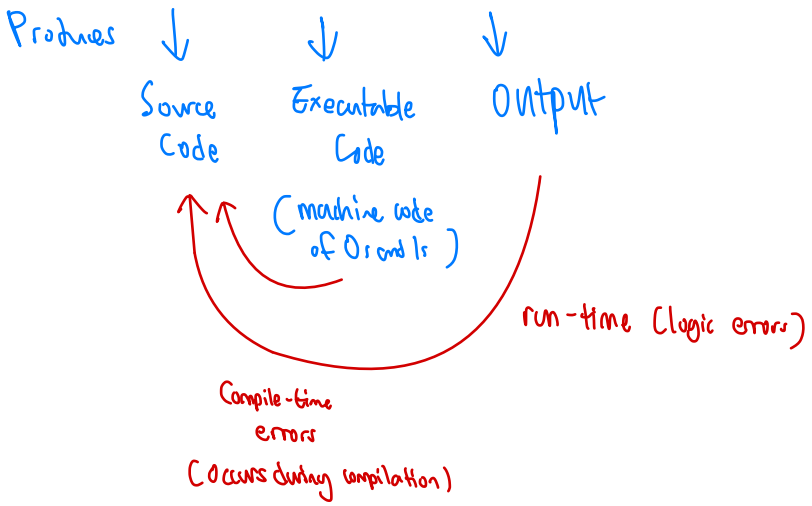
C Programming

- C: Compiled language (C to Assembly, then machine code)

- Directly understood by computer → fast.

- Edit, Compile, Execute



General Form:

pre-processor directives — instructions to compiler

eg. library headers — #include
- header file
- ↳ Contains prototypes (of function and constants)
  - ↳ tells us what a function expects in terms of its argument and what it returns.
  - <stdio.h>
    - ↳ printf()
    - scanf()

define constants — #define

macros

not optional * main function header

{

declaration of variables — declare types → know what machine instructions to generate later on.

executable statements
- Input data
- Computation
- Output results → returns 0 → if executed successfully
  - → status of execution

}

von Neumann Architecture / Stored-Program Architecture

CPU → 1. Registers – special, fast, expensive type of memory.
– inside processors → store temp. results of computation.

2. Control Unit – contains instruction register – stores current instruction
– program counter – tells you where to find next instruction in the memory

3. Arithmetic/Logic Unit (ALU) – defines various unit to perform additions, multiplications, comparisons, decision making...

Memory → where programs and variables (data.) are stored.

[RAM] → C: content of variable unknown on declaration (not zero)

RAM – read + write

ROM – read only

Flash – read + write, special because retains memory after restart.

I/O devices → keyboard, screen, mouse etc.

Variables

- used in a program to store data
- name + data type + value
  - name: identifier
  - value: can always be modified
- addresses

variable stored in memory
→ put into slots
→ every slot has an index number
→ index number aka address
→ to retrieve variable, specify address number

index number = addresses

- If initial value (initializer) not specified → unknown value (not zero)
- gcc -Wall
  - "warn for all" mistakes
- gcc a.c -o a ⇒ a.out
  - specifies name of executable file.

Data Types

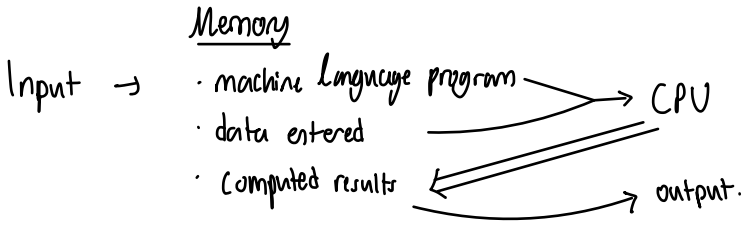
- every variable has to be declared with a data type
- determines the kind of data the variable can hold.
- C makes use of data types to determine assembly instruction to generate.
- int: -2^{31} to $2^{31}-1$ (4 bytes - 32 bit)

  float / double
  - 4 bytes
  - 8 bytes → more memory, much slower operation

  Char: holding a character in a single quote
  also a very small integer (8 bit integer)
  ∴ everything is a number → ASCII
- Strongly typed vs Weakly typed
  - variable is declared with a type
  - depends on how variable is used.
- C supports implicit type conversions / type-casting

```
float x = 1.5
int y = (int)x   → turns value in x into integer
y = 1            → truncate 0.5.
```

Program Structure



Preprocessor directives → #include <stdio.h>, #include <math.h>, #define PI 3.142

* · Inclusion of header files
  - → stdio.h - scanf, printf (req: args, return vals.)
  - → functions from certain libraries
    - eg. math.h → specify -lm option in compilation
      string.h

* · Macro expansions → defining constants, can take in arguments
  - → #define NAME substituted_value.
  - → tag substitution → every instance of NAME sub with sub_val.
  - → no semi-colon!

· Conditional compilation — under certain circumstances will produce certain code
⇒ code produces changes with different compilation

Input → stdin, scanf

Compute → ???

Output → stdout, printf

Input/Output:

scanf (format string, input list) — eg. &age, &cap (use comma.)

printf (format string)

printf (format string, print list)

= for a function to write into a variable, needs to know address of variable.
⇒ go to address and put into variable
⇒ else runtime error!

eg. %d → int
%lf → double (scan)
%f → float or double (print), float (scan)
%c → char
%e → float or double (print in scientific notation)

widths
eg. %5d → __123 (min width)
%8.3f → min width of 8 with 3 d.p.
} printf only

Escape sequences:
\n, \t, \", %%
0, 7, 15. / \ double quote \ percent symbol

Conventions

- name: · does not begin with a digit,
  (user defined id) · begins with lowercase
  - If constants – capital letter.
  - var and func – avoid same name.
- assignments: · Left side of "=" known as *lvalue* (must be assignable)
  - Can be cascaded, associativity right to left — side effect:
    - returning the value of RHS
    - avoid convoluted code.

i.e. `a=b=c=3+6`

```
a=(b=(c=3+6));
   └ c=3+6;
   → b=c;
   → a=b;
```

- precedence:

(highest) primary expression operators:	(), exp++, exp--	L→R
unary operators:	*(ptr.), &(addr.), +(+ve), -(-ve value), !, ++exp, (typecast)	L←R
binary operators:	*, /, %,	L→R
	+, -	
assignment operators:	=, +=, -=, *=, /=, %=	L←R

Difference btw a++ and ++a-

eg. `int a=5;`

`b=a++;` ⇒ assigns value to b first, then increment a.

`b=++a;` ⇒ a is incremented first, then assigned to b.

- Mixed Arithmetic:

`int m = 10/4` → m=2 (2)

`float p = 10/4` → p=2.0 (2)

`int n = 10/4.0` → n=2 (2.5)

`float q = 10/4.0` → p=2.5 (2.5)

`int r = -10/4.0` → r=-2 (truncated) (-2.5)

- Type casting:

```
int aa = 6
float ff = 15.8
float pp = (float) aa/4   → pp=1.5.   (aa is float)
int nn = (int) ff/aa      → nn=2      (ff is int)
float qq = (float) (aa/4) → qq=1.0    (int ⇒1)
```

- % in C: gives a remainder, i.e. -10 % 4 = -2 *

- everything in C is a number: 0 = false *
  any other valr. = true.

- Shortcut Circuit Evaluation

eg. `if ( (a != 0) && (b/a > 3) )`

(a != 0: False; (b/a > 3): not evaluated ⇒ Lazy evaluation (Once outcome known, Stop evaluation))

Works via assembly language.

Break → only break out inner most loop.

- Precedence for Boolean Expression:

(highest) primary expression operators:	(), [], ., →, exp++, exp--	L→R
unary operators:	*, &, +, -, !, ~, ++exp, --exp, (typecast), sizeof	L←R
binary operators:	*, /, %	L→R
	+, -	
	<, >, <=, >=	
	==, !=	
	&& * higher precedence than ||	
	||	
ternary operators:	? :	L←R
assignment operators:	=, +=, -=, *=, /=, %=	L←R

Parenthesis!