Intro - Programming Cycle

Week1

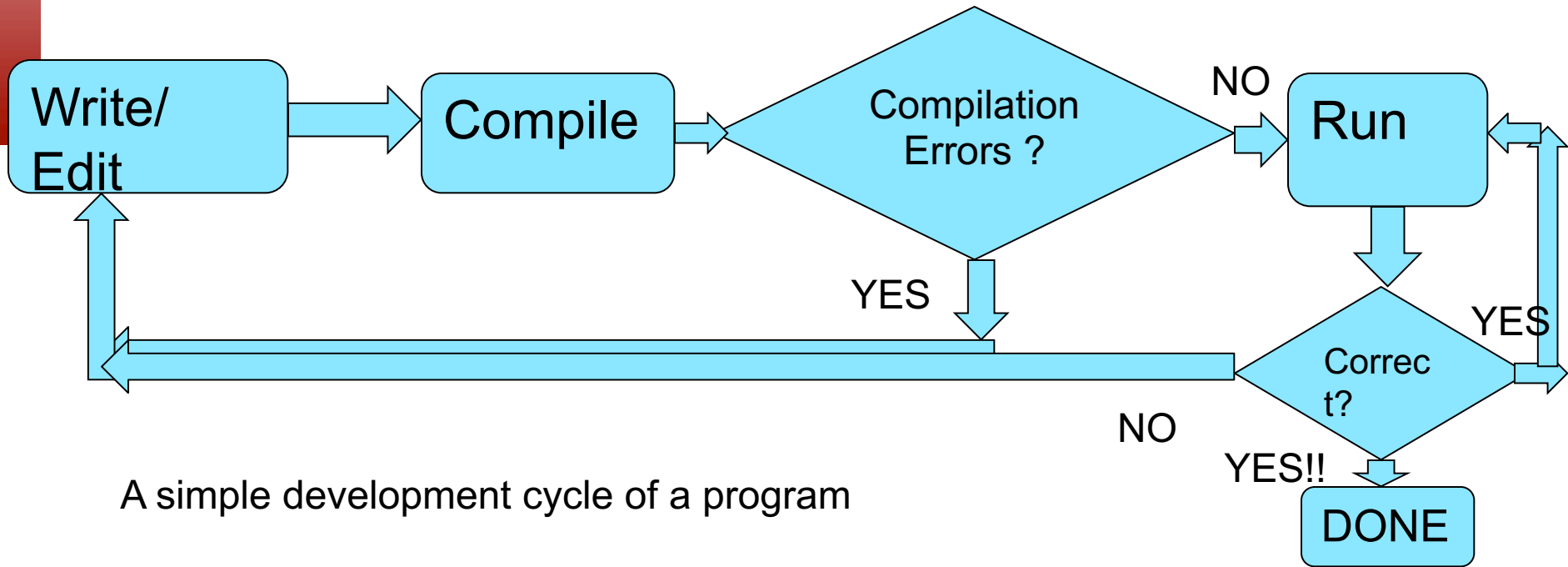
The Programming Cycle

The Programming Cycle

1. Write your program or edit (i.e., change or modify) your program.
2. Compile your program. If compilation fails, return to editing step.
3. Run your program. If output is not correct, return to editing step.

The Programming Cycle

1. Write your program or edit (i.e., change or modify) your program.
2. Compile your program. If compilation fails, return to editing step.
3. Run your program. If output is not correct, return to editing step.



A simple development cycle of a program

Editing

- Open an editor. An editor is a system program that lets you type in text, modify and update it.

- Create your program. Type in your program in an editor. For example use the program gedit or Notepad++. Save what you type into a file called sample.c.

Step 2: Compile

Step 2: Compile

- After editing, you have to COMPILE the program.

Step 2: Compile

- After editing, you have to COMPILE the program.
- The computer cannot execute a C program or the individual statements of a C program directly.

Step 2: Compile

- After editing, you have to COMPILE the program.
- The computer cannot execute a C program or the individual statements of a C program directly.
- For example, in C you can write

```
g = a %b
```

Step 2: Compile

- After editing, you have to COMPILE the program.
- The computer cannot execute a C program or the individual statements of a C program directly.
- For example, in C you can write

```
g = a %b
```

- The microprocessor cannot execute this statement. It translates it into an equivalent piece of code consisting of even more basic statements. For example

Step 2: Compile

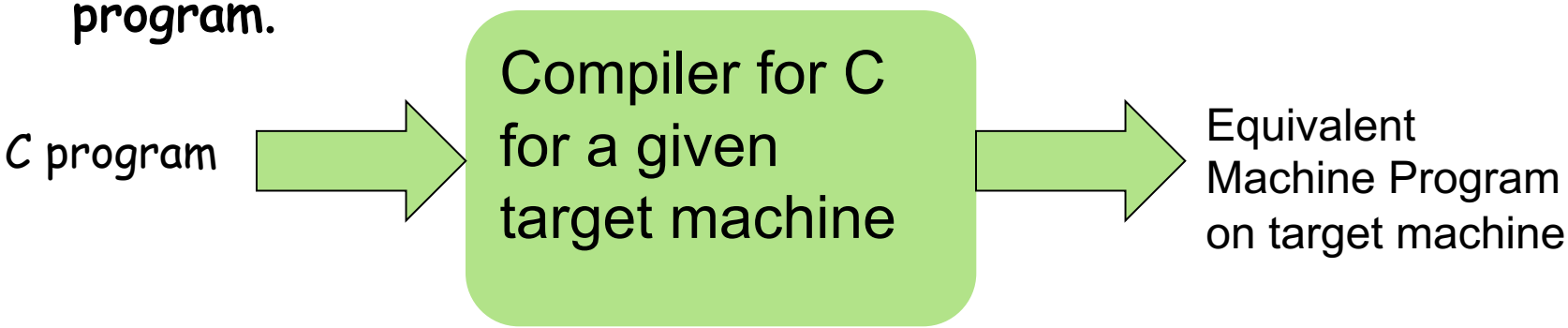
- After editing, you have to COMPILE the program.
- The computer cannot execute a C program or the individual statements of a C program directly.
- For example, in C you can write

```
g = a %b
```

- The microprocessor cannot execute this statement. It translates it into an equivalent piece of code consisting of even more basic statements. For example
  - Load from memory location 0xF04 into register R1
  - Load from memory location 0xF08 into register R2
  - Integer divide contents of R1 by contents of R2 and keep remainder in register R3
  - Store contents of R3 into memory location 0xF12.

Why program in high level languages like C

- Writing programs in machine language is long, tedious and error-prone.
- They are also not portable—meaning program written for one machine may not work on another machine.
- Compilers work as a bridge.
- Take as input a C program and produce an equivalent machine program.



How do you compile?

- On Unix/Linux systems you can **COMPILE** the program using the gcc command.

```
% gcc sample.c
%
```

- If there are no errors, then the system silently shows the prompt (%).
- If there are errors, the system will list the errors and line numbers. Then you can edit (change) your file, fix the errors and recompile.
- As long as there are compilation errors, the **EXECUTABLE** file is not created.

Compilation

- We will use the compiler *gcc* . The command is

```
% gcc  yourfilename.c
```

- gcc stands for Gnu C compiler.
- If there are no errors then gcc places the machine program in an executable format for your machine and calls it a.out.
- The file a.out is placed in your current working directory. More on directories in a little bit!

Simple! Program

sample.c: The program prints the message “Welcome to C”

Simple! Program

- **We will see some of the simplest C programs.**

sample.c: The program prints the message “Welcome to C”

Simple! Program

- We will see some of the simplest C programs.
- Open an editor and type in the following lines. Save the program as sample.c

sample.c: The program prints the message “Welcome to C”

Simple! Program

- We will see some of the simplest C programs.
- Open an editor and type in the following lines. Save the program as sample.c

```
# include <stdio.h>
main ()  {
     printf("Welcome to C");
}
```

sample.c: The program prints the message "Welcome to C"

Compile and Run

Compile and Run

- Now compile the program. System compiles without errors.

```
% gcc sample.c
%
```

- Compilation creates the executable file a.out by default.

Compile and Run

- Now compile the program. System compiles without errors.

```
% gcc sample.c
%
```

- Compilation creates the executable file a.out by default.
- Now run the program. The screen looks like this:

Compile and Run

- Now compile the program. System compiles without errors.

```
% gcc sample.c
%
```

- Compilation creates the executable file a.out by default.
- Now run the program. The screen looks like this:

```
% ./a.out
Welcome to C%
```

Program statements

Program statements

```
# include <stdio.h>

main ()
{
    printf(“Welcome to C”);
}
```

Program statements

```
# include <stdio.h>

main ()
{
    printf("Welcome to C");
}
```

1. This tells the C compiler to include the standard input output library.

2. Include this line routinely as the first line of your C file.

Program statements

```
# include <stdio.h>

main ()
{
    printf("Welcome to C");
}
```

1. This tells the C compiler to include the standard input output library.

2. Include this line routinely as the first line of your C file.

main() is a function. All C programs start by executing from the first statement of the main function.

Program statements

```
# include <stdio.h>

main ()
{
    printf("Welcome to C");
}
```

1. This tells the C compiler to include the standard input output library.

2. Include this line routinely as the first line of your C file.

main() is a function. All C programs start by executing from the first statement of the main function.

printf is the function called to output from a C program. To print a string, enclose it in " " and it gets printed.

Program statements

```
# include <stdio.h>

main ()
{
    printf("Welcome to C");
}
```

1. This tells the C compiler to include the standard input output library.

2. Include this line routinely as the first line of your C file.

main() is a function. All C programs start by executing from the first statement of the main function.

printf is the function called to output from a C program. To print a string, enclose it in " " and it gets printed.

```
printf("Welcome to C");
```

Program statements

```
# include <stdio.h>

main ()
{
    printf("Welcome to C");
}
```

1. This tells the C compiler to include the standard input output library.

2. Include this line routinely as the first line of your C file.

main() is a function. All C programs start by executing from the first statement of the main function.

printf is the function called to output from a C program. To print a string, enclose it in " " and it gets printed.

`printf("Welcome to C");` is a statement in C. Statements in C end in semicolon ;

Errors

Errors

- Let us systematically enumerate a few common errors.

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.
2. Forgetting main function

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.
2. Forgetting main function
3. Forgetting semicolon

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.
2. Forgetting main function
3. Forgetting semicolon
4. Forgetting open or close brace ({ or }).

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.
2. Forgetting main function
3. Forgetting semicolon
4. Forgetting open or close brace ({ or }).
5. Forgetting to close the double quote.

Errors

- Let us systematically enumerate a few common errors.

1. Forgetting to include stdio.h.
2. Forgetting main function
3. Forgetting semicolon
4. Forgetting open or close brace ({ or }).
5. Forgetting to close the double quote.

Try deliberately making these mistakes in your code. Save them and try to compile. Study the error messages for each.

Familiarity with error messages will help you find coding errors later.

Acknowledgments: This lecture slide is based on the material prepared by Prof. Sumit Ganguly, CSE, IIT Kanpur. The slide design is based on a template by Prof. Krithika Venkataramani.