Chapter 1

Intro to C/C++

ในหัวข้อนี้ เราจะแนะนำเกี่ยวกับการเขียนโปรแกรมภาษา C และ C++ ในเบื้องต้น เพื่อเป็นพื้นฐานสำหรับการเขียนโปรแกรมที่ซับซ้อนในหัวข้ออื่นๆ ต่อไป

1.1 Syntax

```c
#include <stdio.h>

int main() {
  return 0;
}
```

โปรแกรมด้านบนนี้เป็นส่วนของโปรแกรมที่ต้องมีสำหรับการเขียนโปรแกรมภาษา C เพื่อให้การทำงานของโปรแกรมเป็นไปได้อย่างปกติ

บรรทัดที่ 1: **`#include`** คือการนำ header file library เข้ามาใช้ในโปรแกรม โดย header files มีหน้าที่เพิ่มฟังก์ชั่นการทำงานของโปรแกรมให้สามารถทำงานได้ตามวัตุประสงค์ของเรา ในที่นี้ ”stdio.h” เป็นชื่อที่ย่อมาจาก ”Standard Input Output” ซึ่งเป็น header file สำคัญในการรับข้อมูลนำเข้าและส่งออก

บรรทัดที่ 3: เป็นการประกาศฟังก์ชั่นหลักที่ใช้ในการทำงานของโปรแกรม ทุกครั้งที่โปรแกรมเริ่มทำงานจะเริ่มทำงานที่ฟังก์ชั่น main นี้ก่อนเสมอ

บรรทัดที่ 4: คำสั่ง **`return 0;`** เป็นคำสั่งเพื่อจบการทำงานของโปรแกรม โดย 0 เป็นรหัสคำสั่งที่โปรแกรมส่งออกให้ทราบว่าการทำงานของโปรแกรมนี้มีความผิดพลาดเป็น 0 (ไม่มี error)

หมายเหตุ: จะสังเกตเห็นได้ว่าคำสั่งของภาษา C/C++ จะลงท้ายด้วย semicolon (**`;`**) เสมอ และขอบเขตการทำงานจะถูกระบุด้วยเครื่องหมายปีกกา (**`{ }`**)

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  return 0;
}
```

โปรแกรมด้านบนนี้เป็นส่วนของโปรแกรมภาษา C++ โดยสังเกตว่าจะคล้ายกับภาษา C มาก แตกต่างกันเพียงบรรทัดที่ 1 และมีบรรทัดที่ 2 เพิ่มมา

บรรทัดที่ 1: "bits/stdc++.h" เป็น header file ของภาษา C++ ที่รวมหลาย header files ที่เป็นประโยชน์เข้าไว้ด้วยกัน เพื่อเพิ่มความสะดวกของผู้ใช้ (ไม่ต้องทำการ **`#include`** หลายๆ รอบ)

บรรทัดที่ 2: `using namespace std;` เป็นคำสั่งเพื่อละการเขียน "**`std::`**" หน้าบางคำสั่ง เพื่อให้สามารถขียนโปรแกรมได้รวดเร็วยิ่งขึ้น

1.1.1 Comments

การเขียนคำอธิบายในภาษา C/C++ สามารถทำได้ 2 วิธี

1. Single-line comments เป็นการเขียนคำอธิบายแบบบรรทัดเดียว ใช้เครื่องหมาย **`//`** ไว้หน้าข้อความที่ต้องการให้กลายเป็นอธิบาย และไม่ถูก compile

   ```cpp
   // This is a single-line comment
   #include<bits/stdc++.h> //Can be used after each line too
   ```

2. Multi-line comments เป็นการเขียนคำอธิบายแบบหลายบรรทัด ใช้เครื่องหมายเริ่ม **`/*`** และเครื่องหมายจบ **`*/`** โดยข้อความที่อยู่ระหว่างเครื่องหมายทั้ง 2 จะกลายเป็นคำอธิบายที่ไม่ถูก compile

   ```cpp
   /*
   This is the first line of multi-line comment
   This is the second line of multi-line comment
   */
   ```

1.2 Variables

ในภาษา C/C++ มีตัวแปรที่เก็บข้อมูลได้หลากหลายประเภทด้วยกัน

1.2.0.1 Basic Data Types

Data type	Size	Description
`boolean`	1 byte	เก็บค่าความจริง (true / false)
`char`	1 byte	เก็บตัวอักขระ 1 ตัว
`int`	2 or 4 bytes	เก็บจำนวนเต็ม
`double`	8 bytes	เก็บจำนวนทศนิยม ความแม่นยำไม่เกิน 15 หลักทศนิยม

นอกจากนี้ ยังมีการนำ `char` มาประยุกต์เป็น array of char (`char[]`) เพื่อให้สามารถเก็บชุดอักขระ (String) ได้

1.2.1 Variables Declaration

การประกาศตัวแปรสามารถทำได้โดยการระบุประเภทตัวแปร ชื่อตัวแปร เป็นเบื้องต้น ซึ่งสามารถประกาศได้ 2 วิธี ได้แก่

1. การประกาศตัวแปรพร้อมระบุค่า

```
int firstNumber = 10;
```

2. การประกาศตัวแปรแล้วระบุค่าภายหลัง

```
double secondNumber;
secondNumber = 10.2;
```

หมายเหตุ: การระบุค่าใหม่ไปยังตัวแปรที่มีค่าอยู่แล้ว จะเป็นการเขียนค่าใหม่ทับลงไป ค่าเก่าจะหายไป

```
int thirdNumber = 10;
thirdNumber = 15;
//thirdNumber's current value is 15
```

นอกจากนี้ การประกาศตัวแปรประเภทเดียวกันหลายตัวแปรสามารถทำได้ในบรรทัดเดียวกัน โดยใช้เครื่องหมาย comma (`,`) ในการคั่นระหว่างตัวแปร

```
int w = 1, x = 2;
```

การระบุค่าเดียวกันให้กับตัวแปรหลายตัวแปรก็สามารถทำได้ในบรรทัดเดียวกันได้

```
int y, z;
y = z = 3;
/*
same result as
y = 3;
z = 3;
*/
```

การประกาศตัวแปรค่าคงที่สามารถทำได้คล้ายกับการประกาศตัวแปรทั่วไป เพียงแค่เพิ่ม **`const`** ไปข้างหน้าประเภทตัวแปรนั้น โดยตัวแปรค่าคงที่จะไม่สามารถแก้ไขได้หลังจากการประกาศ จึงต้องทำการประกาศตัวแปรพร้อมระบุค่า

```
const int maxNumber = 100;
maxNumber = 10; //error: can\'t change constant variable's value
```

1.2.1.1 General rules for variables naming

1. ชื่อตัวแปรสามารถมีตัวอักษร, ตัวเลข, และ underscore (_) เท่านั้น ไม่สามารถมีช่องว่างหรือตัวอักษรพิเศษ เช่น `!`, `#`, `%` เป็นต้น
2. ชื่อตัวแปรต้องขึ้นต้นด้วยตัวอักษรหรือ underscore เท่านั้น ไม่สามารถขึ้นต้นด้วยตัวเลขได้
3. ชื่อตัวแปรเป็น case sensitive กล่าวคือตัวอักษรพิมพ์ใหญ่และพิมพ์เล็กมีผลต่อการระบุชื่อตัวแปร (ตัวแปรชื่อ firstNumber และ FirstNumber ถือว่าไม่เป็นตัวแปรเดียวกัน)
4. ชื่อตัวแปรไม่สามารถเป็นคำสงวน (Reserved words) ได้

1.2.1.2 Reserved words

alignas	alignof	and
and_eq	asm	atomic_cancel
atomic_commit	atomic_noexcept	auto
bitand	bitor	bool
break	case	catch
char	char8_t	char16_t
char32_t	class	compl
concept	const	consteval
constexpr	constinit	const_cast
continue	co_await	co_return
co_yield	decltype	default
delete	do	double
dynamic_cast	else	enum
explicit	export	extern
false	float	for
friend	goto	if
inline	int	long
mutable	namespace	new
noexcept	not	not_eq
nullptr	operator	or
or_eq	private	protected
public	reflexpr	register
reinterpret_cast	requires	return
short	signed	sizeof
static	static_assert	static_cast
struct	switch	stnchronized
template	this	thread_local
throw	true	try
typedef	typeid	typename
union	unsigned	using
virtual	void	volatile
wchar_t	while	xor
xor_eq		

1.3 Input/Output

การรับค่าจากผู้ใช้ และการแสดงผลทางหน้าจอสามารถทำได้แตกต่างกันตามภาษา C หรือ C++

```c
int w;
double x;
char y;
char[10] z;

//C language
scanf("%d %lf %c %s", &w, &x, &y, z);
printf("%d %lf %c %s\n", w, x, y, z);
```

การรับค่าของภาษา C จะใช้คำสั่ง **scanf** โดยจะต้องระบุประเภทของตัวแปรที่ต้องการจะรับเข้ามาด้วย ในตัวอย่างจะเห็นได้ว่า

- ตัวระบุประเภทตัวที่ 1 คือ %d ซึ่งจะจับคู่กับตัวแปรชื่อ w
- ตัวระบุประเภทตัวที่ 2 คือ %lf ซึ่งจะจับคู่กับตัวแปรชื่อ x
- ตัวระบุประเภทตัวที่ 3 คือ %c ซึ่งจะจับคู่กับตัวแปรชื่อ y
- ตัวระบุประเภทตัวที่ 4 คือ %s ซึ่งจะจับคู่กับตัวแปรชื่อ z

และจะสังเกตได้ว่าการจับคู่ในการรับค่าจะต้องใส่เครื่องหมาย **&** หน้าตัวแปรทุกตัว นอกจากตัวแปรประเภท char[]

ประเภทตัวแปร	ตัวระบุประเภท
int	%d
double	%lf
char	%c
char[]	%s

การแสดงผลของภาษา C จะใช้คำสั่ง **printf** โดยจะมีลักษณะคล้ายกับการรับค่าด้วยคำสั่ง **scanf** กล่าวคือต้องใช้ตัวระบุประเภทเช่นเดียวกัน แต่มีความแตกต่างกันที่ ไม่ต้องใส่เครื่องหมาย **&** หน้าตัวแปรใดๆ เลย

หมายเหตุ: \n ที่อยู่ต่อท้ายเป็นการระบุว่า ให้ขึ้นบรรทัดใหม่เมื่อแสดงผลบรรทัดนี้แล้ว

```c
//C++ language
cin >> w >> x >> y >> z;
cout << w << ' ' << x << ' ' << y << ' ' << z << endl;
```

การรับค่าของภาษา C++ จะใช้คำสั่ง **cin** โดยมีเครื่องหมาย **<<** คั่นอยู่ระหว่างคำสั่ง และตัวแปรต่างๆ
การแสดงผลของภาษา C++ จะใช้คำสั่ง **cout** โดยมีเครื่องหมาย **>>** คั่นอยู่ระหว่างคำสั่ง และตัวแปรต่างๆ
นอกจากนี้ โปรแกรมภาษา C++ ก็สามารถใช้คำสั่ง **scanf** และ **printf** ของภาษา C ได้เช่นกัน

หมายเหตุ: endl ที่อยู่ต่อท้ายเป็นการระบุว่า ให้ขึ้นบรรทัดใหม่เมื่อแสดงผลบรรทัดนี้แล้ว (เช่นเดียวกันกับ \n)

1.3.1 Operators

1.3.1.1 Arithmetic

ตัวดำเนินการ	ชื่อ	คำอธิบาย	การใช้งาน
+	การบวก	บวก 2 จำนวนเข้าด้วยกัน	x + y
-	การลบ	ลบ 2 จำนวนเข้าด้วยกัน	x - y
*	การคูณ	คูณ 2 จำนวนเข้าด้วยกัน	x * y
/	การหาร	หารจำนวนแรกด้วยจำนวนที่ 2	x / y
%	การหารเอาเศษ	เศษจากการหารจำนวนแรกด้วยจำนวนที่ 2	x % y
++	Increment	เพิ่มค่าตัวแปรขึ้นไป 1	x++ หรือ ++x
--	Decrement	ลดค่าตัวแปรลงไป 1	x-- หรือ --x

1.3.1.2 Assignment

ตัวดำเนินการ	การใช้งาน	ผลลัพธ์
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5

1.3.1.3 Comparison

ตัวดำเนินการ	ชื่อ	การใช้งาน
==	เท่ากับ	x == y
!=	ไม่เท่ากับ	x != y
>	มากกว่า	x > y
<	น้อยกว่า	x < y
>=	มากกว่าหรือเท่ากับ	x >= y
<=	น้อยกว่าหรือเท่ากับ	x <= y

1.3.1.4 Logical

ตัวดำเนินการ	ชื่อ	การใช้งาน
&&	และ (ตรรกศาสตร์)	`x > 1 && x < 10`
||	หรือ (ตรรกศาสตร์)	`x < 1 || x > 10`
!	นิเสธ	`!(x > 1 && x < 10)`

1.4 Conditions

เราสามารถเลือกให้โปรแกรมทำบางคำสั่ง เมื่อเงื่อนไขเป็นจริงเท่านั้น โดยการใช้ **if-else** หรือ **switch** ได้

```
if (condition1) {
   // code to be executed if the condition1 is true
}else if (condition2) {
   // code to be executed if the condition1 is false and
    condition2 is true
} else if (condition3) {
   // code to be executed if the condition1,2 is false and
    condition 3 is true
} else {
   // code to be executed if the condition1,2,3 is false
}
```

การเขียนเงื่อนไขโดยใช้ **if**, **else if**, และ **else** จะเป็นที่นิยมเนื่องจากใช้งานได้ง่ายและสะดวก

```
switch (expression) {
   case x:
        // code to be executed if expression is equal to x
          break;
     case y:
        // code to be executed if expression is equal to y
          break;
     default:
        // code to be executed if expression is not equal to any
    case
}
```

การเขียนเงื่อนไขโดยการใช้ **switch** จะต้องระบุ `expression`ให้ตรงกับ `case` พอดี ไม่สามารถเปรียบเทียบน้อยกว่า หรือมากกว่าได้

หมายเหตุ: ทุก `case` ของ `switch` จะต้องมีคำสั่ง **break** ต่อท้ายอยู่เสมอ

1.5 Loop

การวนซ้ำสามารถใช้ในการทำงานซ้ำหลายๆ ครั้งตราบใดที่เงื่อนไขของการวนซ้ำยังเป็นจริงอยู่ ประโยชน์ของการวนซ้ำคือลดระยะเวลาการเขียนลง ลดการเกิด errors และทำให้โปรแกรมอ่านง่ายขึ้น

การวนซ้ำทุกรูปแบบจะมี 3 สิ่งสำคัญได้แก่

1. จุดเริ่มต้น (initial)
2. เงื่อนไข (condition)
3. การเปลี่ยนแปลง (update)

```
for (initial; condition; update) {
  //block of code to be executed
}

for (int i = 0; i < 5; i++) {
  printf("%d\n", i);
}

initial
while (condition) {
  //block of code to be executed
    update
}

int i = 0;
while (i < 5) {
  printf("%d\n", i);
    i++;
}
```

ตัวอย่างโปรแกรมด้านบนเป็นโปรแกรมเพื่อแสดงผลตัวเลข 0 ถึง 4 บรรทัดละ 1 จำนวน โดยการวนซ้ำด้วย **for** และ **while** เริ่มต้นที่ **i = 0** โดยมีเงื่อนไขการวนซ้ำคือ **i < 5** และมีส่วนการเปลี่ยนแปลงคือ **i++**

- ขณะที่ **i = 0** ซึ่ง **i < 5** จึงแสดงผลเลข **0** ออกมา
- ขณะที่ **i = 1** ซึ่ง **i < 5** จึงแสดงผลเลข **1** ออกมา
- ขณะที่ **i = 2** ซึ่ง **i < 5** จึงแสดงผลเลข **2** ออกมา
- ขณะที่ **i = 3** ซึ่ง **i < 5** จึงแสดงผลเลข **3** ออกมา
- ขณะที่ **i = 4** ซึ่ง **i < 5** จึงแสดงผลเลข **4** ออกมา
- ขณะที่ **i = 5** ซึ่ง **i !< 5** จึงหยุดการวนซ้ำ

นอกจากนี้ยังมีการวนซ้ำแบบ **do-while** ซึ่งจะทำงานก่อน 1 รอบไม่ว่าเงื่อนไขที่กำหนดจะเป็นจริงหรือไม่ แล้วจึงตรวจสอบเงื่อนไขก่อนทำงานรอบถัดไป

```c
int i = 5;
do {
   printf("%d\n", i);
}while (i < 5);
```

โปรแกรมด้านบนนี้จะแสดงผลเลข 5 ออกมาแล้วจึงหยุดการทำงาน

ในบางกรณี เราจำเป็นต้องหยุดการวนซ้ำกลางคัน หรือข้ามการวนซ้ำบางขั้นตอน จึงมีคำสั่ง **break** สำหรับหยุดการวนซ้ำนั้น และ **continue** สำหรับข้ามการวนซ้ำขั้นตอนหนึ่งๆ

```c
for (int i = 0; i < 5; i++) {
   if (i == 2) {
        break;
      }
   printf("%d\n",i);
}
```

โปรแกรมด้านบนนี้จะแสดงผลแค่เลข 0 และเลข 1 บรรทัดละ 1 จำนวน

- ขณะที่ **i = 0** ซึ่ง **i < 5** จึงแสดงผลเลข **0** ออกมา
- ขณะที่ **i = 1** ซึ่ง **i < 5** จึงแสดงผลเลข **1** ออกมา
- ขณะที่ **i = 2** ซึ่ง **i < 5** แต่ตรงกับเงื่อนไขของคำสั่ง **break** จึงหยุดการวนซ้ำทันที

```c
for (int i = 0; i < 5; i++) {
   if (i == 2) {
        continue;
      }
   printf("%d\n",i);
}
```

โปรแกรมด้านบนนี้จะแสดงผลแค่เลข 0, 1, 3, และ 4 บรรทัดละ 1 จำนวน

- ขณะที่ **i = 0** ซึ่ง **i < 5** จึงแสดงผลเลข **0** ออกมา
- ขณะที่ **i = 1** ซึ่ง **i < 5** จึงแสดงผลเลข **1** ออกมา
- ขณะที่ **i = 2** ซึ่ง **i < 5** แต่ตรงกับเงื่อนไขของคำสั่ง **continue** จึงข้ามการวนซ้ำขั้นตอนนี้ไป และไม่มีการแสดงผลเลข **2** ออกมา
- ขณะที่ **i = 3** ซึ่ง **i < 5** จึงแสดงผลเลข **3** ออกมา
- ขณะที่ **i = 4** ซึ่ง **i < 5** จึงแสดงผลเลข **4** ออกมา
- ขณะที่ **i = 5** ซึ่ง **i !< 5** จึงหยุดการวนซ้ำ