For-each, Random number
Makefile, README

Prof. Jinkyu Jeong (jinkyu@skku.edu)
TA -- Minwoo Ahn (minwoo.ahn@csl.skku.edu)
TA -- Donghyun Kim (donghyun.kim@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
For-each, Random number
#include <iostream>
using namespace std;

int main() {
    const int N = 3;
    int scores[N] = {90, 95, 100};

    for (auto n : scores)
        cout << n << endl;
}

for-each-element in array:

    for (element declaration : array)
        statement;
Reference Type

```cpp
#include <iostream>
using namespace std;

int main() {
    const int N = 3;
    int scores[N] = {90, 95, 100};

    for (auto &n : scores)
        cout << n << endl;
}

Use reference (pointer) instead of copying element (which may be expensive)
```
Swap (C-style)

```cpp
#include <iostream>
using namespace std;

void swap (int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int main() {
    int x = 3, y = 5;
    swap(&x, &y);
    cout << x << ' ' << y << endl;
}
```

Pass pointers
#include <iostream>
using namespace std;

void swap (int &a, int &b) {
    auto t = a;
    a = b;
    b = t;
}

int main() {
    int x = 3, y = 5;
    swap(x, y);
    cout << x << ' ' << y << endl;
}
Random Numbers

• Pseudo random number generator

```cpp
#include <iostream>
#include <cstdlib>  // c standard lib
using namespace std;

int main() {
    srand(2312);  // set initial seed value
    for (int i = 0; i<10; i++)
        cout << rand() << endl;
}
```

Pseudo random:
Sequence of numbers from rand() is the same, if seed value is the same
Random Numbers (2)

• Set seed from time()

```cpp
#include <iostream>
#include <cstdlib>  // c standard lib
#include <ctime>    // c time lib
using namespace std;

int main() {
    srand((time(NULL)));
    for (int i = 0; i < 10; i++)
        cout << rand() << endl;
}
```

time(NULL) returns the number of seconds since 00:00 hours, Jan 1, 1970 UTC
[Lab – Practice #1]

• int randRange(min, max)
  – Command line: number of random numbers
  – Input: min, max (check the values if min <= max)
  – Output: random numbers in the range min to max inclusive (min <= output <= max)

```
$ ./randRange 5
min? 10
max? 100
13
97
34
21
59
```
Why makefile?

• Simplify compiling source codes
• Describe the relationships among files
• Provide commands for updating each file
• Recompile each changed file

```
g++ -o test main.cpp test.cpp hello.cpp
```

VS

```
make
```
Rule

- A **rule** explains how and when to remake certain files, or to carry out an action.
- A **target** is the name of a file that is generated by a program, or the name of an action to carry out (e.g., clean, install).
- A **prerequisite** is a file that is used as input to create the target.
- A **recipe** is an action that ‘make’ carries out.

```
target ... : prerequisites ...
  recipe
  ...
  ...
```
Example

# Makefile

hello: main.o hello.o
    g++ -W2 -o hello main.o hello.o

main.o: main.cpp
    g++ -W2 -c main.cpp

hello.o: hello.cpp
    g++ -W2 -c hello.cpp

clean:
    rm *.o hello

Target ‘hello’ depends on ‘main.o’ and ‘hello.o’

Target ‘clean’ is not a file, but it is the name of an action

$ make
$ ls
$ hello hello.cpp hello.o main.cpp main.o
How *make* processes a makefile

1. **Start with the first target (default goal)**
2. **Check all prerequisites?**
   - Yes → **Check the rule about this prerequisite**
   - No → **Recompile prerequisite?**
     - Yes → do recipe
     - No → **do recipe**
3. **end**
Example

• If ‘main.cpp’ is modified and enter make command

```
# Makefile
①Find default goal
hello: main.o hello.o
   g++ -W2 -o hello main.o hello.o ⑤Do recipe
main.o: main.cpp ②Check
   g++ -W2 -c main.cpp ③Do recipe
hello.o: hello.cpp ④Check
   g++ -W2 -c hello.cpp
clean:
   rm *.o hello
```
Variable

• Be defined once and substituted in multiple places
• Substitute the variable’s value by writing $(variable)$

```
# Makefile
TARGET=hello
CXX=g++
CXXFLAGS=-W2
OBJECTS=main.o hello.o

$(TARGET): $(OBJECTS)
  $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJECTS)
main.o: main.cpp
  $(CXX) $(CXXFLAGS) -c main.cpp
hello.o: hello.cpp
  $(CXX) $(CXXFLAGS) -c hello.cpp
clean:
  rm $(OBJECTS) $(TARGET)
```
Automatic variables

- $@$: the file name of the target of the rule
- $^$: the names of all the prerequisite
- $?$: the names of all the prerequisites that are newer than the target

$(TARGET): $(OBJECTS)

$(CXX) $(CXXFLAGS) -o $@ $^ 
main.o: main.cpp  
  $(CXX) $(CXXFLAGS) -c $^ 
hello.o: hello.cpp  
  $(CXX) $(CXXFLAGS) -c $^
Special built-in targets

• .PHONY
  – This target is not really the name of a file
  – Two reasons to use a phony target
    • avoid conflict with a file of a same name
    • improve performance

• Others
  – .SUFFIXES, .DEFAULT, .POSIX, etc.

```
.PHONY: clean
clean:
  rm *.o hello
```
README
README

• Contains information about other files in a directory or archive of computer software

• Form of documentation
  – README, README.ME, README.txt, README.md
  – GitHub support markdown(.md)
README contents

• Project title
  • One paragraph of project description goes here
  – Getting started
    • What things you need to install and how to install
    • How to compile
  – Running
    • Explain how to run
  – Versioning
  – Authors
# Markdown

## Sub-heading

Paragraphs are separated by a blank line.

Two spaces at the end of a line produces a line break.

Text attributes *italic*, **bold**, `monospace`.

Horizontal rule:

---

Bullet list:

- apples
- oranges
- pears

Numbered list:

1. wash
2. rinse
3. repeat
Markdown syntax

• Headers

```
# Header1
## Header2
### Header3
#### Header4
##### Header5
###### Header6
####### Header7
```

• Blockquote

```
> First blockquote
> > Second blockquote
```

First blockquote

Second blockquote
Markdown syntax

• Lists

1. First
2. Second
3. Third

1. First
2. Second
3. Third

*Red
*Green
*Blue

+Red
+Green
+Blue

-Red
-Green
-Blue

• Link

[Redis](https://redis.io) - automatic!

https://redis.io

Redis
Markdown syntax

• Horizontal line

* * *
***
*****
---
---------------------------

• Bold, italic, cancellation, monospace

**bold**

***italic***

~~cancellation~~

`monospace`

bold

italic

cancellation

monospace
Markdown syntax

• Code

```
```
```
#include <iostream>
int main(void) {
    cout << "Hello" << endl;
}
```
[Lab – Practice #2]

• Write README.md and upload it on your remote repository

What is Redis?

Redis is often referred as a **data structures** server.

Building Redis

• It is simple. just type **make**
  
  ```
  % make
  ```

• For 32bit machine
  - You can run a 64bit 32bit Redis binary using:
    
    ```
    % make 32bit
    ```

Link

Redis [https://redis.io](https://redis.io)
Reference

• GNU make manual

• GitHub Guides – Mastering Markdown
  – https://guides.github.com/features/mastering-markdown/