                return Token(quit);
35  return Token(name, s);
        }
37  error("Bad token");
        return Token('_');
39  }
}

```

(a) What does `get()` get if `full==true`?

ANS: Whatever token is stored in `buffer`.

(b) What is `str` here?

ANS: It's none other than the reference to an `istream` which is a member variable of `Token_stream`—after all, we are in the scope of `Token_stream` here.

(c) What is the purpose of `str.unget()` on line 14?

ANS: If you encountered a character that could be the first character of a double, put it back with `unget()` (this is the console's `unget()`, not `Token_stream`'s) and then use the `istream& str` to get the whole double in one go.

(d) Describe what happens in the `default` case. How does it provide for the declaration of a new variable? How does it handle a built-in function like `pow()`? How does it recognize an existing variable in

symbol_table?

ANS: You get to the default case if `buffer` is empty the character `get()` gets is not a simple arithmetic operator or a number token. In that case we check to see if the next character is either alphabetic or “_”, in which case we start a string `s` with it and keep concatenating to `s` while characters are alphabetic, digit or “_”. As soon as we get something other than those things we put it back in the `istream` buffer and check whether `s==` to one of the “special” operator token types like `let`, `const`, `sqrt`, `pow`, or `quit`, in which cases we construct the appropriate special operator token and return it...otherwise it’s a variable token, so we construct a variable token and return that.

5. Rewrite the calculator program to incorporate the variable

```
istream& str;
```

as a member variable of `Token_stream`, rather than passing it from function to function by reference. Email your complete working code to ghagopian@collegeofthedesert.edu.

6. Consider the following code fragment for handling Variables in the calculator.

```

1 struct Variable {
2     string name;
3     double value;
4     bool immutable;
5     Variable(string n, double v, bool b) :
6         name(n), value(v), immutable(b) { }
7 };
8
9 // The active variables.
10 class Symbol_table {
11     vector<Variable> names;
12 public:
13
14 double get(string s) {
15     for (int i = 0; i<int(names.size()); ++i)
16         if (names[i].name == s) return names[i].value;
17     error("get: undefined name",s);
18     return 0.0;
19 }
20
21 void set(string s, double d)
22 { for (int i = 0; i<=int(names.size()); ++i)
23     if (names[i].name == s) {
24         names[i].value = d;
25         return;
26     }
27     error("set: undefined name",s);
28 }

```

(a) Describe the constructor for a `Variable`. How does it work?

ANS: `Variable(string n, double v, bool b)` uses the initiation list to set the values of the member variables `name`, `value`, and `immutable`. Notably, `Variable` is global.

(b) What is `Symbol_table`. What purpose does it serve?

ANS: The class `Symbol_table` has a private data member which is `vector<Variable> names` and contains all the variables that have been declared using the `let` token and the `declaration()` function. There are member functions `get()` and `set()` and `declare()` for keeping track of new and old variables and their values.

- (c) Why does this `get()` function not collide with `Token_stream`'s `get()` function?
ANS: They have different scopes. One is a member of the `Symbol_table` class and the other is a member of the `Token_stream` class.
- (d) Describe in detail how `get()` works and what its purpose is.
ANS: This `get` is called when `primary()` encounters a `name` token. It takes a string as an argument and searches through the vector or `Variables`, `names`, for that string. If it finds the string in `names` it returns its value with `names[i].value` to `primary()`, if it doesn't find it it throws an error("get: undefined name ", `s`).
- (e) Describe in detail how `set()` works and what its purpose is.
ANS: `Symbol_table`'s `set()` function is called by the `declaration()` function and takes two parameters: `string s` and `double d`, then searches through the `names` vector for the `Variable` with name `s`. If it finds `s` it sets the value of that `Variable` with `d` and returns. If it doesn't find `s` in `names` it throws an error with the message "set: undefined name ".
- (f) Could these `get()` and `set()` functions be made member functions of the `struct Variable`? Discuss.
ANS: Yes, but that would be a less clear way to handle these functions, since it's `Symbol_table`'s job to manage `Variables`.