CS205 C/C++ Program Design Assignment3

Name：陈逸飞

Student ID：12010502

Develop environment: Apple clang version 12.0.5 (clang-1205.0.22.11)

Part 1- Analysis

In this assignment, we are required to slove three problems.

> Part 1. Split String (30pt in total, 20pt for code, 10pt for Makefile)
>
> Part 2.Product and Bigger (30pt in total, 20pt for code, 10pt for CMakeLists.txt)
>
> Part 3. Pascal's Triangle (20pt)
>
> Part 4. CSV file (20pt)

With the knowledge we learned in the past three weeks, it's easy for us to finish these four tasks. However, these four parts are different with each other, which help us to foucs on making the codying skills we learned more experienced.

1.1 Split String (30pt in total, 20pt for code, 10pt for Makefile)

Topics: struct, usage of struct, usage of struct pointer, call-by-reference, make, shared library

The things we need to do in the first task is to split a set of string in pairs according to the following rules.

> a. If there are not less than 3 characters in the string, put digits in two consecutive even positions (index 0 and index 2) into one {pair};
>
> b. If there are not less than 4 characters in the string, put digits in two consecutive odd positions (index 1 and index 3) into another {pair};
>
> c. Discard all the characters which are used to make pairs;
>
> d. Repeat step.a-c until no more {pair} can be produced.

According to these rules , we can just translate them into C++ code.

In the first part, the functions which are need to be implement were already marked in the document. The struct **Pair** , **spliterPair** and **printpair** in `pair.h` and `pair.cpp`

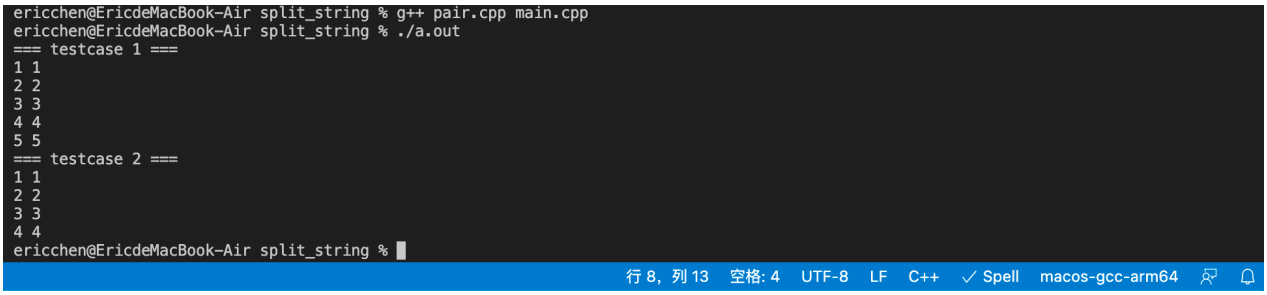
1. **Pair**: This is the struct of the basic element in this task. Since the string includes unknow type of data, we can simply define the **Char** type.
2. **spliterPair**: This is the core function which split the whole string into pairs. accoding to the rules above.
3. **printPair**: This is the function that print out the single pair uing pointer of Pair.

Another 10 points is for the makefile, which not only we need to compile all the **.cpp** files, but also to generate **libpair.so** using shared library. The **.so** should be generate in advance.

At the first, I simply finish the job with simply commend to compile all the file.

Int this part, the explanation in the documents lead me to a deeper look of usage of library, which I will talk about in the last section.

Output



After testing the program, the next step is to write makefile to first generate **lib pair.so** and then compile **main.cpp** to executable file, which make sure using the **lib pair.so** correctly.

Here is the structure of the split-string.

```
ericchen@EricdeMacBook-Air split_string_code % tree
.
├── Makefile
├── lib
│   └── libpair.so
├── main
├── main.cpp
├── pair.cpp
└── pair.h

1 directory, 6 files
```

We set **libpair.so** to be the **TARGET**, and the **pair.cpp** to be the **prerequisite**. The commend includes three steps. Firstly, create a folder named **lib**. Then compile pair.cpp to be libpair.so and put the generated file into lib folder. At last, generate the executable file of **main**.

The original version

```
libpair.so : pair.cpp
  mkdir lib
  g++ -shared -fPIC -o ./lib/libpair.so pair.cpp
  g++ -o main main.cpp ./lib/libpair.so
```

The version with replace statement

```
GCC = g++
TARGET = libpair.so
OBJECT = pair.cpp
LIB_PATH = ./lib/

$(TARGET) : $(OBJECT)
  mkdir lib
  $(GCC) -shared -fPIC -o $(LIB_PATH)$(TARGET) $(OBJECT)
  $(GCC) -o main main.cpp $(LIB_PATH)$(TARGET)
```

After using the **make** commend to generate the required file, input `./main` to run the program.

1.2 Part 2.Product and Bigger (30pt in total, 20pt for code, 10pt for CMakeLists.txt)

Topics: inline function, default arguments, template function, function template specialization,cmake

The Part2 actually include two small takes: Product and Compare.

1.2.1 Product

The function that we need to implement is in **product.h**. We need to use inline functions to get the product of anywhere between 2 and 5 integers or float number.

Since the product process not allow the 0 to be exist, and if there are missing number in 5 parameters, we must set the parameters into default setting.

1.2.2 Bigger

The function bigger we need to implement is in **bigger.h**. The data types that we compare are integer, double, char and string. So using template and specialized template can save a lot space and time. The char type need to be noticed specially, since the tester may input long char array to find the bigger one. So using **const char *** type can turn into string type.

In **const char ***, I use function **strcmp** to compare the result of two char `int ptr = strcmp(a,b)`;. And in **string** type, I use `int factor = a.compare(b);` to compare.

1.2.3 CMakeLists.txt

In this part, we use **cmake** to compile and link **main_prouct.cpp** and **main_bigger.cpp**. to be project **bigger** and **product**.

```
cmake_minimum_required(VERSION 3.23.0)

project(main_bigger)
project(main_product)

add_executable(main_bigger main_bigger.cpp) # source file
add_executable(main_product main_product.cpp) # source file
```

After input the **cmake** and **make** commend, the following file will be generated.

```
ericchen@EricdeMacBook-Air product_and_bigger % cmake
Usage

  cmake [options] <path-to-source>
  cmake [options] <path-to-existing-build>
  cmake [options] -S <path-to-source> -B <path-to-build>

Specify a source directory to (re-)generate a build system for it in the
current working directory.  Specify an existing build directory to
re-generate its build system.

Run 'cmake --help' for more information.

ericchen@EricdeMacBook-Air product_and_bigger % make
-- The C compiler identification is AppleClang 13.1.6.13160021
-- The CXX compiler identification is AppleClang 13.1.6.13160021
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler:
/Library/Developer/CommandLineTools/usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler:
/Library/Developer/CommandLineTools/usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to:
/Users/ericchen/Desktop/cppLab/Assignment3/product_and_bigger
[ 25%] Building CXX object CMakeFiles/main_bigger.dir/main_bigger.cpp.o
[ 50%] Linking CXX executable main_bigger
[ 50%] Built target main_bigger
[ 75%] Building CXX object CMakeFiles/main_product.dir/main_product.cpp.o
[100%] Linking CXX executable main_product
[100%] Built target main_product
```

The current file structure

```
ericchen@EricdeMacBook-Air product_and_bigger % tree
.
├── CMakeCache.txt
├── CMakeFiles
│   ├── 3.23.0
│   │   ├── CMakeCCompiler.cmake
│   │   ├── CMakeCXXCompiler.cmake
│   │   ├── CMakeDetermineCompilerABI_C.bin
│   │   ├── CMakeDetermineCompilerABI_CXX.bin
│   │   ├── CMakeSystem.cmake
│   │   ├── CompilerIdC
│   │   │   ├── CMakeCCompilerId.c
│   │   │   ├── CMakeCCompilerId.o
│   │   │   └── tmp
│   │   └── CompilerIdCXX
│   │       ├── CMakeCXXCompilerId.cpp
│   │       ├── CMakeCXXCompilerId.o
│   │       └── tmp
│   ├── CMakeDirectoryInformation.cmake
│   ├── CMakeError.log
│   ├── CMakeOutput.log
│   ├── CMakeTmp
│   ├── Makefile.cmake
│   ├── Makefile2
│   ├── TargetDirectories.txt
│   ├── cmake.check_cache
│   ├── main_bigger.dir
│   │   ├── DependInfo.cmake
│   │   ├── build.make
│   │   ├── cmake_clean.cmake
│   │   ├── compiler_depend.make
│   │   ├── compiler_depend.ts
│   │   ├── depend.make
│   │   ├── flags.make
│   │   ├── link.txt
│   │   ├── main_bigger.cpp.o
│   │   ├── main_bigger.cpp.o.d
│   │   └── progress.make
│   ├── main_product.dir
│   │   ├── DependInfo.cmake
│   │   ├── build.make
│   │   ├── cmake_clean.cmake
│   │   ├── compiler_depend.make
│   │   ├── compiler_depend.ts
│   │   ├── depend.make
│   │   ├── flags.make
│   │   ├── link.txt
```

```
│   │   ├── main_product.cpp.o
│   │   ├── main_product.cpp.o.d
│   │   └── progress.make
│   └── progress.marks
├── CMakeLists.txt
├── Makefile
├── bigger
├── bigger.h
├── cmake_install.cmake
├── main_bigger
├── main_bigger.cpp
├── main_product
├── main_product.cpp
├── product
└── product.h

9 directories, 51 files
```

Output of main_product

```
ericchen@EricdeMacBook-Air product_and_bigger % ./main_product
=== test product of int ===
2
6
24
120
=== test product of double ===
0.8
2.4
9.6
48
```

Output of main_bigger

```
ericchen@EricdeMacBook-Air product_and_bigger % ./main_bigger
=== test int ===
2
=== test double ===
2.5
=== test const char* ===
It's true, I already knew this before i sign in
=== test string ===
The course we take is very hard
```

1.3 Part 3. Pascal's Triangle (20pt)

Topics: recursion, header files

Pascal's Triangle is also called Yanghui triangle, which we have already experienced in Java class. Finish the task with recursion is easy as long as we follow the rules of creating the Pascal's Triangle:

> Defining every number in it as the sum of the item above it and the item above and to the left of it.

This can be translate into code.

```
return pascal(row - 1, column) + pascal(row - 1, column - 1);
```

Since we are using a two dimentional arrray, another thing that we need to be notice is that the blank space should be set into 0, which promise the recursion formula is correct.

The only function that we need to implement is `int pascal(int row, int column)`. Using recursion to get the result is quite easy with the tips, yet it will loss some efficiency.

```
ericchen@EricdeMacBook-Air pascal_triangle % ./main
1
0
3
6
```

1.4 Part 4. CSV file (20pt)

Topics: file I/O, std::string manipulation

> These values are separated by commas (.csv means “Comma Separated Values”, it’s a commonly used format for exchanging data). When a piece of data is missing (province or state), you get two commas in succession, for instance (first row in the file): **Aarhus,,Denmark,56.150,10.217.**

.CSV file is a specific type of data processing, which can be considered as a **.txt** file. The read and store process is same as we learned in Lab. By using **ifstream** and **ofstream**.

```
ifstream world_data("world_cities.csv");
ofstream china_data;
china_data.open("china_cities.csv");
```

The task required us to extract the cities in China, so we need to search for the third word in the line. `Shenzhen,Guangdong,China,22.550,114.100` . In order to locate the line, we just need to compare the third element of the line with **China**.

I create **run.cpp** and **select_cities.h**, so if you use Coderunner, you can just run run.cpp alone. Or you can use command line `g++ run.cpp -o run` with `./run`.

Run.cpp

```
#include "select_cities.h"
```

```cpp
int main()
{
    string data;
    ifstream world_data("world_cities.csv");
    ofstream china_data;
    china_data.open("china_cities.csv");

    if (!world_data.is_open())
    {
        cout << "Error: fail to open input file" << endl;
        exit(1);
    }

    if (!china_data.is_open())
    {
        cout << "Error: fail to opening output file" << endl;
        exit(1);
    }

    // 读取数据
    while (getline(world_data, data))
    {
        if(judge_city_in_China(data) == 1)
            china_data << data << endl;
    }

    world_data.close();
    china_data.close();
    return 0;
}
```

Select_cities.h

```cpp
#ifndef CITIES_H
#define CITIES_H

#define NUM_CHINA 6

#include <iostream>
#include <istream>
#include <streambuf>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>

using namespace std;
```

```
bool judge_city_in_China(string Sentence)
{
    string search[NUM_CHINA];
    string China = "China";
    istringstream read(Sentence);

    for (int i = 0; i < NUM_CHINA; i++)
    {
        getline(read, search[i], ',');
        if (search[i] == China)
            return 1;
    }
    return 0;
}
#endif
```

China_citites.csv

```
Beijing,,China,39.900,116.400
Changchun,Jilin,China,43.900,125.200
Chengdu,Sichuan,China,30.667,41.000
Chongqing,,China,29.567,106.567
Dalian,Liaoning,China,38.917,121.633
Dongguan,Guangdong,China,23.033,113.717
Gaoxiong,Taiwan,China,22.633,120.267
Guangzhou,Guangdong,China,23.133,113.267
Handan,Hebei,China,36.600,114.483
Hangzhou,Zhejiang,China,30.250,120.167
Harbin,Heilongjiang,China,45.750,126.633
Hong Kong,Hong Kong,China,22.283,114.167
Jinan,Shandong,China,36.667,116.983
Kunming,Yunnan,China,25.067,102.683
Lanzhou,Gansu,China,36.033,103.800
Lhasa,Tibet,China,29.650,91.100
Macau,Macau,China,22.167,113.550
Nanjing,Jiangsu,China,32.050,118.767
Nanning,Guangxi,China,22.817,108.317
Qingdao,Shandong,China,36.067,120.383
Qiqihar,Heilongjiang,China,47.433,123.450
Shanghai,,China,31.200,121.500
Shenyang,Liaoning,China,41.817,123.417
Shenzhen,Guangdong,China,22.550,114.100
Shigatse,Tibet,China,29.267,88.883
Shijiazhuang,Hebei,China,38.050,114.500
Taibei,Taiwan,China,25.033,121.633
Tainan,Taiwan,China,22.983,120.183
Taiyuan,Shanxi,China,37.867,112.567
Taizhong,Taiwan,China,24.150,120.667
Tianjin,,China,39.133,117.183
```

```
Wuhan,Hubei,China,30.583,114.283
Xi'an,Shaanxi,China,34.267,108.900
Xining,Qinghai,China,36.633,101.767
Zhengzhou,Henan,China,34.767,113.650
Ürümqi,Xinjiang,China,43.833,87.600
```

Part 2- Error handling

2.1 forget to update the length——spilt_string

According to the rules, the pairs are determined by the loaction of the element in the string. So we can search the value set by set(a set of value is 4). In the end of every loop, we need to update the set to next four elements. At the first time I define a **string_length** to represent the length of the input string, but I forget to update the string_length, so the string check is actually out of range.

```
ericchen@EricdeMacBook-Air split_string % g++ pair.cpp main.cpp
ericchen@EricdeMacBook-Air split_string % ./a.out
=== testcase 1 ===
libc++abi: terminating with uncaught exception of type std::out_of_range: basic_string
zsh: abort      ./a.out
ericchen@EricdeMacBook-Air split_string %
```

2.2 forget to set value in the blank area——Pascal's Triangle

The Pascal's triangle use the values of two elements above it to compute.

```
return pascal(row - 1, column) + pascal(row - 1, column - 1);
```

So every elements above the target value should be set, or it will cause chaos.

2.3 Produce the whole Pascal's Triangle——Pascal's Triangle

At first, I thought about we can first generate the required Triangle and select the elements in it. However, this menthod loss a lot of efficiency. When I increase the number of the row and column, the time increase dramatically.

```
  for (int i=1;i<n;i++)
  {
    for (int j=1;j<i;j++)           //j<i是 因为a[n][n] 全是1了; j=0那一竖排也全是
1;
      a[i][j]=a[i-1][j-1]+a[i-1][j];
  }
```

2.4 getline——CSV

The input and output file is easy to read and store, but when I read the line, I don't know how to spilt the data with comma. By searching in the web, I learnd how to use `istringstream read(Sentence);` to read in the words with `getline(read, search[i], ',');` . Therefore we can judge whether the sentence have the words **"China"** or not.

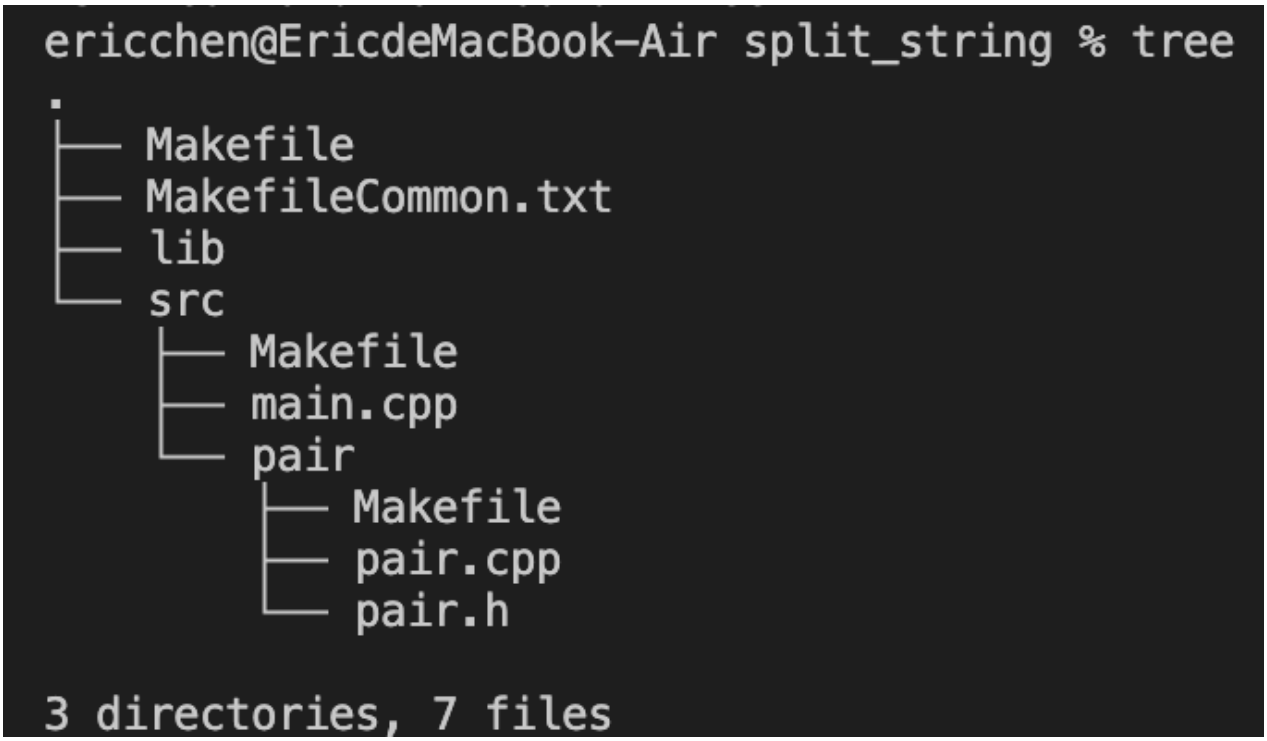
Part 3-Conclusion

3.1 cmake and makefile

> Here, the step of generating libpair.so is unnecessary in fact, but you can try to imagine a scenario: we provide a application to our user (that is main here). And one day, our user wants us to provide some functions of our application for them to develop application by themselves. We just need to modify the makefile, then generate some shared libraries and share these libraries to them. That's the big power of make, it can handle many scenarios, even if complex scenarios.

In order to get a better understanding of cmake usage, I rewite the the first tasks structure and makefile.

The original structure



There are three **makefile** in this project.

The outer **Makefile** compile all the small makefile subdirectories.

If we had a big project with a lot of dynamic libraries, it would be too much trouble to go into the directory and compile it every time. We can write a total Makefile in test_makefile_so and use this Makefile to execute all of the submakefiles so that we can run this one Makefile when the total is recompiled.

The commend to compile

```
ericchen@EricdeMacBook-Air split_string % make clean; make install
rm -rf libpair.so *.o *.so
rm -rf main *.o *.so
compiling pair.o
linking libpair.so
cp libpair.so ../../lib/
compiling main.o
linking main
cp main ../lib/
```

The update structure

```
ericchen@EricdeMacBook-Air split_string % tree
.
├── A1Make1.png
├── A1Make2.png
├── Makefile
├── Maketree.png
├── README
├── lib
│   ├── libpair.so
│   └── main
└── src
    ├── Makefile
    ├── main
    ├── main.cpp
    ├── main.o
    └── pair
        ├── Makefile
        ├── libpair.so
        ├── pair.cpp
        ├── pair.h
        └── pair.o

3 directories, 16 files
```

To executae, we need to first get into **lib** file to find main.exe.

```
ericchen@EricdeMacBook-Air split_string % cd lib
ericchen@EricdeMacBook-Air lib % ./main
=== testcase 1 ===
1 1
2 2
3 3
4 4
=== testcase 2 ===
1 1
2 2
3 3
4 4
```

split_string/Makefile

```
#管理下面的文件夹内的Makefile

SUBDIRS = src/pair  #编译库
SUBDIRS += src

LIB_PATH = ./lib

#定义一个函数，遍历所有的$(SUBDIRS)，$1是参数
define make_subdir
@for i in $(SUBDIRS); do\
(cd $$i && make $1) \
done;
endef


#默认的目标为编译所有子目录
ALL:
  $(shell if ! test -d $(LIB_PATH); then mkdir $(LIB_PATH); fi;)
  $(call make_subdir)

#清空所有的子目录
clean:
  $(shell if test -d $(LIB_PATH); then rm -rf $(LIB_PATH); fi;)
  $(call make_subdir,clean)

# 把子目录里生成的程序或so都复制到$(LIB_PATH)目录下
install:
  $(shell if ! test -d $(LIB_PATH); then mkdir $(LIB_PATH); fi;)
  $(call make_subdir, install)
```

./src/Makefile

```
LINK  = @echo linking $@ && g++
```

```
GCC   = @echo compiling $@ && g++
FLAGS = -g -DDEBUG -W -Wall -fPIC
GCCFLAGS=
DEFINES =
HEADER  = -I./
LIBS  =
LINKFLAGS =

HEADER += -I./pair
#链接的时候从lib目录下找libpair.so
LIBS  += -L../lib -lpair

OBJECT := main.o \

#加入了lib的相对路径，编写完的main会install到lib下面
LIB_PATH = ../lib/

TARGET = main

$(TARGET) : $(OBJECT)
  $(LINK) $(FLAGS) $(LINKFLAGS) -o $@ $^ $(LIBS)

.cpp.o:
  $(GCC) -c $(HEADER) $(FLAGS) $(GCCFLAGS) -fpermissive -o $@ $<

install: $(TARGET)
  cp $(TARGET) $(LIB_PATH)

clean:
  rm -rf $(TARGET) *.o *.so
```

./src/pair/Makfile

```
LINK  = @echo linking $@ && g++
GCC   = @echo compiling $@ && g++
FLAGS = -g -DDEBUG -W -Wall -fPIC
GCCFLAGS=
DEFINES =
HEADER  = -I./
LIBS  =
LINKFLAGS = -shared

#HEADER += -I./pair
#链接的时候从bin目录下找libpair.so

OBJECT := pair.o  #编译pair.cpp

#指出LIB的相对路径
LIB_PATH = ../../lib/
```

```
#生成的文件libpair.so
TARGET = libpair.so

$(TARGET) : $(OBJECT)
  $(LINK) $(FLAGS) $(LINKFLAGS) -o $@ $^ $(LIBS)

.cpp.o:
  $(GCC) -c $(HEADER) $(FLAGS) $(GCCFLAGS) -fpermissive -o $@ $<

install: $(TARGET)
  cp $(TARGET) $(LIB_PATH)

clean:
  rm -rf $(TARGET) *.o *.so
```

> Reference:
>
> https://blog.csdn.net/liujun3512159/article/details/122420870

I hope in the course project I can make good use of makefile.