

Tutorial 4

Review for “class” concepts and “TokenScanner” Applications

Sep. 19, 2022

Lai Wei (USTF)

(SDS, 120090485@link.cuhk.edu.cn)

Objectives today

1. Terminal knowledges
2. Class & objects basics
3. Class constructors
4. Class attributes & functions
5. Class member scopes
6. Class overload operators
7. One example with class applications: Token Scanner
8. How to learn this course well (continued)
9. Q & A time

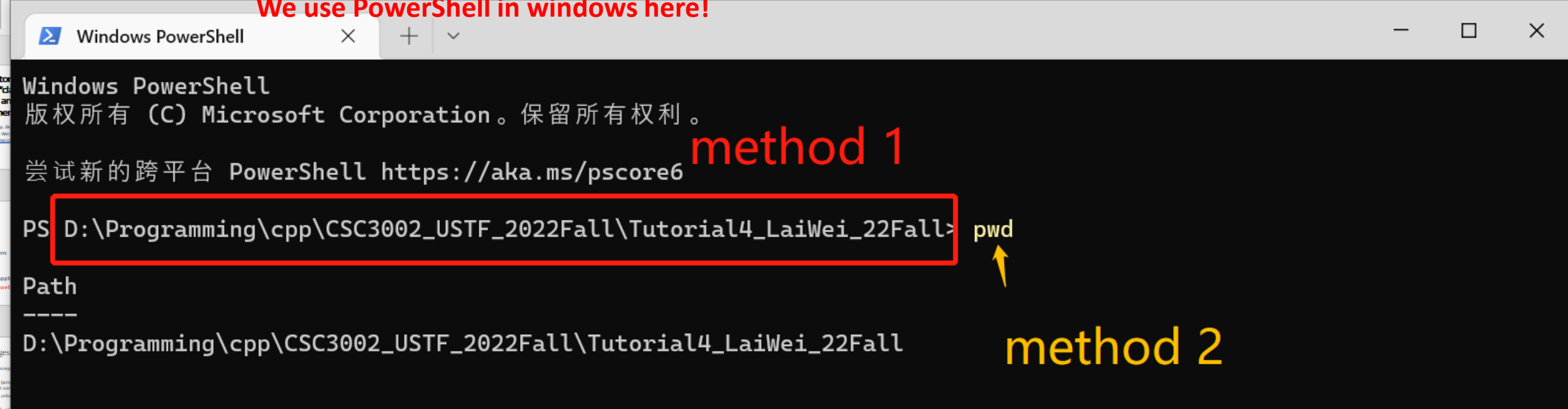
1. Terminal knowledges

- Actually, our terminal has a concept of “**current working directory**” (当前目录)
- The terminal can only see files (programs) under current working directory, and the environment variable “Path”
- A **relative path** (相对路径) refers to a location that is **relative to a current directory**.
- An **absolute path** (绝对路径) always **contains the root element and the complete directory** list required to locate the file.

1. Terminal knowledges

- Actually, our terminal has a concept of “**current working directory**” (当前目录)
- How to get this “**current working directory**” ? (windows, mac same)

We use PowerShell in windows here!



```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
尝试新的跨平台 PowerShell https://aka.ms/pscore6
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall> pwd
Path
----
D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall
```

The screenshot shows a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The terminal content includes the standard PowerShell header, a red box around the prompt and the current directory path, and the output of the `pwd` command. Annotations include "method 1" in red text pointing to the directory path in the prompt, and "method 2" in yellow text pointing to the `pwd` command and its output.

1. Terminal knowledges

- A **relative path** (相对路径) refers to a location that is **relative to the current directory**.

```
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code> ls
```

```
目录: D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code
```

Mode	LastWriteTime	Length	Name
d-----	10/6/2022 11:48 PM		.vscode
d-----	3/2/2021 6:52 PM		exercise
-a-----	10/5/2022 11:54 PM	342	1.cpp
-a-----	10/5/2022 11:54 PM	263	1.py
-a-----	10/6/2022 12:11 AM	926	2.cpp
-a-----	10/6/2022 12:12 AM	982	3.cpp
-a-----	10/7/2022 12:15 AM	752	4.cpp
-a-----	10/6/2022 7:38 PM	127	5.cpp
-a-----	10/7/2022 10:00 AM	2100	6

“ls”: list all files in current working directory

```
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code> ls
```

```
目录: D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code
```

Mode	LastWriteTime	Length	Name
d-----	10/6/2022 11:48 PM		.vscode
d-----	3/2/2021 6:52 PM		exercise
-a-----	10/5/2022 11:54 PM	342	1.cpp
-a-----	10/5/2022 11:54 PM	263	1.py
-a-----	10/6/2022 12:11 AM	926	2.cpp
-a-----	10/6/2022 12:12 AM	982	3.cpp
-a-----	10/7/2022 12:15 AM	752	4.cpp
-a-----	10/6/2022 7:38 PM	127	5.cpp
-a-----	10/7/2022 12:02 AM	2103	6.cpp
-a-----	10/7/2022 12:08 AM	1886	7.cpp
-a-----	10/7/2022 12:01 AM	57383	a.exe

- So basing that we're in "D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code" directory,
- These files have **relative path** to "1.cpp" "1.py" "2.cpp"

1. Terminal knowledges

- An **absolute path** (绝对路径) always **contains the root element and the complete directory** list required to locate the file.
- “D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code\1.cpp”
- “D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code\1.py”
-

1. Terminal knowledges

- The terminal can only see files (programs) under current working directory, and the environment variable “Path”

```
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code> g++ 2.cpp -o hello.exe
```

“g++” is in the environment variable path, so we can call it by relative path

“2.cpp” is in the current working directory, so we can call it by relative path

“hello.exe” is a relative path, create executable under current working dir

Absolute path form:

```
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code> C:\Qt\Qt5.12.10\Tools\mingw730_64\bin\g++.exe D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code\2.cpp -o D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code\hello.exe
```


1. Terminal knowledges

- Changing directories: by “cd <dir>” command
 - “..” means upper level directory
 - <dir> can be a relative path or an absolute path

```
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall> cd code
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall\code> cd ..
PS D:\Programming\cpp\CSC3002_USTF_2022Fall\Tutorial4_LaiWei_22Fall> cd D:\Programming\cpp\CSC3002_USTF_2022F
all\Tutorial4_LaiWei_22Fall\code
```

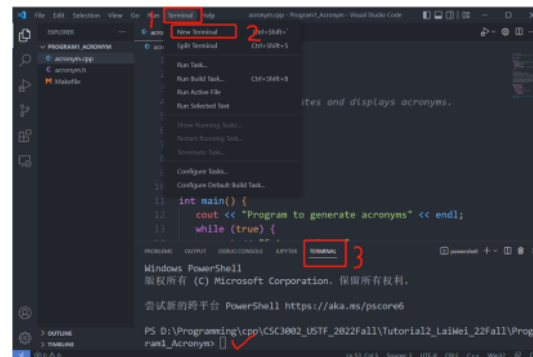
1. Terminal knowledges

- Recall this in tutorial 2: When you open the terminal in VS Code, it automatically change the current working directory at your opened folder.
- So I forced you to do like this, so we won't have problem in "relative path". If you are in wrong working directory, you can't compile the code "g++ helloworld.cpp" because it can't find the "helloworld.cpp".

5.1. Run the code via command lines

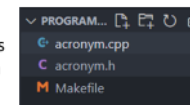
a) Compile by pure command lines, with "g++" compile command

- Open a terminal in the current code folder.

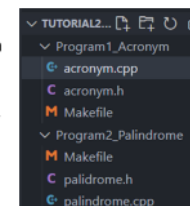


The screenshot shows the VS Code interface with a terminal window open. The terminal prompt is `PS D:\Programming\cpp\CSC3002_16TF_2022Fall\Tutorial2_LaiMei_22Fall\Program1_Acronym >`. The user has entered the command `g++ acronym.cpp`, which has been executed successfully, as indicated by the green checkmark and the output `Program to generate acronyms`. Red boxes highlight the terminal window and the command.

Correct: Makefile is in current program folder workspace, no path problem.



Wrong: Makefile is NOT in program folder workspace, may bring relative-path problem.



2. Class & objects basics

```
1 class MyClass:
2     def __init__(self) -> None:
3         self.var1 = None
4         self.var2 = None
5         self.__var3 = 5.0
6
7     def print(self):
8         print(self.var1, self.var2)
9
10    def __increment(self):
11        self.__var3 = self.__var3 + 1
12
```

```
1 class MyClass{
2
3     public: ★
4         MyClass(){};
5
6     ★ int var1, var2;
7     ★ void print(); // only prototype here
8
9     private:
10        float var3 = 5;
11    ★ void increment(){
12        var3 = var3 + 1;
13    }
14
15 }; ★
16
17 // Outside of the class definition
18 void MyClass::print(){
19     ★ std::cout << var1 << " " << var2 << std::endl;
20 }
```

3. Class constructors

- C++ class can have multiple constructors, with different parameter list.

```
1 class MyClass{
2 public:
3     MyClass(){}; // empty constructor
4     MyClass(int i, int j, float k){
5         var1 = i;
6         var2 = j;
7         var3 = k;
8     }
9     MyClass(float k){
10        var3 = k;
11    }
```

```
25 int main(){
26     MyClass m2();
27     MyClass m3(1, 2, 3.4);
28     MyClass m4(5.6);
29     return 0;
30 }
```

3. Class constructors

- E.g., the `std::string` class.

public member function

`std::`**string::string**

<string>

C++98 C++11 ?

```
default (1) string();
copy (2) string (const string& str);
substring (3) string (const string& str, size_t pos, size_t len = npos);
from c-string (4) string (const char* s);
from buffer (5) string (const char* s, size_t n);
fill (6) string (size_t n, char c);
range (7) template <class InputIterator> string (InputIterator first, InputIterator last);
initializer list (8) string (initializer_list<char> il);
move (9) string (string&& str) noexcept;
```

3. Class constructors

- Instantiate an instance.

```
1  int main(){  
2      MyClass m0; // OK, only declare a variable here  
3      m0(5, 6, 7.8); // WRONG! Constructor should be called  
4                      // same time as declaration  
5  
6      MyClass m1(); // OK, called MyClass() constructor  
7      MyClass m2 = MyClass(); // OK, called MyClass() constructor  
8      MyClass m3(8.0); // OK, called MyClass(float k) constructor  
9  }
```

4. Class attributes & functions

- Attributes are basically variables that belongs to the class.
- No need to use grammar such as “self.” in the class.

```
12
13     int var1, var2;
14     void print(); // only prototype here
15
16 private:
17     float var3 = 5;
18     void increment(){
19         var3 = var3 + 1;
20     }
21 };
22
```

4. Class attributes & functions

- If member function parameter has the same name as the class attribute, use “this->x” to distinguish them.
- Why not “this.x”? Because “this” is a pointer pointing to this instance.

```
1  class MyClass{
2  public:
3      MyClass(){}; // empty constructor
4
5      MyClass(int var1, int var2, float var3){
6          this->var1 = var1;
7          this->var2 = var2;
8          this->var3 = var3;
9      }
10
11     MyClass(float var3){
12         this->var3 = var3;
13     }
```


4. Class attributes & functions

- You can declare the function inside the class definition (.h file), and implement it outside of the class (.cpp file)
- or you can implement it in class definition as well. (not recommend)

```
1  class MyClass{
2
3  public:
4      MyClass(){};
5
6      int var1, var2;
7      void print(); // only prototype here
8
9  private:
10     float var3 = 5;
11     void increment(){
12         var3 = var3 + 1;
13     }
14
15 };
16
17 // Outside of the class definition
18 void MyClass::print(){
19     std::cout << var1 << " " << var2 << std::endl;
20 }
```

5. Class member scopes

- In C++, there are three access specifiers:
- `public` - members are accessible from outside the class
- `private` - members cannot be accessed (or viewed) from outside the class. **By default, attributes are private!**
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

```
3  class MyClass{
4      int var4 = 0; // private
5
6  public:
7      MyClass(){}; // empty constructor
8      MyClass(int i, int j, float k){
9          var1 = i;
10         var2 = j;
11         var3 = k;
12     }
```

5. Class member scopes

```
1 class MyClass{
2
3 public:
4     MyClass(){};
5
6     int var1, var2;
7     void print(); // only prototype here
8
9 private:
10    float var3 = 5;
11    void increment(){
12        var3 = var3 + 1;
13    }
14
15 };
16
17 // Outside of the class definition
18 void MyClass::print(){
19     std::cout << var1 << " " << var2 << std::endl;
20 }
```

```
1 int main(){
2     MyClass m1(5, 6, 7.8);
3     std::cout << m1.var1 << " " << m1.var2 << std::endl;
4     std::cout << m1.var3 << std::endl;
5 }
6
```

```
4.cpp:28:21: error: 'float MyClass::var3' is private within this context
    std::cout << m1.var3 << std::endl;
                        ^~~~~
4.cpp:19:18: note: declared private here
    float var3 = 5;
                ^
```

6. Class overload operators

- Customizes the C++ operators for operands of user-defined types.
- What operator can be overloaded?

any of the following operators: + - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &= |=
<< >> >>= <<= == != <= >= <=> (since C++20) && || ++ -- , ->* -> () []

- How to do that?
 - as a member function
 - as a non-member function (free function)
 - As a “friend” function

6. Class overload operators

	as a member function	as a non-member function (free function)	As a “friend” function
Definition where?	In class definition	Out class, or together with implementation	In class definition
Implementation where?	In class, with definition / Out class, both OK	Out of class definition	Out of class definition
prototype	Unary operator: 0 parameter. Binary operator: 1 parameter.	Unary operator: 1 parameter. Binary operator: 2 parameter.	Same as free function prototype

- <https://en.cppreference.com/w/cpp/language/operators>

6. Class overload operators

Expression	As member function	As non-member function	Example
@a unary, prefix	(a).operator@ ()	operator@ (a)	!std::cin calls std::cin.operator!()
a@b binary	(a).operator@ (b)	operator@ (a, b)	std::cout << 42 calls std::cout.operator<<(42)
a=b *assignment	(a).operator= (b)	cannot be non-member	Given std::string s ;, s = "abc"; calls s.operator=("abc")
a(b...) *call	(a).operator()(b...)	cannot be non-member	Given std::random_device r ;, auto n = r(); calls r.operator()
a[b] *indexing (at)	(a).operator[](b)	cannot be non-member	Given std::map<int, int> m ;, m[1] = 2; calls m.operator[](1)
a-> *get pointer's attribute	(a).operator-> ()	cannot be non-member	Given std::unique_ptr<S> p ;, p->bar() calls p.operator->()
a@ unary, postfix	(a).operator@ (0)	operator@ (a, 0)	Given std::vector<int>::iterator i ;, i++ calls i.operator++(0)

in this table, @ is a placeholder representing all matching operators: all prefix operators in @a, all postfix operators other than -> in a@, all infix operators other than = in a@b

Member function, defined and implement in the class

Member function, defined in class and implement out

Free function to overload operator<<

```
1 class BoolNumber{
2 public:
3     // constructor, assign value 0 / 1 to the instance
4     BoolNumber(int v){
5         if (v == 0 || v == 1)
6             value = v;
7         return;
8         // raise error if v is not 0 / 1
9         throw std::invalid_argument("v can only be 0 or 1!!!");
10    }
11    // default constructor, assign 0 to the instance
12    BoolNumber(){ BoolNumber(0); }
13
14    int value;
15
16    BoolNumber operator+(BoolNumber& rhs){
17        BoolNumber ans;
18        ans.value = std::min(1, this->value+rhs.value);
19        return ans;
20    }
21
22    BoolNumber& operator*(BoolNumber& rhs);
23 };
24
25 BoolNumber& BoolNumber::operator*(BoolNumber& rhs){
26     this->value = this->value*rhs.value;
27     return (*this);
28 }
29
30 std::ostream& operator<<(std::ostream& os, BoolNumber rhs){
31     os << rhs.value;
32     return os;
33 }
```

DEMO in BoolNumber.cpp

- Boolean Algebra: only 0 and 1.

Figure 1. Basic logic truth tables

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Table

A	B
0	1
1	0

Motivation

7. Token Scanner: From previous years' slides

- Many applications need to divide a string into words, or more generally, into **tokens** (i.e. logical units that may be larger than a single character).
- Given that the problem of dividing a string into individual tokens comes up so frequently in applications, it is useful to build a library package that takes care of that task. The primary goal is to build a package that is **simple** to use but also **flexible** enough to meet the needs of a variety of clients.

Design

- ✓ Task 1: **Associate** the token scanner with a source of tokens, which might be a string, an input stream, etc.
- ✓ Task 2: **Retrieve** individual tokens from the source of tokens and deliver them one at a time.
- ✓ Task 3: **Test** whether the token scanner has any tokens left to process.

pseudocode - reading tokens from a scanner:

```
Set the input for the token scanner to be some string or input stream.  
while (more tokens are available) {  
    Read the next token.  
}
```

Design

- ✓ TokenScanner should define tokens. What should be considered as a token?
 - ✓ a word in a string?
 - ✓ a single character?
 - ✓ a punctuation mark?
 - ✓ a space?
- ✓ Different applications define tokens in different ways
 - ✓ TokenScanner class must give the client some control over what types of tokens are recognized, e.g. whether a space should be recognized as a token.

Design

Methods in the **TokenScanner** Class

<code>scanner.setInput(str) or scanner.setInput(infile)</code> Sets the input for this scanner to the specified string or input stream.
<code>scanner.hasMoreTokens()</code> Returns true if more tokens exist, and false at the end of the token stream.
<code>scanner.next token()</code> Returns the next token from the token stream, and "" at the end.
<code>scanner.saveToken(token)</code> Saves token so that it will be read again on the next call to nextToken.
<code>scanner.ignoreWhitespace()</code> Tells the scanner to ignore whitespace characters.
<code>scanner.scanNumbers()</code> Tells the scanner to treat numbers as single tokens.
<code>scanner.scanStrings()</code> Tells the scanner to treat quoted strings as single tokens.

Demo

Ignoring white space:

```
Test program for the TokenScanner class
Input line: hello!world!
"hello"
"! "
"world"
"! "

Input line: 116010000@link.cuhk.edu.cn
"116010000"
"@ "
"link"
"."
"cuhk"
"."
"edu"
"."
"cn"

Input line: today is Monday
"today"
"is"
"Monday"
```

Not ignoring white space:

```
Test program for the TokenScanner class
Input line: hello!world!
"hello"
"! "
"world"
"! "

Input line: 116010000@link.cuhk.edu.cn
"116010000"
"@ "
"link"
"."
"cuhk"
"."
"edu"
"."
"cn"

Input line: today is Monday
"today"
" "
"is"
" "
"Monday"
```

Demo

Scan single digits:

```
Input line: pi = 3.1415
"pi"
"="
"3"
" "
"."
"1"
"4"
"1"
"5"
```

Scan the numbers as a whole:

```
Test program for the TokenScanner class
Input line: 2020/3/8 Women's Day
"2020"
"/"
"3"
"/"
"8"
"Women"
"'"
"s"
"Day"
```

Implementation: constructor

In tokenscanner.h:

```
/*
 * Constructor: TokenScanner
 * Usage: TokenScanner scanner;
 *         TokenScanner scanner(str);
 * -----
 * Initializes a scanner object. The initial token stream comes from
 * the string str, if it is specified. The default constructor creates
 * a scanner with an empty token stream.
 */

TokenScanner();
TokenScanner(std::string str);
```

In tokenscanner.cpp:

```
TokenScanner::TokenScanner() {
    ignoreWhitespaceFlag = false;
    singleDigit = false;
}

TokenScanner::TokenScanner(string str) {
    ignoreWhitespaceFlag = false;
    singleDigit = false;
    setInput(str);
}
```

Implementation: private attributes

In tokenscanner.h:

```
private:
```

```
/* Instance variables */
```

```
    std::string buffer;          /* The input string containing the tokens */  
    int cp;                     /* The current position in the buffer */  
    bool ignoreWhitespaceFlag; /* Flag set by a call to ignoreWhitespace */  
    bool singleDigit;
```


Implementation: setInput()

In tokenscanner.h:

```
/*
 * Method: setInput
 * Usage: scanner.setInput(str);
 * -----
 * Sets the input for this scanner to the specified string.
 * Any previous input string is discarded.
 */

void setInput(std::string str);
```

In tokenscanner.cpp:

```
void TokenScanner::setInput(string str) {
    buffer = str;
    cp = 0;
}
```

Implementation: options::skipWhitespace

In tokenscanner.h:

```
/*
 * Method: ignoreWhitespace()
 * Usage: scanner.ignoreWhitespace();
 * -----
 * Tells the scanner to ignore whitespace characters.
 * By default, the nextToken method treats whitespace
 * characters (typically spaces and tabs) just like any
 * other punctuation mark and returns them as single-
 * character tokens. Calling
 *     scanner.ignoreWhitespace();
 * changes this behavior so that the scanner ignores
 * whitespace characters.
 */

void ignoreWhitespace();

/* Private methods */

void skipWhitespace();
```

In tokenscanner.cpp:

```
void TokenScanner::ignoreWhitespace() {
    ignoreWhitespaceFlag = true;
}

void TokenScanner::skipWhitespace() {
    while (cp < buffer.length() && isspace(buffer[cp])) {
        cp++;
    }
}
```

Implementation: hasMoreTokens()

In tokenscanner.h:

```
/*
 * Method: hasMoreTokens
 * Usage: if (scanner.hasMoreTokens()) . . .
 * -----
 * Returns true if there are additional tokens
 * for this scanner to read.
 */

bool hasMoreTokens();
```

In tokenscanner.cpp:

```
bool TokenScanner::hasMoreTokens() {
    if (ignoreWhitespaceFlag) skipWhitespace();
    return cp < buffer.length();
}
```

Implementation: nextToken()

In tokenscanner.h:

```
/*
 * Method: nextToken
 * Usage: token = scanner.nextToken();
 * -----
 * Returns the next token from this scanner. If
 * called when no tokens are available, nextToken
 * returns the empty string.
 */

std::string nextToken();
```

In tokenscanner.cpp:

```
string TokenScanner::nextToken() {
    if (ignoreWhitespaceFlag) skipWhitespace();
    if (cp >= buffer.length()) {
        return "";
    } else if (isalnum(buffer[cp])) {
        if ((not isalpha(buffer[cp]) && singleDigit)) {
            cp += 1;
            return buffer.substr(cp-1, 1);
        }
        int start = cp;
        while (cp < buffer.length() && isalnum(buffer[cp])) {
            cp++;
        }
        return buffer.substr(start, cp - start);
    } else {
        return string(1, buffer[cp++]);
    }
}
```

8. How to learn this course well (continued)

- Finish the “Review Questions” for each chapter in textbook
- https://github.com/HouWhalee1222/CSC3002_Review_Question_Answer

Search or jump to... Pull requests Issues Marketplace Explore

HouWhalee1222 / CSC3002_Review_Question_Answer Public

Watch 1 Fork 1 Starred 3

Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 1 tag

Go to file Add file Code

Commit	Message	Time
I-am-Future	Update README.md	14 Feb 2021 on 9 Jun 22 commits
images	finished to ch20	4 months ago
.gitignore	rearranged some representation issues	4 months ago
CSC3002-Unofficial-Answer.md	updated the filename	4 months ago
CSC3002-Unofficial-Answer.pdf	rearranged some repres	4 months ago
README.md	Update README.md	4 months ago

Download pdf here

Register and star this repo!

About

An unofficial answer of review questions to the textbook "Programming Abstractions in C++" for CSC3002 in CUHK-Shenzhen

Readme

3 stars

1 watching

1 fork

Releases 1

pdf format v1.0.0 Latest on 8 Jun

Packages

README.md

CSC3002-Review-Question-Unofficial-Answer

This repository contains an unofficial answer of review questions to the textbook ("Programming Abstractions in

8. How to learn this course well (continued)

- Supplementary materials for this tutorial
- Class basic concepts
- <https://www.learncpp.com/cpp-tutorial/welcome-to-object-oriented-programming/>
- Operator overloading
- <https://www.learncpp.com/cpp-tutorial/introduction-to-operator-overloading/>

9. Q & A time

- Thank you for your listening!
- Lai Wei (USTF)
- (SDS, 120090485@link.cuhk.edu.cn)
- Additional office hour appointment is temporality not available. Ask others instead.
- WeChat Q&A is deprecated in the future.
- Don't ask me about homework problems anymore. Ask TAs instead.
- Office hour: Friday 10:00-11:00 am, Start-up Zone library L103