การตั้งชื่อตัวแปร

- อ่านเข้าใจง่าย
- สื่อความหมายให้ชัดเจน
- ต้องประกาศก่อนใช้งานเสมอ
- "_" นับเป็นตัวอักษรนะ
- อย่าตั้งชื่อ keyword
- ปกติแล้วจะไม่ใช้อักษรใหญ่ในตัวแรก ยกเว้น เช่น P, Q, S, A

Snakecase = Student_ID
Camelcase = StudentID
camel = studentID

ข้อมูลและขนาด

ชนิดของตัวแปร	ขนาด (bits)	ขอบเขต	ข้อมูลที่เก็บ
char	8	-128 ถึง 127	ข้อมูลชนิดอักขระ ใช้เนื้อที่ 1 byte
unsigned char	8	0 ถึง 255	ข้อมูลชนิดอักขระ ไม่คิดเครื่องหมาย
int	16	-32,768 ถึง 32,767	ข้อมูลชนิดจำนวนเต็ม ใช้เนื้อที่ 2 byte
unsigned int	16	0 ถึง 65,535	ข้อมูลชนิดจำนวนเต็ม ไม่คิดเครื่องหมาย
short	8	-128 ถึง 127	ข้อมูลชนิดจำนวนเต็มแบบสั้น ใช้เนื้อที่ 1 byte
unsigned short	8	0 ถึง 255	ข้อมูลชนิดจำนวนเต็มแบบสั้น ไม่คิดเครื่องหมาย
long	32	-2,147,483,648 ถึง 2,147,483,649	ข้อมูลชนิดจำนวนเต็มแบบยาว ใช้เนื้อที่ 4 byte
unsigned long	32	0 ถึง 4,294,967,296	ข้อมูลชนิดจำนวนเต็มแบบยาว ไม่คิดเครื่องหมาย
float	32	3.4*10e(-38) ถึง 3.4*10e(38)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 4 byte
double	64	3.4*10e(-308) ถึง 3.4*10e(308)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 8 byte
long double	128	3.4*10e(-4032) ถึง 1.1*10e(4032)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 16 byte

Signed = มีค่าเท่าเดิม

Void = ไม่ได้รับค่าตัวแปร แต่ใช้กับฟังก์ชันได้ ในกรณีที่ไม่ให้รับหรือส่งค่าเข้ามาในฟังก์ชัน

sizeof = ระบุขนาดของข้อมูล → str[32] ขนาด

Overflow ข้อมูลล้นเกิน

ค่าคงที่

ตัวแปร แต่เปลี่ยนแปลงค่าไม่ได้

Const datatype identifier = value;

- Const ↑ ประกาศว่าเป็นค่าคงที่
- datatype ↑ ชนิดข้อมูล char, int, float
- identifier ↑ ชื่อตัวแปร
- value ↑ ค่าที่ต้องการให้เก็บ

valueจะมี อักษรบางอย่างเพื่อบอกว่าเป็น constantอะไร

เช่น long constant จะมี l,L อยู่ด้านหลัง เช่น 123456789l, 123456789L

unsigned long constant = Ul ลงท้าย

0x 0X จะบ่งบอกว่าเป็นเลขฐาน16 อยู่ด้านหน้าเลข = 0x1f

0b 0B = Binary

0 = Octal

การprintf สามารถใช้ %o (ฐาน8) และ %x (ฐาน16) ได้

แบบ Macro เราใช้#defineได้

#define identifier value

จะระบุ ชนิดของตัวแปรให้เองตาม ชนิด valueที่เจอ

char 'A'+1

= 66

เพราะแปลง A เป็น 65 (ASCII)

นำมา +1 เป็น 66

* constant ใช้กับ Arrayได้ เพื่อไม่ให้ array เปลี่ยน

2.1 รหัสควบคุมในภาษา C

- \a ส่งเสียง Beep
- \n ขึ้นบรรทัดใหม่
- \t แท็บในแนวนอน
- \b ย้อนกลับไป 1 ตัวอักษร
- \v แท็บในแนวตั้ง
- \f ขึ้นหน้าใหม่
- \r รหัส Return
- \' แทนตัวอักษร Single Quote(')
- \'' แทนตัวอักษร Double Quote('')
- \\ แทนตัวอักษร Backslash(\)
- \000 แทนตัวอักษรที่มีค่า ASCII เท่ากับ 000 ในระบบเลขฐานแปด
- \xhh แทนตัวอักษรที่มีค่า ASCII เท่ากับ hh ในระบบเลขฐานสิบหก

Operator

ลำดับความสำคัญ
ทำอันไหนก่อน

? อ่าน

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
2	++ -- () [] . ->	Suffix/postfix increment and decrement Function call Array subscripting Element selection by reference Element selection through pointer	
3	++ -- + - ! ~ (*type*) * & sizeof new, new[] delete, delete[]	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of Dynamic memory allocation Dynamic memory deallocation	Right-to-left
4	.* ->*	Pointer to member	Left-to-right
5	* / %	Multiplication, division, and remainder	
6	+ -	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	== !=	For relational = and ≠ respectively	
10	&	Bitwise AND	
11	^	Bitwise XOR (exclusive or)	
12	|	Bitwise OR (inclusive or)	
13	&&	Logical AND	
14	||	Logical OR	
15	?: = += -= *= /= %= <<= >>= &= ^= |=	Ternary conditional Direct assignment (provided by default for C++ classes) Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	Right-to-left
16	throw	Throw operator (for exceptions)	
17	,	Comma	Left-to-right

คำสั่ง (stdio.h)

printf() = print แบบอย่าง

getchar() = รับตัวอักษรตัวแรก

puts() = print string

putchar() = แสดงผล อักษรใดๆ

// = comment 1 บรรทัด

/* = comment หลายบรรทัด ปิดด้วย */

```
for (x=0; x<10; x++){ }
```

x++ = ++x = x+1

x-- = --x = x-1

- x=0 : กำหนดค่าตัวแปร
- x<10 : เงื่อนไข
- x++ : ทำแบบอย่าง เพิ่มค่าให้ตัวแปร x ทีละ 1

ผลลัพธ์:

x=0
x=1
x=2
x=3
x=4
x=5
x=6
x=7
x=8
x=9

```
if (เงื่อนไข){}
```

ทำแบบอย่าง

Type Conversions

การแปลงประเภทข้อมูล ประเภท narrow (ต่ำสุด) กับ wider (กว้างขึ้น) เพื่อไม่ให้สูญเสียข้อมูล

โดย แบ่งเป็น 2 แบบ

Implict conversion

แปลงให้อัตโนมัติ เช่นการกำหนดค่าตัวแปร

```
int a=10.5; a=10
```

หรือ การ float + int จะกลายเป็น float + float

logic ก็กลายเป็นตัวเลขได้

false = 0 * ถ้าค่า = 0 จะแปลงเป็น false
true = 1 ถ้าค่า ≠ 0 เวลาแปลงจะเป็น true

Type casting

แปลง float เป็น int แต่จะทำให้ทศนิยมหาย

2.5 → 2 ค่าจะผิดเพี้ยนไป

Increment and decrement

n++
++n } เพิ่ม 1

n--
--n } ลด 1

Bitwise operator

& (and) 101 & 111 → 101

| (or) 101 | 100 → 101

<< (left shift) 1101 << 2 = 0100
>> (right shift) 1101 >> 2 = 0011
} 0 เท่ากับจำนวนที่ว่าง

^ (xor) 1 ^ 0 = 1 0 ^ 0 = 0 1 ^ 1 = 0

~ (not) กลับค่า ~101 = 010

Assignment operator

= นั่นแหละค่ะ บอกว่า ตัวแปร มีค่าหน้าที่อะไรบ้าง

Condition Expression

```
z = (a > b) ? a : b
```

True z = a

false z = b

Statement and blocks

expression ต้องมี ; (semi colon) ถึงจะเป็น statement ได้

ประกาศตัวแปร

ยกเว้น compound statement และ declaration เข้าด้วยกัน

Null statement คือ statement ที่ไม่มี expression

คือ ; โง่ๆไม่มีอะไรเลย

คล้าย pass ใน python

if else

```
if (expression) {      ← เงื่อนไข
    statement;
    }
else if (){  ;
    }
else (){  ;
    }
```

Switch

ตัวแปรที่เป็นเลขจำนวนเต็มได้

ค่าคงที่ เลขจำนวนเต็ม

ห้ามเป็น float

```
switch (expression) {
    case    constant expression:
            statement

    case    constant expression:
            statement

    default:
        statements
    }
```

ถ้าเข้า case ก็จะทำงานต่อไปเรื่อยๆ จนกว่าจะหลุดหรือ break
default มีหรือไม่มีก็ได้ โดยถ้าไม่เข้า case ใดเลยจะเข้า default แทน
case ห้ามซ้ำกัน

loop while

while (เงื่อนไข)
    statement

ถ้า false จะหลุดลูปทันที

loop for

ไม่จำเป็นต้องมี ; ก็ได้

for (exp1 ; exp2 ; exp3)
    statement

ถ้า for(;;) จะ infinity loop

ถ้า exp2 = false จะหลุดลูป

loop do - while

do {
    statement
} while (expression);

statement จะทำอย่างน้อย 1 ครั้ง ก่อนเช็คเงื่อนไข

break continve

break หยุดลูปทันที

continve หยุดลูปกลับมาใหม่ while และ Do while
ถ้ากลับไปเงื่อนไข ถ้า True วนกลับไปทำงานต่อ
ถ้าไม่หยุดลูป

Goto label

goto วนไปยังบรรทัดเป้าหมาย

Function Definition

ประเภทของข้อมูลที่ return ชื่อฟังก์ชัน (พารามิเตอร์)
{ ประกาศตัวแปร

statement
;
}

ถ้าไม่รับ parameter ให้ใส่ void

Function prototype

return value type name (parameters)

ถ้า ไม่ระบุ compiler จะเดาว่าเป็นชนิด int

Function call - Call by value

เรียกใช้ฟังก์ชัน แล้วมันทำงานยังไง...

Standard library function and Math library

ฟังก์ชันต่างๆ printf() scanf() getchar()

อยู่ใน standard library ถ้าฟังก์ชันพวกนี้เกี่ยวข้องกับ Math

library

ใน Unix (รวมถึง linux Macos Android อื่นๆ)

จะเก็บไลบรารีไว้ใน /user/lib/ หรือ /user/lib64/

ของ Math คือ /user/lib/libm.a

return value

ฟังก์ชันต้องมี return ส่งกลับไปให้ที่เรียกใช้ยกเว้นชนิด void

โดย void จะบอกว่าไม่ return นะ

Recursion การเรียกซ้ำ

ฟังก์ชันที่เรียกใช้ตัวเองได้ (direct)
ฟังก์ชันที่เรียกฟังก์ชันอื่น แล้วฟังก์ชันนั้นเรียกตัวเอง (indirect)
เช่น การหา factorial
ต้องกำหนดเงื่อนไขในการหยุดเรียกด้วยนะ

ตัวแปร local และ Global

local มีเฉพาะในบล็อกนั้นเท่านั้น (ฟังก์ชัน)

Global ประกาศนอกฟังก์ชันทุกฟังก์ชันเรียกใช้ได้

ถ้า local ชื่อซ้ำกับ Global การแก้ไขค่าตัวแปร local ไม่ส่งผลอะไรต่อ Global

parameter นับเป็น local

local ถ้าใช้เสร็จแล้วจะคืนค่าความจำ

Memory จะแบ่งเป็นส่วนๆ เรียก segment
segmentation fault = เข้าไปในส่วนที่เราไม่ได้เป็นเจ้าของ จะฟ้องโปรแกรมเราทันที

Text segment/code segment เก็บ Machine code
เก็บการทำงานของคำสั่งต่างๆ

global { uninitialized data (bss) = ตัวแปรที่ยังไม่ประกาศค่าของการเก็บใช้ที่นี่
initialized data = ตัวแปรที่ประกาศค่า

local เก็บใน stack

memory จะมี low และ High address

keep ของ mem เพิ่มในขณะทำงาน โดยไม่ได้ประกาศตัวแปร

ถ้า stack ชนกับ keep โปรแกรมจะตาย

stack โตลงด้านล่าง มีพื้นที่จำกัด ถ้า stack เกินจะเรียกว่า stack overflow

keep จะโตด้านบน

Global ถ้าไม่กำหนดค่าเริ่มต้น จะมีค่าเป็น 0

local ในแลตต จะมีค่าเท่ากับค่าที่ค้างอยู่ใน memory ณ ตอนนั้น

Function call และ stack frame

เวลาเราเรียกใช้ฟังก์ชั่นจะทำการจองพื้นที่ใน stack และ push 1 ที่
ถ้าเกิดว่าทำงานเสร็จแล้ว ฟังก์ชันจะ pop ออก

แต่ละ stack เรียก stack frame หรือเรียกว่า activation record

return address จะเก็บอยู่บน stack frame เพราะหลักจากที่เราประมวลผลฟังก์ชั่นเสร็จแล้ว จะต้องทำงานต่อไป

ใน stack frame มี return address, ตัวแปร local, พารามิเตอร์ ของฟังก์ชั่น

Storage class

แบ่งเป็น 4 class
มี auto, register, extern, static

มี 2 duration
automatic storage duration ↙ local อยู่ในนี้และเหมือนกัน
ตัวแปรเกิดเอง ตายเอง เราไม่เกี่ยว
static storage duration (อยู่อย่างถาวร
จนกว่าโปรแกรมจะตาย)
auto และ register เป็น asd (เป็น local ทั้งคู่
register ต้องประกาศ ก่อนถึงจะเป็น register การประกาศ register
นั้นคือการขอ register จาก compiler
*มีจำกัด บางครั้งต้องจำแปลงครึ่งบนครึ่งหลัง
เข้าถึงข้อมูลได้เร็วมาก
ถ้าขอregister ไม่ได้จะแปลงเป็น auto แทน
auto อยู่ใน stack frame
เมื่อประกาศค่าจะเป็น extern โดย default
static เป็นตัวแปรที่ใช้อยู่ใน local ทำให้ มีชีวิตอยู่จนสิ้นสุด
process
extern ใช้ได้ในกรณีที่ยังไม่ประกาศ ตัวแปร globalได้เพื่อให้
compiler ปล่อยผ่านโดยห้ามกำหนดค่าเริ่มต้นก่อน

Scope rules (ขอบเขตและช่วงชีวิตของตัวแปร)

Local variable ประกาศใน block ใช้ได้แค่ใน block

Global variable ประกาศนอก ฟังก์ชั่น ใช้ได้ทุกที่

storage classes

มี 4 class auto register extern statics

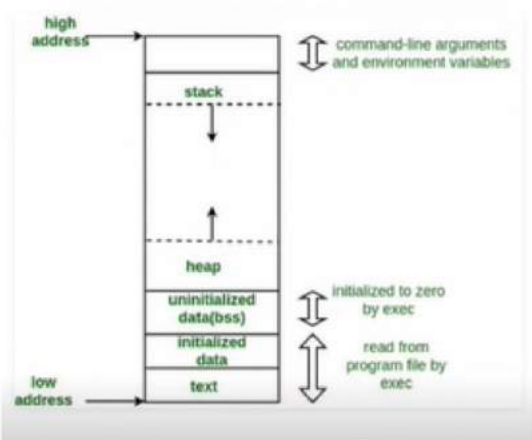
แบ่งตาม ที่เก็บข้อมูล

automatic storage duration เกิดอัตโนมัติ

statics storage duration เมื่อจบการทำงาน ก็จบไป พร้อมกัน

Storage classes in C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block



Array

ถ้าไม่ใส่จะกำหนดค่าให้เองตามในนี้

int A [10] = {0, ..., 9}

ถ้าไม่กำหนดค่าหรือค่าไม่ครบที่เหลือจะเป็น 0 โดยอัตโนมัติ

ขนาด Array ควรเป็นค่าคงที่ (constant)

ใช้ # define เป็นคำสั่ง (directive) ที่สั่งให้ complier

เปลี่ยนตัวแปร เป็นค่านั้นๆ ก่อน complie

const int ชื่อarray = 5;

ต้องประกาศค่าทันทีตอนสร้าง

เปลี่ยนแปลงค่าไม่ได้

#define จะเปลี่ยน ตัวแปรทั้งหมดเป็น ค่านั้นๆ

const เหมือน ตัวแปรตัวหนึ่งแต่เปลี่ยนค่าไม่ได้ แค่นั้น

เก็บในหน่วยความจำ

คล้าย Global variable

ให้ user กำหนดขนาด Array

* ใช้ได้เฉพาะ complier มาตรฐาน C99 เท่านั้น

โดยเป็นการเปลี่ยนตัวแปรตำแหน่งจริง (int) ตามปกติ

ระวัง

C ไม่เช็คขนาด Array ให้ แม้ complie ผ่าน แต่จะมี bug มีโอกาสจะตั้ง อ่าน ค่าอะไรก็ไม่รู้มาให้ ทำให้เกิด buffer overflow

pass by value v.s. reference

pass by value	pass by reference
ก็อปปี้ค่าของตัวแปรจากผู้เรียกไปให้ฟังก์ชัน	ส่งตำแหน่งของตัวแปรจากผู้เรียกไปให้ฟังก์ชัน
สร้างตัวแปร local variable ใหม่ในฟังก์ชัน	ตัวแปรในฟังก์ชันเป็นตัวเดียวกันกับในผู้เรียก
การเปลี่ยนแปลงค่าในฟังก์ชันไม่ส่งผลถึงตัวแปรต้นตำรับ	การเปลี่ยนแปลงค่าในฟังก์ชันเปลี่ยนตัวแปรต้นตำรับด้วย

ส่งทั้ง Array ให้ส่งชื่อ Array เลย โดยเวลาสร้าง parameter ต้องมี

function (int array[], int size_array);

A[0], A[1], A[2], ..., A[N]

เป็นตัวแปร

สรุป

* **pass by value:** ส่งค่าไปยังฟังก์ชัน ใช้งานได้เลย
* **pass by reference:** ส่งตำแหน่งไปยังฟังก์ชัน ← ต้องมาหาค่าอีกที
  * ดังนั้น **array** กับ **A** ชี้ไปยังตำแหน่งเดียวกัน
  * **A[2]** กับ **array[2]** ก็อยู่ตำแหน่งเดียวกัน
  * เปลี่ยนค่าที่เก็บไว้ใน **array[2]** ก็เหมือนเปลี่ยนค่าของ **A[2]**

Array 2 มิติ

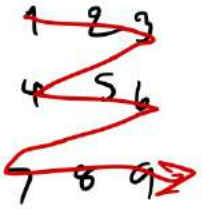
int A [2][3]; สร้างไพ่ทางขนาด

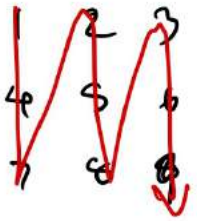
↑ ↑

row column

Array เก็บ array ได้

row major column major





A[][]; ✗ ไม่ได้เพราะว่า complier ไม่รู้ว่าจะสิ้น row ไหน เสียไหน่

มากกว่า 2 มิติ ก็เหมือนๆ ดีเหมือนกัน

Structure

เป็นตัวแปรที่เก็บข้อมูลหลายๆตัวไว้ด้วยกัน

ควรมาตอนใช้ Typedef เพื่อให้เรียกใช้งานง่ายขึ้น โดย

typedef struct ชื่อ ชื่อเรียก ;

เวลาใช้งานเหมือน

struct ชื่อ ข้อมูลต่างๆ... ;

เป็น

ชื่อเรียก ข้อมูลต่างๆ... ;แทน

เข้าถึงข้อมูลได้โดย .

first . ชื่อตัวแปร = ... ;

↑ structure_variable

pointer

เป็นตัวแปรที่อ้างถึง address ของหน่วยความจำ

&c = ได้ address ของ c

* จะเป็นการบอกให้เข้าไปในพื้นที่เก็บข้อมูลนั้นๆ

อย่าง p = &c จะเก็บ address ของ c

แต่ *p = &c จะเข้าไปในตำแหน่ง address ตัวที่อยู่ที่ c=5

*p = 5

การจองพื้นที่ในหน่วยความจำ

malloc ( ) จำนวน byte ที่ต้องการจองพื้นที่

free (data) ส่วนที่ต้องการคืนให้ระบบ