VE280 MID TERM REIVEW

Topics

- Linux Command
- Compiling C++ program
- C++ basics
- Recursion
- Function Pointer
- Program Argument
- I/O Stream
- Testing/ Debugging
- Exception
- Class

Question Portion

- Simple Question and Answer. 24%+12%
- Read Codes and Answer the question. 18%
- Correct the codes. 12%
- Write codes base on the instruction. 12%+10%+12%

Linux command

- Details in slide 2
- Command options
  - *E.g: ls –a –l; cp –r dir1 dir2*
- Constrains
  - *E.g redir can only remove empty dir*
- **Wildcard: ***
  - *ls *.h *.cpp*

Compiling program in Linux

- Compiling C++: g++ -o program program.cpp
- Run program: ./program <argument>
- Turn on all warnings: -Wall
- Header guard:

E.g //add.h

```
#ifndef ADD_H
#define ADD_H
<your declarations>
#endif
```

Makefile

Target: Dependency
<tab> command

```
all: run_add
run_add: run_add.o add.o
        g++ -o run_add run_add.o add.o
run_add.o: run_add.cpp
        g++ -c run_add.cpp
add.o: add.cpp
        g++ - c run.cpp
clean:
        rm –f run_add *.o
```

Why makefile?

C++ basics

- Function Declaration

  Return_type function_name (*parameters*);

- Function Definition

  Return_type function_name (*parameters*){

  //codes

  }

C++ basics

- **Function Call mechanism**
  - *Call by value*
  - *Call by reference*

```
Void add_one(int x){
        x++;
}
```

```
Void add_one(int& x){
        x++;
}
```

```
Void sum(int a[], int size);
```

C++ basics

- **Pointer**
  - *A pointer contains an address of a variable*
  - *E.g:*

```
int x = 1;
int* pt_x;
pt_x = &x;
*pt_x = 2;
```

What's the value of x?

x = 2

What's the value of pt_x?

The address of x. (maybe 0x000A0001)

C++ basics

- **Reference**
  - *Reference is an alternative name of an object*
  - *Must be initialed by a variable of same type!*
  - ***E.g***

```
int x=1;
int &ref_x = x;
x = 2;
```

What's the value of ref_x?

```
ref_x = 2;
```

What's the difference between reference and pointer?

C++ basics

- **Structure**
  - *An object contains many variables(sometime functions).*
  - *E.g*

```
struct Student{
        string name;
        int age;
};
struct Student ZHK = {“Zhou Hongkuan”, 2};
```

C++ basics

- **Const Quaifier**
  - *Can not be modified and must be initialized when defined*
  - *Usually a global variable.*
  - *E.g :* const int a = 10;

Wrong Example

```
const int x =1;
x = 2;
```

Wrong Example

```
const int x;
```

C++ basics

- **Const reference**
  - *Mostly used in as function arguments*
  - *E.g*

```
const int &ref = 10; //OK
int &ref = 10; //Wrong
```

```
int x = 10;
const int &ref = x;
cout<<ref<<endl;
x++;
cout<<ref<<endl;
```

```
int x = 10;
const int &ref = x+1;
cout<<ref<<endl;
x++;
cout<<ref<<endl;
```

What's the output?

C++ basics

- **Const reference as function argument**
  - *Fast and secure!*
  - *E.g*

```
int avg_age(const struct Student &stu);//OK
int avg_age(struct Student const &stu);//Wrong!
```

Another example:

```
int add_print(int &x){
    x++;
    cout<<x<<endl;
}
```

```
int main{
    const int x =10;
    add_print(x);//Wrong!
}
```

C++ basics

- **Const pointer**
  - *Pointer to const && const pointer*
  - *E.g*

const int *p;	*can't change *p(the value of the object)!*
int *const p;	*can't change p, but can change *p!*
const int *const p;	*can't change both!*

Will become very complex if combined with typedef!
Check slide 04 p40~41!

Procedure abstraction

- Check the slides.
- We suggest you to write comments in the exam.

Recursion

- When a function calls itself, it is a recursion!
- Be familiar with your project 2!
  - *e.g*

```
void path(node * nd){
        if (nd->next != NULL) tree_path(nd->next);
        cout<<nd->value<<“  ”;
}
```

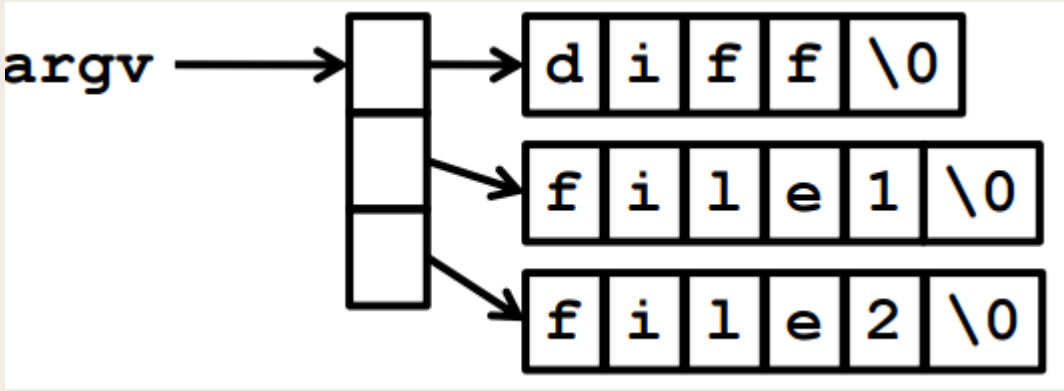
Function pointer

- **Return_type (*name) (argument_type);**
  - *E.g*

```
int (*comp) (int, int);
comp = max;                    //no need for &
int x = comp(3, 5);
```

Passing Arguments to Program

- ./program [argument1] [argument2]…
- int main(int argc, char* argv[])
  - *diff file1 file2*
  - *argc=3*
- Covert c_string to int
  - *atoi(str)*



I/O Streams

```
int a;
double b;
char c;
string d;
cin >> a >> b>> c >> d;
```

```
int a;
double b;
char c;
string d;
cout << a << b<<
c << d <<  endl;
```

File Stream

```
ifstream iFile;
int a;
string b,line;
string name="mytext.txt";
iFile.open(name.c_str());
....
iFile >> a >> b;
while(getline(iFile,line)){
....
}
....
iFile.close();
```

```
ofstream oFile;
int a;
string b;
oFile.open("output.txt");
....
a=1;
b="Ve280";
....
oFile << a << b<<"\n";
....
oFile.close();
```

String stream

```
istringstream iStream;
string a_string =“VE280”;
string level;
int number;
iStream.str(a_string);
iStream >> level >> number;
.....
```

```
ostringstream oStream;
string level = “VE”;
int number = 280;
string result;
oStream << level << number;
result = oStream.str();
.....
```

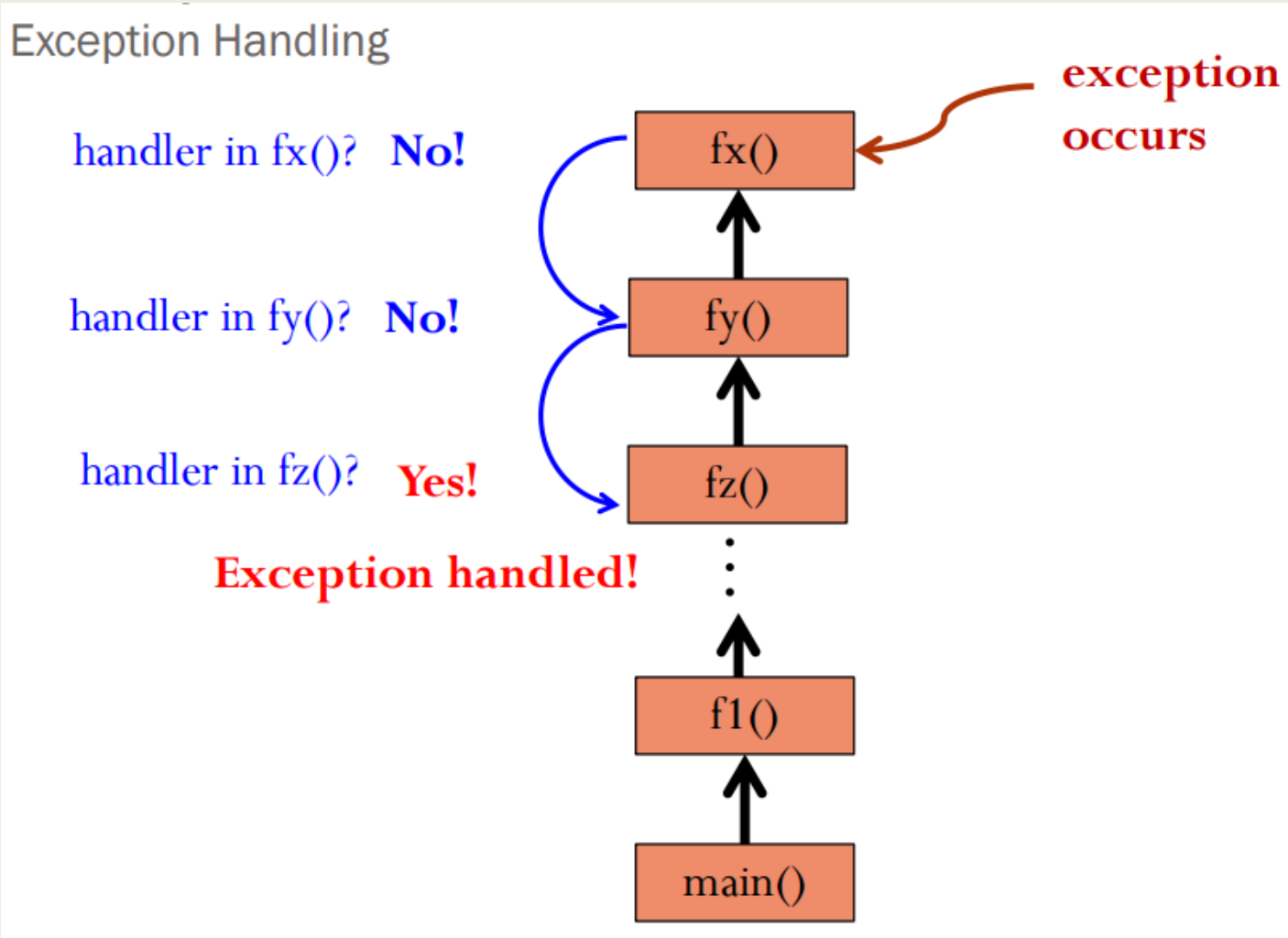
Testing

- Simple inputs
  - *Cases that are "normal" for the problem*
- Boundary conditions
  - *Cases at the edges of what is expected*
- Nonsense
  - *Cases that are clearly unexpected*

Assume user inputs an int

- 3,4,-4,....
- MAX_INT, MIN_INT
- a string....

Exceptions


Exception Handling
exception occurs
handler in fx()? No!
handler in fy()? No!
handler in fz()? Yes!
Exception handled!
fx()
fy()
fz()
f1()
main()

Exceptions

```
void foo() {
    try { Block }
    catch (Type var) { Handler }
}
```

- only the first catch block with the same type as the thrown exception object will handle the exception

Types

- The role of a type:
  - *The set of values that can be represented by items of the type*
  - *The set of operations that can be performed on items of the type.*

- Example
  - *C++ int* *values:*

    operations:

Abstract Data Types

- The basic idea behind a class is to provide **a single entity** that both defines:
  - *The **nature** of an object.*
  - *The **operations** available on that object. These operations are sometimes also called **member functions** or **methods**.*

Example: Intset

```
const int MAXELTS = 100;

class IntSet {

    // OVERVIEW: a mutable set of integers, |set| <= MAXELTS

    int       elts[MAXELTS];

    int       numElts;

    int       indexOf(int v); // return index or MAXELTS

  public:

    IntSet();

    void insert(int v);

      // MODIFIES: this

      // EFFECTS: this = this + {v} if room,

      //          throws int MAXELTS otherwise

    void remove(int v);

      // MODIFIES: this

      // EFFECTS: this = this - {v}

    bool query(int v);  // return whether v in this

    int  size();        // return |this|};
```

representation invariant

Const Member Functions

```
const int MAXELTS = 100;

class IntSet {
    int        elts[MAXELTS];
    int        numElts;
    int indexOf(int v) const;


public:
     void insert(int v);
     void remove(int v);
     bool query(int v) const;
     int  size() const;
};
```

Const Member Functions

- Implement **`size()`**

```
int IntSet::size() const {
  return numElts;
}
```

- A const object can only call its const member functions!

```
const IntSet is;
cout << is.size();
is.insert(2);
```

- If a const member function calls other member functions, they must be const too!

WHAT TO REMEMBER?

- All basic(appear on slides) Linux command.
- Makefile
- C++ basic
- Skill to read and debug code.
- Skill to write program (clearly).
- Go over all the three project you have written.

Class Exercise

- Take out your computer and try linux command with me!

Class Exercise

- Write a header guard for ve280.h

Class Exercise

- Write a header guard for ve280.h

```
#ifndef ve280_H
#define ve280_H
    //...
#endif
```

Class Exercise

- Who is const?

```
#include <iostream>

int *ptr;
const int *ciptr;
int const *icptr;
int * const cptr;
const int * const cicptr;
```

Class Exercise

- Who is WRONG?

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    int p = 5, q = 10;
    const int r = 5, i = 10;
    const int &ref = i;
    int &ref1 = i;
    const int & ref = 42;
    const int & ref1 = r + i;
    int & ref2 = p+q;
    double d = 3.14;
    const int &ref3 = d;
}
```

Class Exercise

- What is the problem?

```
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

int main()
{
    int a = 1;
    unsigned int b = 2;
    if (a - b < 0) cout << "a<b" << endl;
        else cout << "a>b" << endl;
    system("pause");
    return 0;
}
```

Class Exercise

- What is the output?

```
#include <iostream>
#include <climits>
#include <cstdlib>
using namespace std;

int main()
{
    int a[] = { 1,2,3,4,5 } , *p=&a[2];
    cout << *--p << endl;
    system("pause");
    return 0;
}
```

Class Exercise

- Write a recursively program to print the inverse of a string. The input is through std::cin, followed with an enter(‘\n’)
- E.g. input 1234, output 4321.

Answer

```
#include <stdio.h>
#include <cstdlib>

void reverse()
{
    char c;
    if ((c = getchar()) != '\n')
        reverse();
    if (c != '\n')
        putchar(c);
}
void main()
{
    reverse();
    printf("\n");
    system("pause");
}
```

How does it work?

```
#include <stdio.h>
#include <cstdlib>

void reverse()
{
    char c;
    if ((c = getchar()) != '\n')
        reverse();
    if (c != '\n')
        putchar(c);
}
void main()
{
    reverse();
    printf("\n");
    system("pause");
}
```

Wish you all have a good grade!

END!