VE280 2022FA MID RC PART2

L3: Developing Programs

Compilation Process

Compilation process in Linux contains three parts:

- **Preprocessing**: The codes with `#` starts will be implemented.
  - such as: `#define`, `#include`, `#ifdef`
- **Compiler**: Compiles the `.c`/`.cpp` file into object code.
  - The `.c`/`.cpp` files will be compiled to `.o`.
- **Linker**: Links object files into an executable.
  - `.o` files will be linked to an executable file.

Use g++ to Compile Multiple Resources

Three files `class.cpp`, `function.cpp` and `main.cpp` in your directory.

The simplest way to compile them is:\
`g++ -o [name] class.cpp function.cpp main.cpp`,
where `[name]` can be replaced by any name you want.

The complete compile process should be:

`g++ -c class.cpp` $\Rightarrow$ Compile `class.cpp` to `class.o`

`g++ -c function.cpp` $\Rightarrow$ Compile `function.cpp` to `function.o`

`g++ -c main.cpp` $\Rightarrow$ Compile `main.cpp` to `main.o`

`g++ -o [name] class.o function.o main.o` $\Rightarrow$ Link `.o` files to an executable file named `[name]`.

`./[name]` $\Rightarrow$ Run the executable file.
Remark: The **preprocessing** part is implemented automatically by `g++`, so **do not** compile header files.

Header File and Header Guard

Motivation :When we develop a large project where some header files are included for many times in many files.

For example, if `#include "class.h"` in both `main.cpp` and `function.cpp`, the header file `class.h` is included twice.

This may cause multiple definitions of the classes or functions defined in the header file, which will lead to tough problems.

- **Header guard**: used to avoid the above situation.

`class.h`:

```
#ifndef CLASS_H
#define CLASS_H

CODE BODY...

#endif
```

Remark: You **must** write a header guard for any header file.
Mechenism:
For the **first** time when `class.h` is included, the `#ifndef CLASS_H` will return true and the environmental variable `CLASS_H` will be defined. Then the body codes will be implemented until the `#endif`.

For the **second** time when the `class.h` is included, since the variable `CLASS_H` already exists in the environment, `#ifndef CLASS_H` will return false and the body codes will not be implemented twice.

Makefile

Makefile: used to write all the commands during the compile process together in a file.

`Makefile`:

```
main: main.o class.o function.o
    g++ -o main main.o class.o function.o

class.o: class.cpp
    g++ -c class.cpp

main.o: main.cpp
    g++ -c main.cpp

function.o: function.cpp
    g++ -c function.cpp

clean:
    rm -f main *.o
```

How the makefile is constructed:

- Use `:` to link the demand file and dependent files.
- Use a `<tab>` start command to create the demand file from the dependent files.
- Always switch the line between two demands.
- You can add environmental variables in front of a makefile (optional).

How to use a makefile:

- Type `make` to implement the first demand of makefile.
- Type `make [demand name]` to implement a specific demand

L4 Review of C++ Basics

Basic Concepts

- Built-in data types:
  `int`, `double`,`float` ,`char`, `string`.

Question: How many memories does an `int` variable take? How many doese a `char`?

- Input and output by "stream":
  `cout<<"hello world"<<endl`, `cin>>[variables]`
- Operators:
  - Arithmetic: `+`,`-`,`*`,`/`
  - Comparison: `>=`, `==`
  - `x++` or `++x`
  - **Flow**: `>>`,`<<`
- Branch:
  - if/else
  - switch/case
- Loop:
  - while
  - for

lvalue and rvalue

- ***lvalue***: An expression which may appear as **either the left-hand or right-hand side** of an assignment.
- ***rvalue***: An expression which may appear on the **right- but not left-hand** side of an assignment
  - Common lvalues: local variables, return type of "++x", *ptr, ptr[index].
  - Common rvalues: constant, (x+y), return type of "x++" .

Question: What is the result of `x`?

```
x = 3;
++x = x;
//x++ = x;
```

Function declaration and definition

- **Declaration**: should appear before the function is called.

Syntax:

```
Return_Type Function_Name(Parameter_List);
//comment
```

- **Definition**: can appear after the function is called.

Syntax:

```
Return_Type Function_Name(Parameter_List)
{
    //function body
}
//comment
```

Reference

Reference: an important feature of c++.

Comment: Reference is just like the pointer, which means if we change the value of `b`, the value of `a` will also be changed.

Question: Are the following codes correct? What are the values of `a`, `b` and `ref`.

```
int a = 3,b = 1;
int &ref = a;
&ref = b;
```

```
int &ref = 3;
```

```
const int $ref = 3;
```

```
int a = 3,b = 1;
int &ref = a;
ref = b;
```

Pass the value by reference to a function, like:

```
void f(int &a){
    a*=2;
}
```

Benefit: Avoid make a copy of a very big variable, and can change the value of the variable by the function.

Pointers

- Some functions of pointer can be replaced by reference.
- Still very important in the dynamic memory allocation.

Structs

- A set of variables.
- What is the total memory of a stuct variable?
- How to declare and define a struct? How to create a struct variable?
- How to access a struct pointer's member attributes?

L12 Exception

Motivation: Check the runtime failure efficiently and elegantly.

Mechanism: If the exception (failure) occurs, the program will move to the handler.

In c++, we use the "try-catch" block, with **try** block throw the exception and **catch** block handle it.

```
void foo() {
    try { ... throw var;}
    catch (Type var) { }
}
```

For different types of exceptions, you can "throw" **different variable types** so that the program can goto different handlers.

```
try {
    if (foo) throw 2.0;
    // some statements go here
    if (bar) throw 4;
    // more statements go here
    if (baz) throw ‘a’;
}
    catch (int n) { }
    catch (double d) { }
    catch (char c) { }
    catch (...) { }
```

After the exception is thrown, the handler should be the **first** that can handle this failure. After the exception is handled by "catch", the program will implement the codes right after the "catch" block.
Question:

1.

```
ifstream ifs;
string filename = "result.txt";
```

Write the next two statements that first try to connect ifs to the file named "result.txt" and then test if that is successful. If not, throw an exception with the file name.

2. What is the problem in the following block?

```
int factorial(int n) {
int result;
if(n < 0) throw n;
for(result = 1; n>0; n--)
    result *= n;
return result;
}
int main() {
    int y;
    y = factorial(-1);
    catch(int v) {
    cout << "Error: negative input: " << v << endl;
}
return 0;
}
```