VE280 2024FA Mid RC Part 3

- VE280 2024FA Mid RC Part 3
    - Tips for Exam
    - L9: Program Arguments
        - Basics
        - Important Key Points
        - Important Clarification:
        - Example
    - L10: IO
        - Buffer
        - iostream
        - fstream
        - sstream
    - L11: Testing
        - Concepts
        - Steps
    - L12: Exception
        - Concepts
        - Try-Catch Block
            - And finally, good luck for your midterm exam!
            - Hope you survive the busy midterm week :)
    - References:

Tips for Exam

- The RC slide is made for you to test your own understanding of the knowledge and help you identify some neglected important concepts. It is not a substitute for the lecture slides!
- Take some time to review the lecture slides. It do pays off!
- Be really careful when you are reading the question requirements. The less careless you are, the more points you win back!
- Do not stick to one difficult question for too long time. Learn to skip and manage your time wisely.

L9: Program Arguments

It's important to actually **understand** the logic behind the mechanism.

Basics

```
int main(int argc, char *argv[]) {...}
```

Important Key Points

- argc (argument counts) : number of arguments (**including** the program name)

```
test@test: diff file1 file2 // argc = 3
```

- argv (argument vectors) : array of arguments treated as **C string** (**including** the program name)

Important Clarification:

- Remember what we learn in 101 about array:

```
char str[] = "Hello";
cout << *str << endl; // H
```

- Here `"Hello"` is both a C string and an array of character, with `str[0] = 'H', str[1] = 'e',...`
- And the array name `str`, is itself a **pointer** pointing to the first element of this array.
- Hence we have "a C-string is itself an array of char and it can be thought of as a pointer to char" (copied from slide L9 P6).
- Next, it is not difficult to see that an array of C strings can be thought of as an array of pointers to char,

  which is just `char *argv[]`.

Example

```
int main(int argc, char *argv[]) {
  cout << argc << endl;
  for (int i = 0; i < argc; i++) {
    cout << argv[i] << endl;
  }
 // Implementation of diff
 ...(omited here)
 }
```

Command in:

```
./mydiff file1 file2
```

Outputs:

```
 3         // argc
 ./mydiff  // argv[0]
 file1     // argv[1]
 file2     // argv[2]
```

Useful function

You don't need to memorize the function. It may be useful for your exercise, project and future study.

```
#include <cstdlib>
int atoi(const char *nptr); // e.g. converts "39" to 39
```

L10: IO

Buffer

- I/O in C++ is buffered: a region of memory that holds data during input or output operations. **The buffer content is cleaned when:**
- A newline (e.g., endl or '\n') is inserted into the stream.

```
 cout << "ok" << endl;
 cout << "ok" << '\n';
```

- The buffer is explicitly flushed.

```
cout << "ok" << flush;
```

- The buffer becomes full.
- The program decides to read from `cin`.
- The program exits.

iostream

- `cin` : standard input (buffered)
- `cout` : standard output (buffered)
- `cerr` : output error messages (not buffered)

fstream

You should go over the concepts and definitions in the slides.
Here we provide some extra examples for you to test your understanding.

- header file: `#include <fstream>`
  Example

```
#include <fstream>
using namespace std;
int main(){
  ifstream ifs;
  ofstream ofs;
  ifs.open("input.txt");
  ofs.open("output.txt");
  char ch;
  while((ch = ifs.get()) != EOF){ // "ifs.get()" returns a single character if success
    ofs << ch;
    // otherwise -1
  }
  while(ifs.get(ch)){ // "ifs.get(ch)" returns true if the reading is successful,
    ofs << ch;
    // otherwise false
  }
  string s;
  while(getline(ifs,s)){ // returns a reference to its first parameter
    ofs << s;
  }
  ofs << ch << s << endl;
  ifs.close();
  ofs.close(); // Don't forget to close
  return 0;
}
```

sstream

- header file: `#include <sstream>`

```
#include <sstream>
using namespace std;
int main(){
  istringstream is;
  ostringstream os;
  string foo;
  int bar;
  string s = "VE 280.";
  is.str(s);
  is >> foo >> bar;
  os << foo << bar;
  s = os.str();
  return 0;
}
```

L11: Testing

Concepts

- Testing: discover a problem
- Debugging: Fix the problem
- incremental testing: test individual pieces of your program (such as functions) as you write them

Steps

Tips: You don't have to memorize these five steps word by word, but understanding the idea behind them is required.

1. Understand the specification
2. Identify the required behaviors

required behaviors: For any specification, boil the specification down to a list of things that must happen.

(See examples in the lecture slides)

3. Write specific tests
   - Simple inputs (simple cases)
   - Boundary conditions (boundary cases)
   - Nonsense (nonsense cases)
4. Know the answers in advance
5. Include stress tests
   - large test cases
   - long running test cases

L12: Exception

Concepts

- Recognize and Handle: partial function with REQUIRES, runtime check
  - modify the inputs/return default outputs.
  - assert(condition) terminates the program if condition is not true.
  - Encode “failure” in the return values.
- Exception: something bad that happens in a block of code, preventing the block from continuing to execute.
- Mechanism: if the exception occurs, the program will move to the handler. (try to understand it!)

Try-Catch Block

- try : throws the exception
- catch : handles the exception

```
try{
    if(foo) throw 2.0;
    if(bar) throw 'a';
    if(list) throw list_make();
}
catch (int n) { }
catch (char c) { }
catch (list_t l) { }
catch (...) { } //default handler
```

- If the exception is successfully handled in the catch block, execution continues normally with the first statement following the catch block.

```
void foo(int i) {
  try { ... }
  catch (int v) {...}
  ... // Do something next
}
```

Rules:

- You **cannot** write a catch block unless you have a try block before it.
- Exception will be propagated along the calling function stack. Only the first catch block with the same type as the thrown exception object will handle the exception.

Little Exercise

Use it to test your understanding:

What is the output of the following code?

```
void foo(int x){
  try {
    bar(x);
  }
  catch(int a){
    cout << "int in foo\n";
  }
  catch(double b){
    cout << "double in foo\n";
  }
  cout << "exit foo\n";
}

void bar(double x){
  throw x;
  try{
    throw x;
  }
  catch(double a){
    cout << "double in bar\n";
  }
  cout << "exit bar\n";
}

int main(){
  int x = 6;
  foo(x);
}
```

Answer:

```
double in foo
exit foo
```

And finally, good luck for your midterm exam!

Hope you survive the busy midterm week :)

References:

[1] Weikang, Qian. VE280 Lecture 9-12.

[2] Wenjing, Zhang. VE280 RC5. 2023FA.

[3] Zihao, He. VE280 RC4. 2024SP.