WELCOME

A/Prof Frank Guan, PhD

INF1002 – Programming Fundamentals

Week 8

ABOUT ME

- Dr Frank Guan, PhD

  Associate Professor, SIT

- Research interests:
  - Artificial Intelligence (AI)
  - Virtual Reality (VR)
  - Augmented Reality (AR)
  - Entrepreneurship & innovation
- If any module related matter, please email me at:

  Frank.Guan@singaporetech.edu.sg

INTRODUCTION TO C

Dr Frank Guan

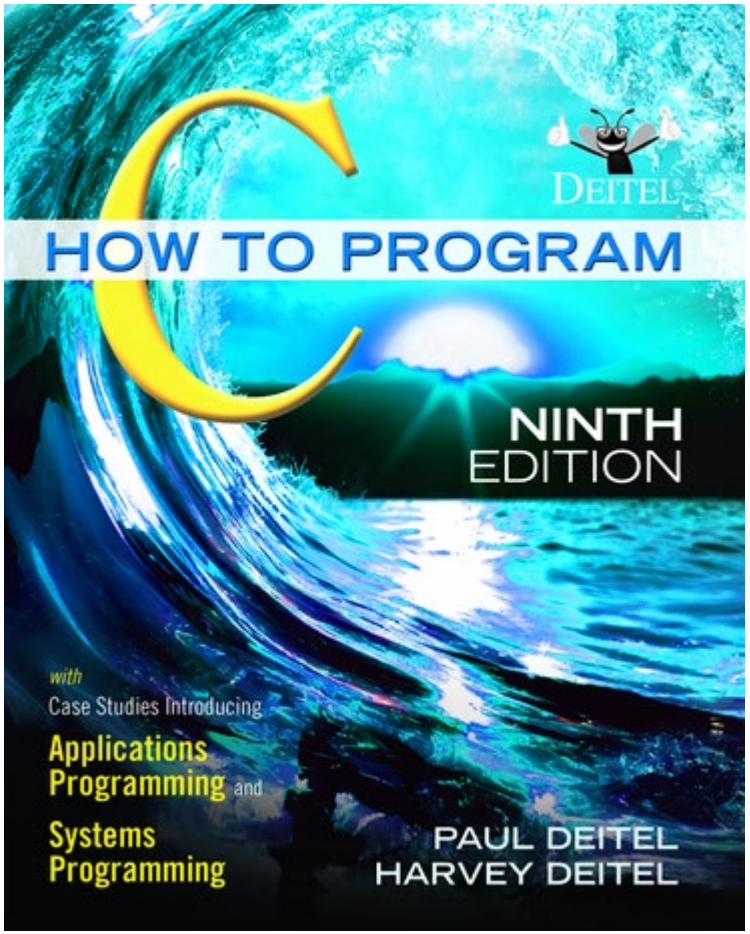
INF1002 – Programming Fundamentals

Week 8

Agenda

1. History of C
2. From Python to C
3. A simple C program
4. Basic data types
5. C formatted I/O
6. Control structures

RECOMMENDED READING



Paul Deitel and Harvey Deitel

C: How to Program

9th Edition

Pearson, 2021

HISTORY OF C

SINGAPORE INSTITUTE OF TECHNOLOGY

QUESTION

- Type in the Zoom Chat window of the result of adding 37 to 45

Natural language communication between human beings

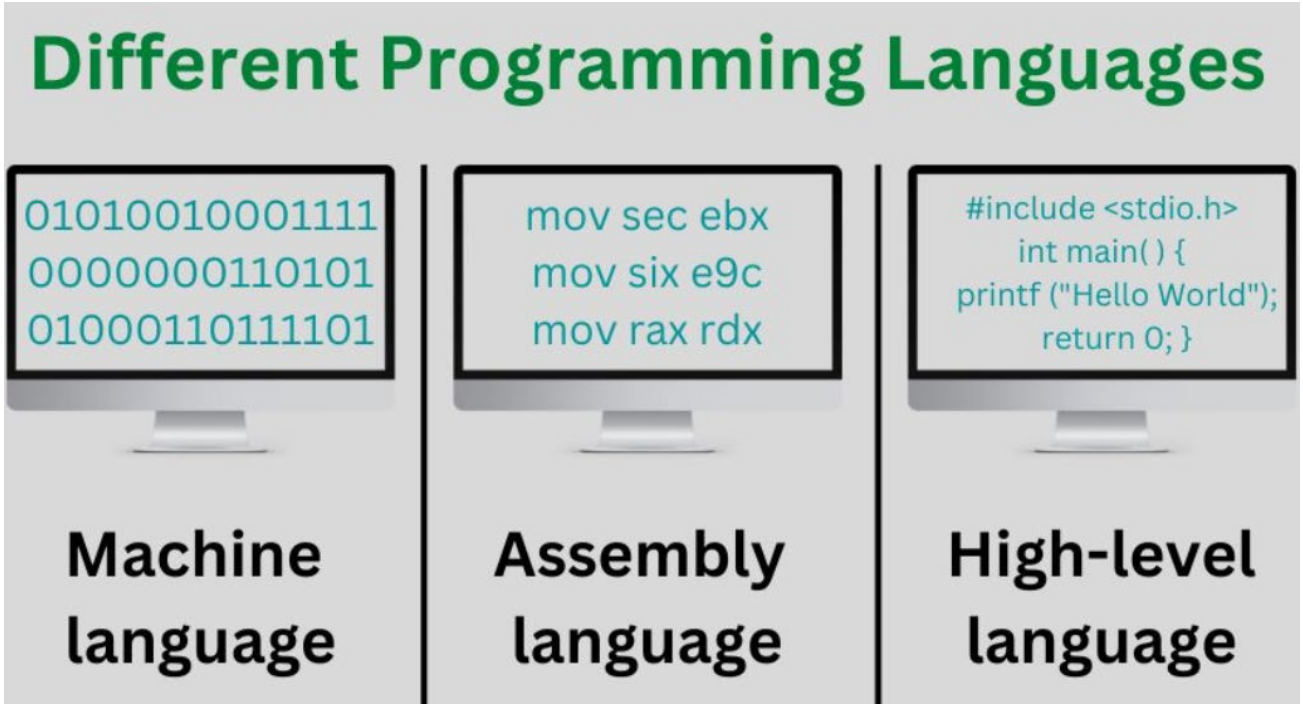
"What is the result of adding 37 to 45?"

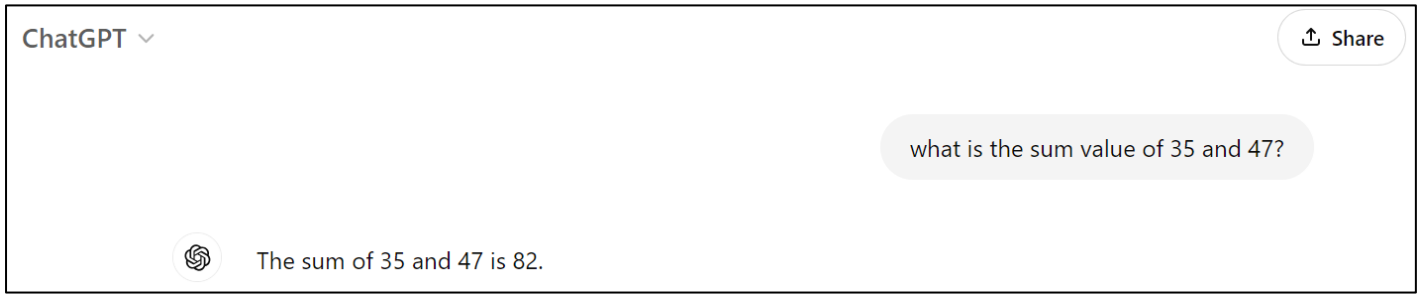
"82"

HOW ABOUT ASKING A COMPUTER WITHOUT USING AI

- Define two integer variables
- Assign values (35 and 47) for each variable
- Calculate the sum value of 35 and 47
- Print out the result (82) on the screen

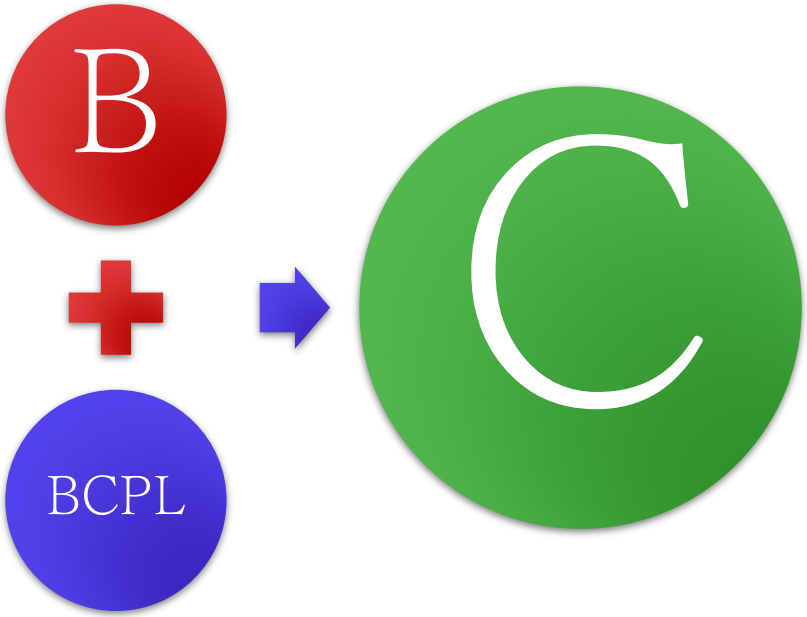
WHY PROGRAMMING LANGUAGE





https://usemynotes.com/assembly-language-and-machine-language/

HISTORY OF C



J. Presper Eckert and John Mauchly

Electronic Numerical Integrator and Computer

1943 - 1946

Martin Richard created the

Basic Combined Programming Language

in 1966 and 1967.

Ken Thompson developed

B – based on BCPL

in 1969 at Bell Lab.

An early version of UNIX was written in B.

HISTORY OF C

C was created by Dennis Ritchie at Bell Lab between 1972 - 1973.

Dennis Ritchie (left) and Ken Thompson were the lead developers of UNIX.

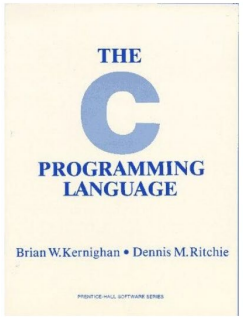
UNIX was re-written in C in 1973.

1970s Traditional C

1989 – C89/ANSI C/Standard C

1990 – ANSI/ISO C

1999 – C99: an attempt to standardise many variations of C



The C Programming Language Paperback - 1978
by Brian W. Kernighan

C VS. RELATED LANGUAGES

More recent derivatives: C++, Objective C, C#

Influenced: Java, Perl, Python (quite different)

C lacks:
- Exceptions
- Range-checking
- Garbage collection
- Object-oriented programming

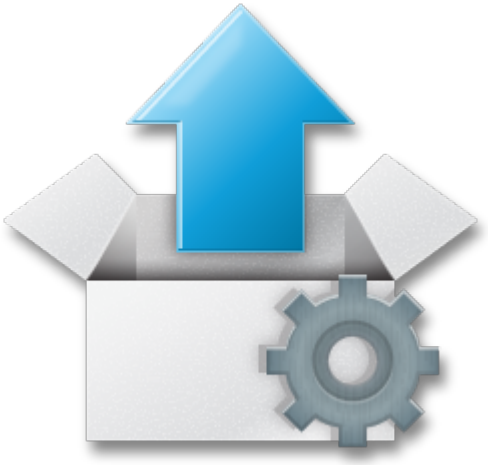
Low-level language - faster code (usually)

FROM PYTHON TO C

SINGAPORE INSTITUTE OF TECHNOLOGY

1. COMPILERS VS. INTERPRETERS

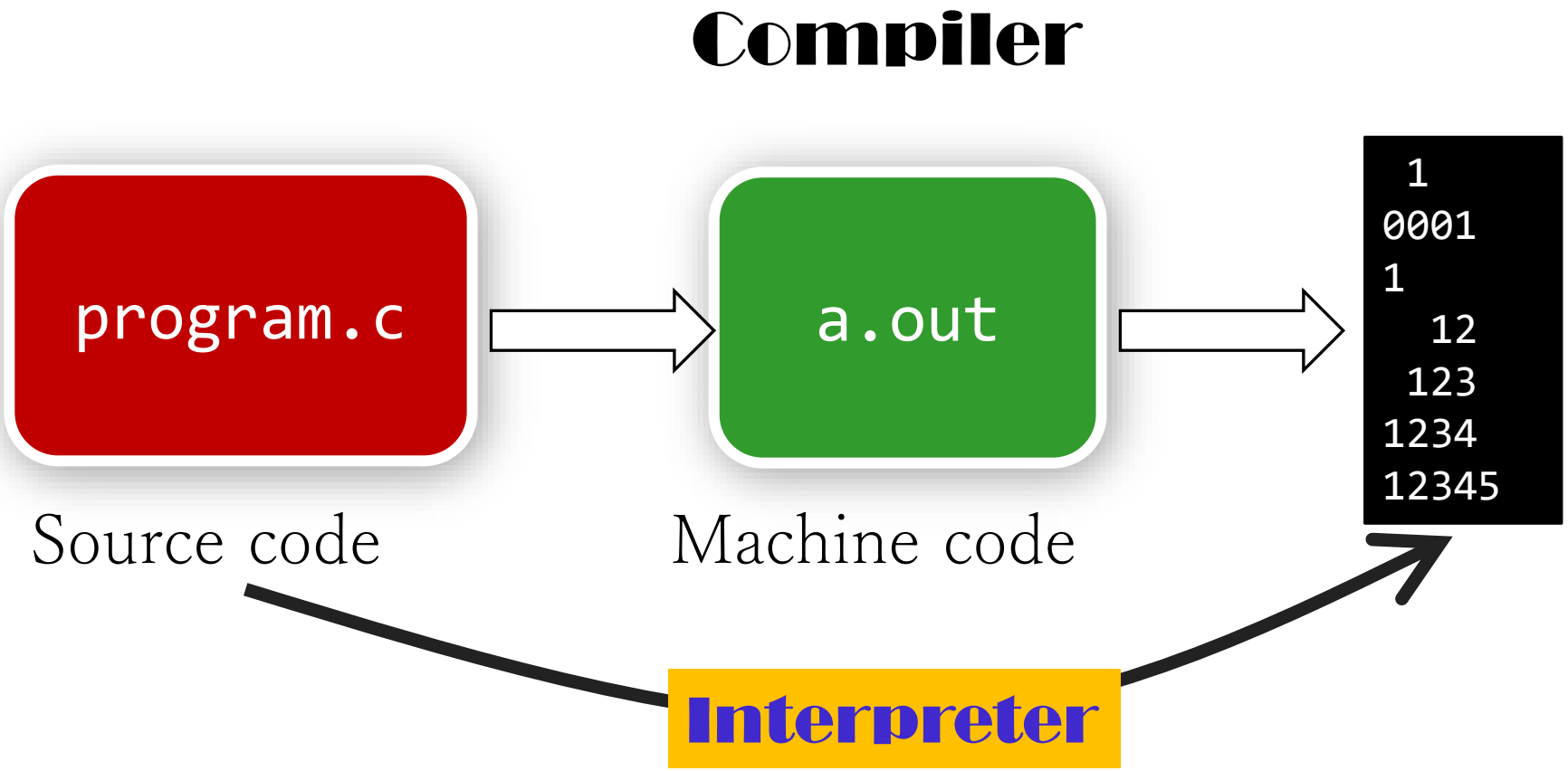
Python uses an **interpreter** to execute a program.



In C, the **compiler** converts a program into machine code.

The machine code can be executed directly by the CPU.

1. COMPILERS VS. INTERPRETERS



1. COMPILERS VS. INTERPRETERS

C programs can lead to **faster runtimes**.

C is designed so the compiler can tell everything it needs to know to translate the program into machine code.

2. VARIABLE DECLARATIONS

C requires variable declarations

These tell the compiler about the variable before it is used.

Once declared, you cannot change its type.

if you happen to misspell a variable's name

2. VARIABLE DECLARATIONS

In Python: no declaration required

You are responsible for checking your own code!

More convenient.

3. WHITESPACE

In Python, whitespace characters are important.

- Identify new statements
- Identify blocks

C **does not** use whitespace except for separating words.

- Use a semi-colon to terminate statements
- Use braces {} to delimit blocks

4. FUNCTIONS

- All C code must be put within functions.

- The `main()` function is the "starting point" for a C program.

4. FUNCTIONS

```c
#include <stdio.h>

int gcd(int a, int b) {

    if (b == 0)
        return a;
    else
        return gcd(b, a % b);

}

int main() {

    printf("GCD: %d\n", gcd(24, 40));

    return 0;

}
```

C Program

```python
def gcd(a, b):
  if b == 0:
    return a
  else:
    return gcd(b, a % b)

print("GCD: " + str(gcd(24, 40)))
```

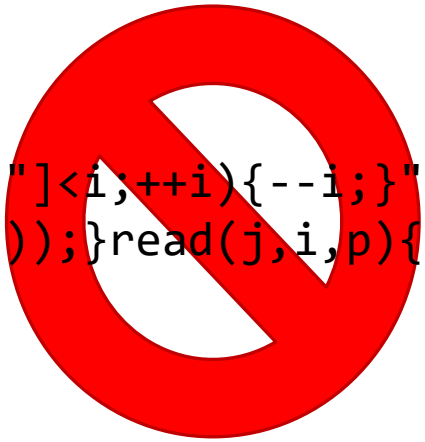
Python Program

From http://www.cs.toronto.edu/~patitsas/cs190/c_for_python.html

GOOD CODING PRACTICE IN C

- Indent blocks of code as in Python
- Format braces and whitespace consistently
- Use comments to explain your code to other programmers
- Use meaningful variable names

```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\
o, world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```



(from http://www.ioccc.org)

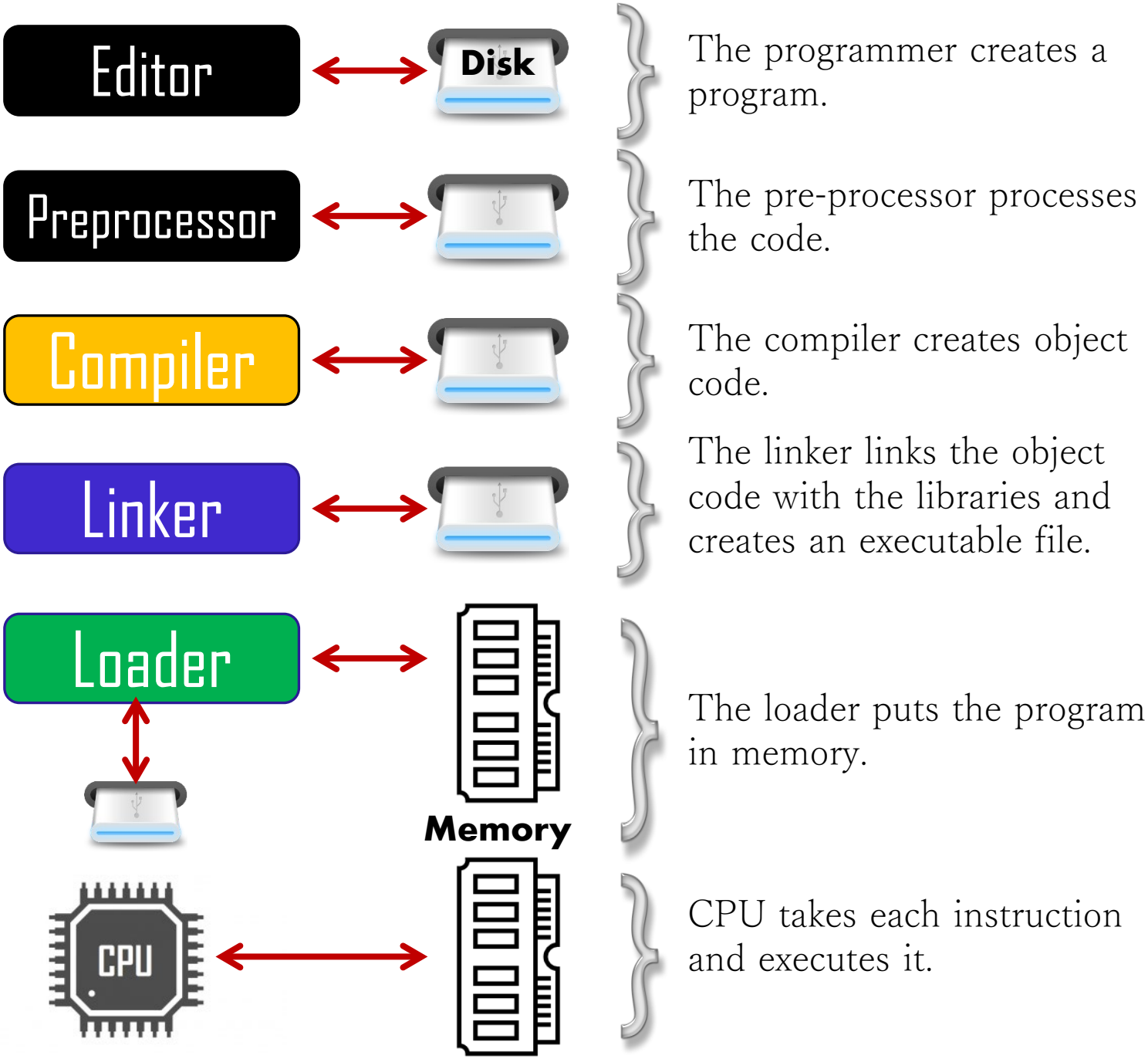
GOOD CODING PRACTICE IN C

- Pay attention to compiler warnings
  - these indicate code that is synactically correct but is likely to lead to run-time errors
- Avoid system-specific features
  - all programs in this module should compile and run using any modern compiler
  - code that works on many systems without modification is called `portable`

A SIMPLE C PROGRAM

SINGAPORE INSTITUTE OF TECHNOLOGY

TYPICAL C PROGRAM DEVELOPMENT ENVIRONMENT



The programmer creates a program.

The pre-processor processes the code.

The compiler creates object code.

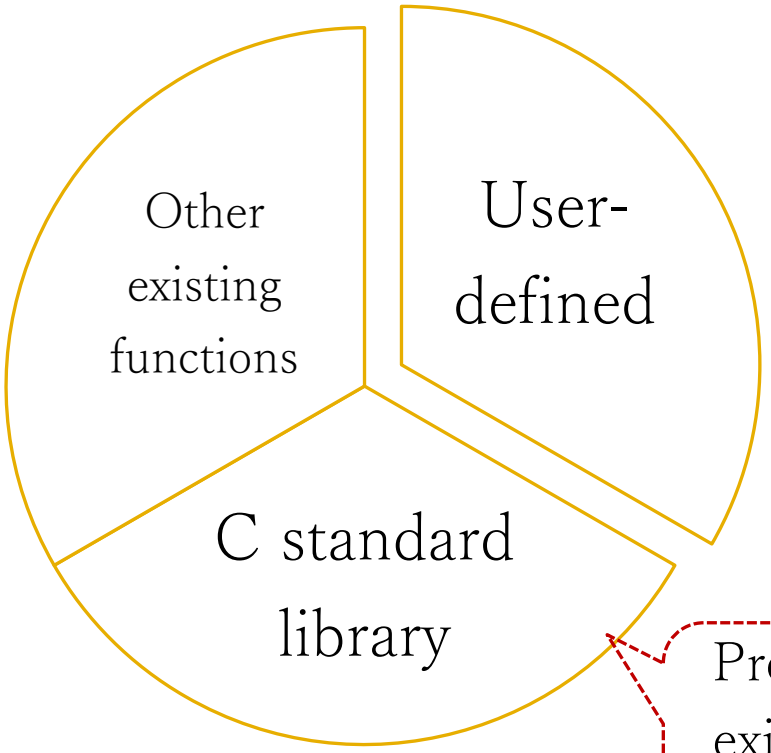
The linker links the object code with the libraries and creates an executable file.

The loader puts the program in memory.

CPU takes each instruction and executes it.

C PROGRAMS

C programs consist of modules called **functions**



A "C" Program

Provides a rich collection of existing **common functions**
Promote software **reusability**

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

Comments: any text between /* and */ is ignored by the compiler.

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

C pre-processor: this line tells the preprocessor to include the content of the standard input/output header **<stdio.h>**

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

C programs contain one or more functions, one of which MUST be `main`. Every program in C begins execution in `main`.

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

One `printf` can print several lines by using additional newline characters `'\n'`

A SIMPLE C PROGRAM

```
/*
 * Hello World program in C.
 */
#include <stdio.h>

int main() {

        printf("Hello world!\n");

        return 0;

}
```

Code: “hello.c”

Returning 0 tells the operating system that the program ended with no errors.

C PRE-PROCESSOR

The **C** pre-processor executes before a program is compiled.

Some actions it performs are:

- definition of symbolic constants
- the inclusion of other files in the file being compiled

Pre-processor directives begin with **#**

#DEFINE PRE-PROCESSOR DIRECTIVE

The #define directive creates symbolic constants.

All subsequent occurrences of NUM_STUDENTS will be replaced with 140.

```
#define NUM_STUDENTS 140

int main() {

    int scores[NUM_STUDENTS];

    for (int i = 0; i < NUM_STUDENTS; i++) {
        scores[i] = 0;
    }

    return 0;

}
```

#INCLUDE PRE-PROCESSOR DIRECTIVE

The **#include** directive causes a copy of a specified file to be included in place of the directive.

#INCLUDE PRE-PROCESSOR DIRECTIVE

The two forms of the **#include** directive are:

<>: is normally used for **standard library headers**, the search is performed normally through pre-designated compiler and system directories.

```
#include <filename>
#include "filename"
```

"": used for **user-defined** files. The pre-processor starts searches in the same directory as the file being compiled

HEADER FILES

A C program might have header files containing:

Function **prototypes** (declaration with no implementation)

Definitions of **constants**

Definitions of various **data types**

Constants

chat1002.h

```
#define MAX_INTENT 32
#define MAX_ENTITY 64

void chatbot_do_load(int inc, char **inv);
void chatbot_do_smalltalk(int inc, char **inv);
```

Function prototypes

```
#include <stdio.h>
#include “chat1002.h"

int main() {
    int done = 0;

    do {
        ...
    } while (!done);

    return 0;
}
```

main.c

```
#include "chat1002.h"

void chatbot_do_load(int inc, char **inv) {
    ...
}

void chatbot_do_smalltalk(int inc, char **inv)
{
    ...
}
```

chatbot.c

ANOTHER SIMPLE C PROGRAM

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

The names **integer1**, **integer2** and **sum** are the names of **variables**.

Code: “sum.c”

A variable is a location in memory where a value can be stored for use by a program.

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

Code: “sum.c”

These definitions specify that the variables integer1, integer2 and sum are of type int.

Code: “sum.c”

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

ANOTHER SIMPLE C PROGRAM

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

The **scanf()** function reads from the standard input, which is usually the **keyboard**.

ANOTHER SIMPLE C PROGRAM

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

This **scanf** has three arguments, **"%d%d"** and **&integer1** and **&integer2**.

ANOTHER SIMPLE C PROGRAM

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;
}
```

scanf(): The first argument, the **format control string**, indicates the **type of data** that should be input by the user. The **%d conversion specifier** indicates that the data should be a (decimal) integer.

ANOTHER SIMPLE C PROGRAM

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("The sum of the numbers = %d\n", sum);
```

scanf(): The second and third arguments of **scanf** begin with an ampersand (**&**)—called the address operator in C—followed by the variable name.

ANOTHER SIMPLE C PROGRAM

Code: "sum.c"

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);
```

`scanf()`: The ampersand tells **`scanf`** the locations (or addresses) in memory at which the variables **`integer1`** and **`integer2`** are stored.

BASIC DATA TYPES

SINGAPORE INSTITUTE OF TECHNOLOGY

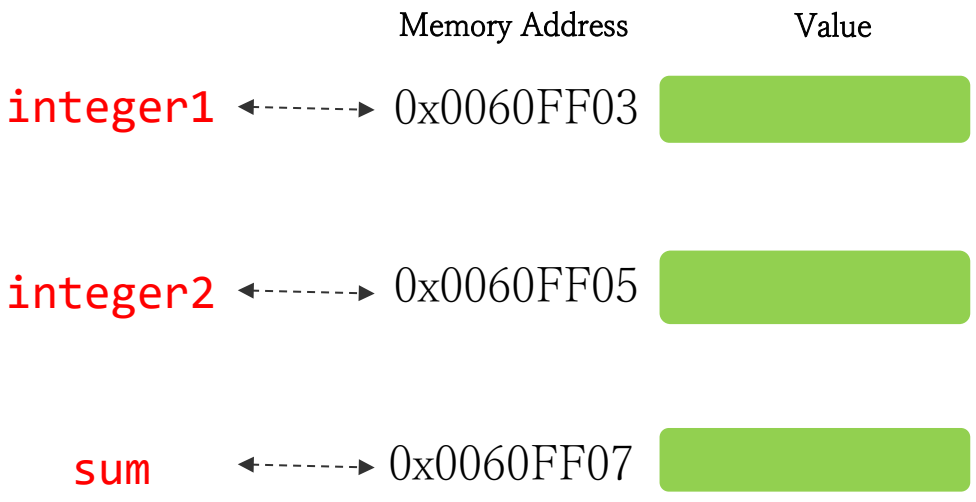
MEMORY CONCEPTS

```
int integer1, inteter2, sum;
```

Every variable has a **name**, a **type** and a **value**.

MEMORY CONCEPTS

```
int integer1, inteter2, sum;
```

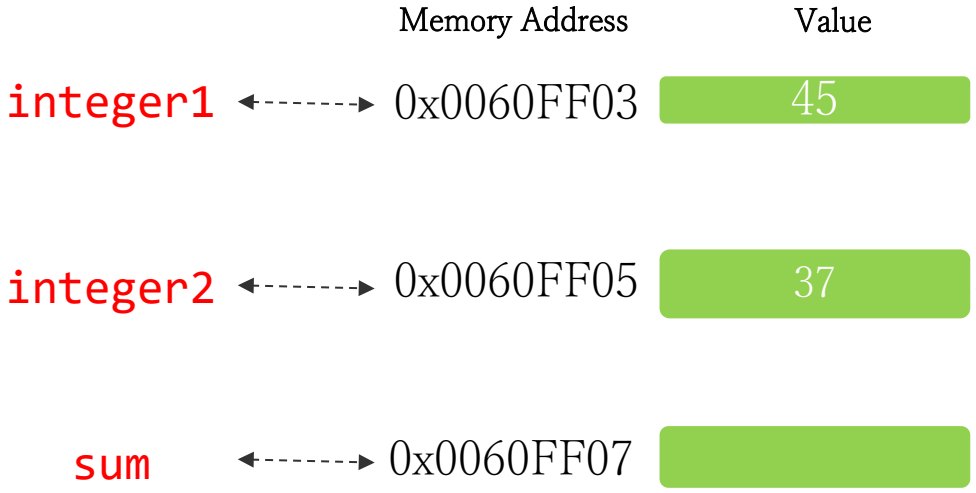


Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer's memory.

MEMORY CONCEPTS

```
scanf("%d%d", &integer1, &integer2);
```

Assume we key in “45” for integer1 and “37” for integer2

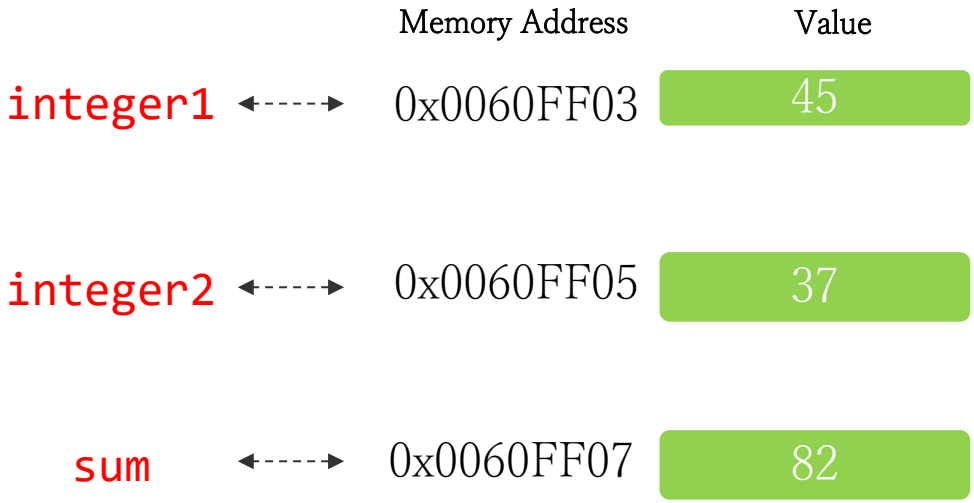


The value typed by the user is placed into a memory location to which the name `integer1` or `integer2` has been assigned.

MEMORY CONCEPTS

```
scanf("%d%d", &integer1, &integer2);
sum = integer1 + integer2;
```

Assume we key in "45" for integer1 and "37" for integer2



Whenever a value is placed in a memory location, the value **replaces** the previous value in that location.

BASIC DATA TYPES

integer: a whole number

floating point value: ie a number with a fractional part.

a double-precision floating point value.

a single character.

valueless special purpose type which we will examine closely in later sections.

EXAMPLE – C DATA TYPES

Code: "datatypes.c"

```
#include <stdio.h>
int main() {

        int      a = 3000;          /* integer data type */
        float    b = 4.5345;        /* floating point data type */
        char     c = 'A';           /* character data type */
        long     d = 31456;         /* long integer data type */
        double   e = 5.1234567890; /* double-precision floating point data type */

        printf("Here is the list of the basic data types\n");
        printf("\n1. This an integer (int): %d", a);
        printf("\n2. This is a floating point number (float): %f", b);
        printf("\n3. This is a character (char): %c", c);
        printf("\n4. This is a long integer (long): %ld", d);
        printf("\n5. This is a double-precision float (double): %.10f", e);
        printf("\n6. This is a sequence of characters: %s",
                "Hello INF1002 students");

        return 0;

}
```

What is the `%.10f` for?

Try to change it to some other number and observe the output

https://alvinalexander.com/programming/printf-format-cheat-sheet

C EXPRESSIONS

Arithmetic can be performed using the usual operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder upon division (modulo)
()	Parentheses

C EXPRESSIONS

The result of an arithmetic operation depends on the type of its operands:

int *op* int	int
int *op* float	float
float *op* float	float
double *op* float	double
char *op* int	char

You can explicitly change the type of an expression using a `cast`:

(int)4.5	round down
(float)4	“promote”

C FORMATTED I/O

SINGAPORE INSTITUTE OF TECHNOLOGY

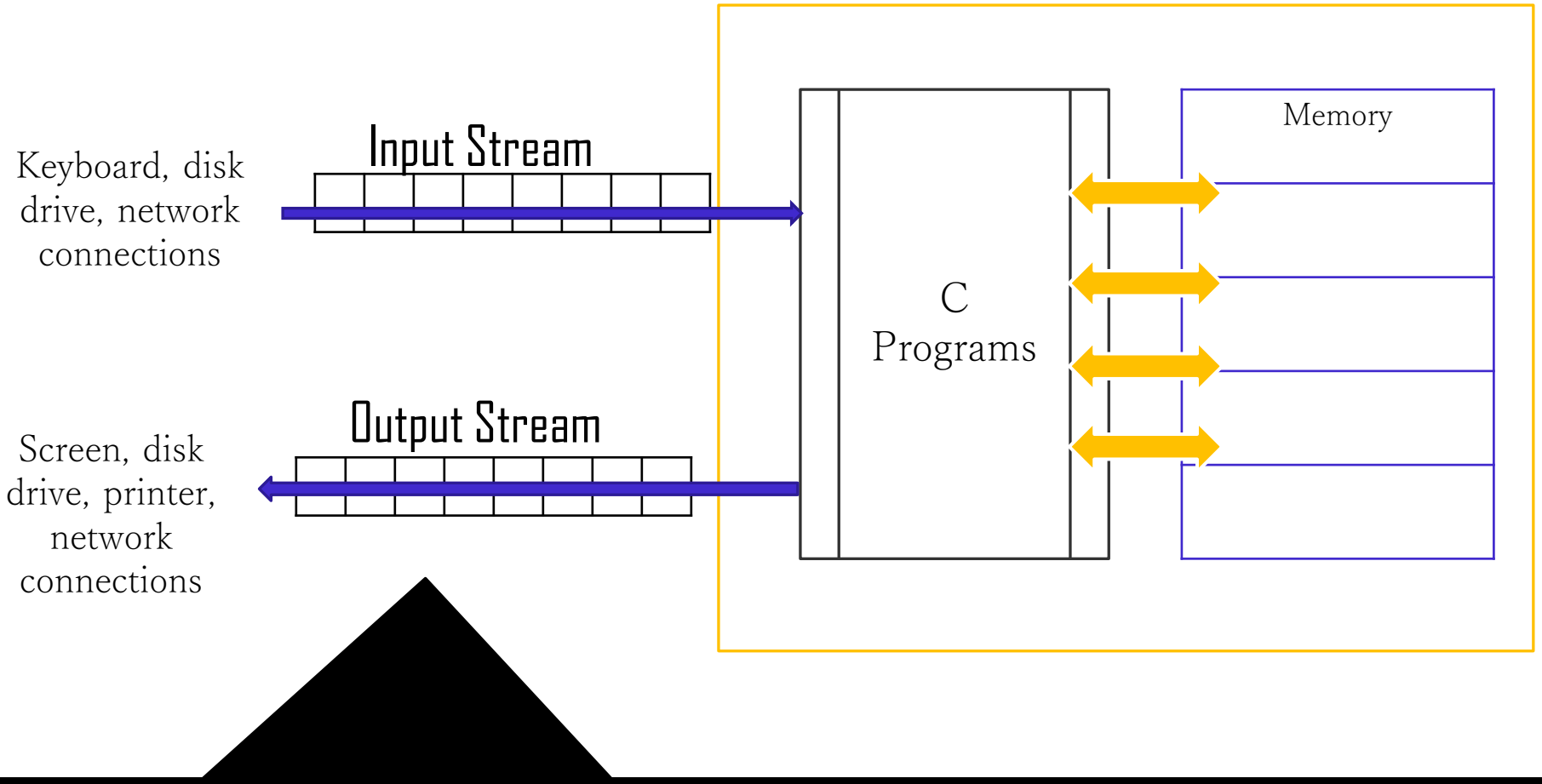
C FORMATTED I/O

Standard input/output:

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

- Input data from the standard input stream
- Output data to the standard output steam
- **`printf, scanf, puts, getchar, putchar`**

STREAMS



All input and output in C is performed with streams.

Stream: A sequence of bytes.

FORMATTING OUTPUT WITH PRINTF

```
printf(format-control-string, other-arguments);
```

- **format-control-string**: describes the output format
- **other-arguments** (optional): correspond to each conversion specification within **format-control-string**

```
printf("%s", "Hello World!");
```

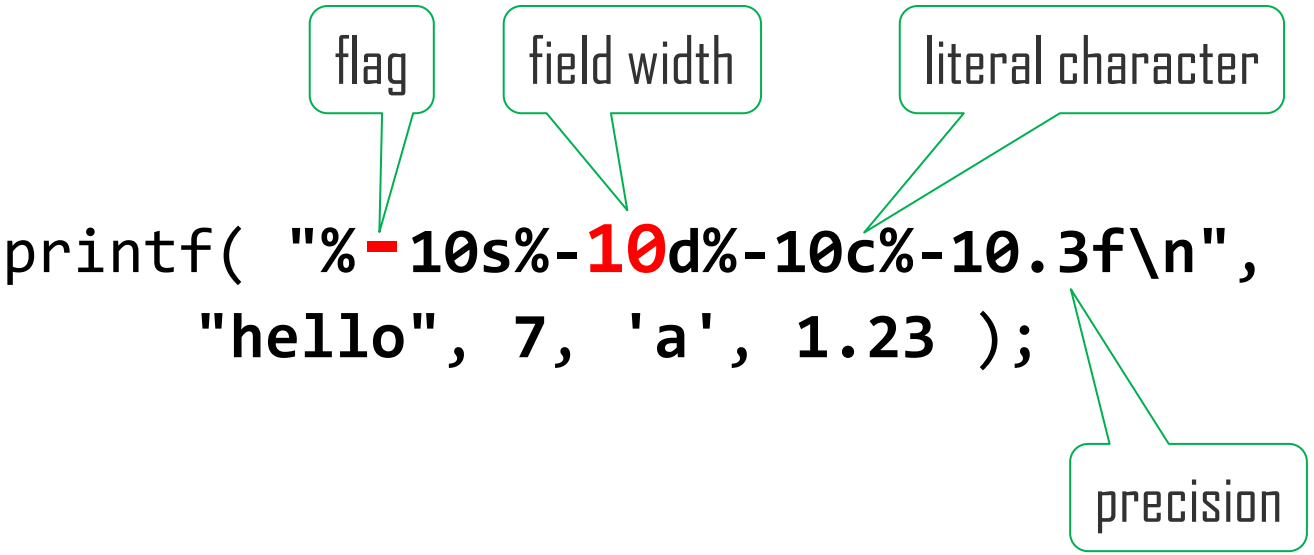
COMMON CONVERSION SPECIFIERS

Specifier	Output
Characters	
`%c`	Character
`%s`	A string of characters
Integers	
`%d`	Decimal integer
`%x`	Hexadecimal integer
`%ld`	Long decimal integer
Floating point numbers	
`%f`	Single-precision
`%lf`	Double-precision

See Deitel & Deitel Ch. 9 for other specifiers.

MORE ON CONVERSION SPECIFIERS

conversion-specifier = <flags><field width><precision><literal character>



PRINTF FLAGS

Flag	Description
`-` (minus sign)	Left-justify the output within the field.
`+` (plus sign)	Insert a plus sign before positive numbers and a minus sign before negative numbers.
space	Insert a space before positive numbers.
`0`	Zero-pad the number to the width of the field.

See Deitel & Deitel Ch. 9 for other flags.

PRINTF FLAGS - EXAMPLE

```
/*
 * Printing field width example adopted from Deitel & Deitel
 */
#include <stdio.h>

int main() {

        printf("%4d\n", 1);
        printf("%04d\n", 1);
        printf("%-4d\n", 1);
        printf("%4d\n", 12);
        printf("%4d\n", 123);
        printf("%4d\n", 1234);
        printf("%4d\n", 12345);

        return 0;
}
```

Output:

```
   1
0001
1
  12
 123
1234
12345
```

Code: “fieldwidth.c”

ESCAPE SEQUENCES

Characters with special meaning to the compiler need to be “escaped”:

Escape Sequence	Output
`\n`	New line
`\t`	Tab
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash

See Deitel & Deitel Ch. 9 for other escape sequences.

SCANF – READING FORMATTED INPUT

```
scanf(format-control-string, other-arguments);
```

```
printf("Enter seven integers: ");
scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g);

printf("The input displayed as decimal integers is:\n");
printf("%d %d %d %d %d %d %d", a, b, c, d, e, f, g);
```

Output

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

NOW TAKE A LOOK BACK AT THE PREVIOUS SAMPLE

```
/*
 * Another simple C program.
 */
#include <stdio.h>

int main() {

        int integer1, integer2, sum;

        printf("Enter two numbers to add\n");
        scanf("%d%d", &integer1, &integer2);

        sum = integer1 + integer2;

        printf("Sum of entered numbers = %d\n", sum);

        return 0;

}
```

CONTROL STRUCTURES

SINGAPORE INSTITUTE OF TECHNOLOGY

C CONTROL STRUCTURES

C has the same control structures as other programming languages:

- `if-else`
- `switch` (not available in Python)
- `for`
- `while`
- `do-while` (not available in Python)

DECISIONS

if-else

```
if (x > 1) {
    printf("More than one.");
} else {
    printf("Not more than one.");
}
```

switch

```
switch (x) {
  case 1:
    printf("x is 1.");
    break;
  case 2:
    printf("x is 2.");
    break;
   …
}
```

LOOPS

for

```
int i;
for (i = 0; i < 10; i++) {
    printf("i = %d\n", i);
}
```

while & do-while

```
int i = 0;
while (i < 10) {
    printf("i = %d\n", i);
    i++;
}

i = 0;
do {
    printf("i = %d\n", i);
    i++;
} while (i <= 10);
```

END-OF-WEEK CHECKLIST

- [ ] C development environment
- [ ] Basic C program structure
- [ ] Comments
- [ ] Pre-processor directives
- [ ] #include
- [ ] #define
- [ ] printf()
- [ ] scanf()
- [ ] Basic data types
- [ ] Variables & memory
- [ ] Streams
- [ ] If/else
- [ ] Switch/case
- [ ] for/while/do..while
- [ ] Coding conventions
- [ ] Format control strings

ADMINISTRATIVE ITEMS

Item	%
Lab Assignments (5)	5%
Group Project	25%
Test (Week 13)	20%
Sub-total	**50%**

- Group project
  - Project description will be uploaded to LMS in Week 9.
  - Same grouping as Python
- Test (Week 13)
  - Physical

ABOUT USAGE OF AI TOOLS

- For Labs
  - AI tools (e.g. ChatGPT, Copilot, etc.) are allowed for reference purpose
  - However, strongly recommend hands-on learning

- For Group Project
  - AI tools are NOT allowed
  - A declaration is needed from each team

- For Test
  - AI tools are NOT allowed

ABOUT LAB

- Start from Week 8 (today)
- Conducted in classroom
- Follow the instructions on the lab sheet
  - A few exercises to be completed in lab class
    - Strongly recommend to complete these exercises in the lab and you may consult the instructor if any help is needed
  - Assignments