ทบทวนภาษา C

เทพพิทักษ์ การุญบุญญานันท์

main()

- เป็น entry point ของโปรแกรมเมื่อเริ่มทำงาน
- จบฟังก์ชัน `main()` $\rightarrow$ จบโปรแกรม

```c
#include <stdio.h>

int main()
{
  printf ("Hello, world.\n");
  return 0;
}
```

ตัวแปรและชนิด

- ชนิดจำนวนเต็ม:

[unsigned] ⟨*ชนิด*⟩

โดย ⟨*ชนิด*⟩ ได้แก่:

- **char**
- **short** [**int**]
- **int**
- **long** [**int**]
- **long long** [**int**]

- ชนิดทศนิยมจุดลอยตัว (floating point):
  - **float**
  - **double**
  - **long double**

ค่าคงที่

- ค่าคงที่จำนวนเต็ม:
  - ฐานสิบ: `97`
  - ฐานแปด: `0141` (= 97)
  - ฐานสิบหก: `0x61` (= 97)
- suffix บอกชนิด:
  - `unsigned`: `97U`
  - `long`: `97L`
  - `long long`: `97LL`

- ค่าคงที่อักขระ:
  - อักขระ ASCII: `'a'` (= 97)
  - รหัสฐานแปด: `'\141'` (= 97)
  - รหัสฐานสิบหก: `'\x61'` (= 97)
  - อักขระพิเศษ:

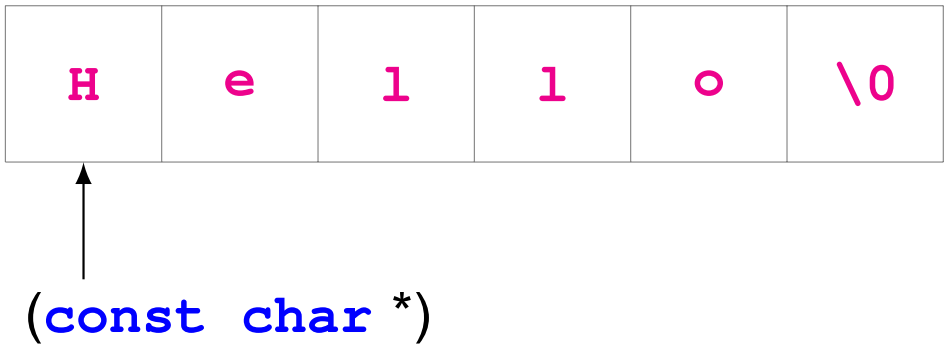
    `'\n'`, `'\t'`, `'\v'`,
    `'\b'`, `'\f'`, `'\a'`,
    `'\''`, `'\"'`, `'\\'`

ค่าคงที่

- ค่าคงที่จุดทศนิยมลอยตัว:
  - `3.14159`
  - `2.99792458e+8` ($= 2.99792458 \times 10^8$)
- suffix บอกชนิด:
  - `float`: `3.14159F`, `2.99792458e+8F`
  - `double`: `3.14159L`, `2.99792458e+8L`

ค่าคงที่

- ค่าคงที่สายอักขระ (string):
  - `"Hello"`



  - การ escape:

```
"\"\\\x61\t\142\""
```

"	\	a	\t	b	"	\0

Operators

- ตัวชี้และที่อยู่: unary `*` `&`
- การเข้าถึงสมาชิก: `.` `->` `[ ]`
- assignment: `=`
- เลขคณิต: `+` `-` `*` `/` `%` และ unary `+` `-`
- การเพิ่ม/ลดค่า: `++` `--`
- การกระทำบิต: `&` `|` `^` `<<` `>>` และ unary `~`
- เปรียบเทียบ: `<` `<=` `==` `!=` `>=` `>`
- การกระทำตรรกะ: `&&` `||` และ unary `!`
- เงื่อนไข: ⟨*เงื่อนไข*⟩ `?` ⟨*ค่าเมื่อจริง*⟩ `:` ⟨*ค่าเมื่อเท็จ*⟩
- ลำดับการกระทำ: `,`
- จัดกลุ่ม: `(` `)`

Statements

- Statement ซับซ้อน

```
{
  int a; /* declarations */

  /* series of statements */
  a = 0;
  a++;
}
```

บล็อคในวงเล็บปีกกาถือเสมือนเป็น statement เดียวในไวยากรณ์

Statements

- **if** statement

```
if ( /* condition */ )
   /* statement */
```

หรือ:

```
if ( /* condition */ )
   /* statement */
else
   /* statement */
```

Statements

- **switch-case** statement

```
switch ( /* value */ )
  {
    case VALUE1:
      /* statements */

    case VALUE2:
      /* statements */

    default:
      /* statements */
  }
```

Statements

- **while** loop

```
while ( /* condition */ )
   /* statement */
```

- **do-while** loop

```
do
   /* statement */
while ( /* condition */ );
```

- **for** loop

```
for ( /* init */ ; /* condition */ ; /* increment */ )
  /* statement */
```

เทียบเท่ากับ:

```
/* init */
while ( /* condition */ )
  {
    /* statement */
    /* increment */
  }
```

Statements

- `goto` statement

```
  goto label;
  ...
label:
  /* goto jumps here */
  ...
```

Statements

- **break** statement
  - กระโดดออกจากลูปหรือ **switch-case** ปัจจุบัน

```
for (i = 0; i < 10; i++)
{
   ...
   if (t > 2)
     break;
   ...
}
/* break jumps here */
```

- **continue** statement
  - กระโดดไปท้ายลูปปัจจุบันเพื่อเริ่มรอบใหม่ (ถ้าเป็นลูป **for** ก็ไปที่ส่วน increment)

```
for (i = 0; i < 10; i++)
{
   ...
   if (t > 2)
     continue;
   ...
   /* continue jumps here */
}
```

Function

- function prototype
  - ประกาศชื่อให้โค้ดผู้เรียกรู้จัก

⟨*ชนิดรีเทิร์น*⟩ ⟨*ชื่อฟังก์ชัน*⟩ **(** ⟨*ชนิด*⟩ ⟨arg_1⟩ **,** ... **);**

ตัวอย่างเช่น:

```
int sum (int a, int b);

int f()
{
   ...
   x = sum (y, 4);
   ...
}
```

Function

- function definition
  - implementation ของฟังก์ชัน
    ตัวอย่างเช่น:

```
int sum (int a, int b)
{
  return a + b;
}
```

Pointer

- ข้อมูลที่เป็นที่อยู่ของข้อมูลอื่น (ชี้ไปยังข้อมูลอื่น)
- การประกาศ:

```
int   *p;    /* pointer to int */
char **q;    /* pointer to a pointer to char */
```

- การให้ค่าและการ dereference:

```
int  a, b;
int  *p = &a;  /* p points to a */
int **q = &p;  /* q points to p */
*p = 1;  /* now a == 1 */
**q = 2; /* now a == 2 */
*q = &b; /* now p points to b */
*p = 3;  /* now b == 3 */
```

Array

- แถวของข้อมูลชนิดเดียวกันเรียงติดกัน
- เข้าถึงข้อมูลแต่ละช่องได้ด้วย index
- การประกาศ:

```
int  p[8];     /* array of 8 int's */
char q[2][3]; /* 2-d array of 2 rows, 3 columns */
```

- การเข้าถึง:

```
p[0] = 2;
x = p[1];
q[1][2] = 'a';
```

Array & Pointer

- ชื่อของแอร์เรย์มีชนิดเป็นพอยน์เตอร์

```
int  p[8];     /* array of 8 int's */
int *q = p;    /* q now points to p[0] */
```

```
int f (int *v);
int  p[8];       /* array of 8 int's */
int  c = f (p); /* pass &p[0] as int pointer */
```

- พอยน์เตอร์สามารถใช้ index แบบแอร์เรย์ได้

```
int  p[8];     /* array of 8 int's */
int *q = p;    /* q now points to p[0] */
q[1] = 7;      /* access p[1] via q */
```

- การบวก/ลบพอยน์เตอร์ด้วยจำนวนเต็มจะคำนวณเหมือนทำ index แอร์เรย์

```
int  p[8];          /* array of 8 int's */
int *q = p + 1;     /* q now points to p[1] */
*q = 7;             /* access p[1] via q */
*(q - 1) = 6;       /* access p[0] via q */
```

- การลบพอยน์เตอร์กับพอยน์เตอร์จะได้ระยะห่างเป็นจำนวนช่องแอร์เรย์

```
int  p[8];          /* array of 8 int's */
int *q = p + 1;     /* q now points to p[1] */
int  d = q - p;     /* d == 1, regardless of int size */
```

Complex Declaration

- หลักการประกาศของภาษา C: ประกาศชนิดของนิพจน์

ตัวอย่าง:

```
int  a;
```

→ **a** มีชนิดเป็น **int**

```
int *b;
```

→ ***b** มีชนิดเป็น **int**

→ **b** เป็นพอยน์เตอร์ไปยัง **int**

Complex Declaration

```
int *c[8];
```

→ *c[0] มีชนิดเป็น int

→ c[0] เป็นพอยน์เตอร์ไปยัง int

→ c เป็นแอร์เรย์ของพอยน์เตอร์ไปยัง int

Complex Declaration

```
int *(*d[8])(int, int);
```

→ `*(*d[0])(int, int)` มีชนิดเป็น `int`

→ `(*d[0])(int, int)` เป็นพอยน์เตอร์ไปยัง `int`

→ `*d[0]` เป็นฟังก์ชันที่รีเทิร์นพอยน์เตอร์ไปยัง `int`

→ `d[0]` เป็นพอยน์เตอร์ไปยังฟังก์ชันที่รีเทิร์นพอยน์เตอร์ไปยัง `int`

→ `d` เป็นแอร์เรย์ของพอยน์เตอร์ไปยังฟังก์ชันที่รีเทิร์นพอยน์เตอร์ไปยัง `int`

Type Conversion

- ภาษา C จะมีการแปลงค่าตามสมควรเมื่อกำหนดค่าข้ามชนิด

```
int    length = 123456;
long   distance = length;   /* high bytes are zero */
float  fdist = length;      /* precision kept */
fdist = 123456.78;
length = fdist;             /* take only integral part */
```

- ในกรณีที่ปลายทางมีความจุน้อยกว่าก็จะถูกตัดทอน

```
long   distance = 12345678912345678 9;
int    length = distance;   /* high bytes are truncated */
float  fdist = distance;    /* precision lost */
length = fdist;             /* overflow */
```

Type Casting

- ใช้ในการบังคับเปลี่ยนชนิดซึ่งปกติจะผิดข้อกำหนดภาษา C

```c
int f (char *p);
int v[8];
char *p = v;    /* Error: incompatible pointer type */
char *q = (char *)v;    /* OK */
int x = f (v); /* Error: incompatible pointer type */
int y = f ((char *)v); /* OK */
```

- ควรใช้อย่างระมัดระวัง!

Enumeration

- เป็นการกำหนดสัญลักษณ์ให้กับข้อมูลที่มีลำดับแทนการใช้ตัวเลข
- ตัวอย่างการประกาศชนิด:

```
enum DOW {
  SUN, MON, TUE, WED, THU, FRI, SAT
};
```

- ตัวอย่างการใช้งาน:

```
enum DOW dow = SUN;
...
if (dow == SUN) {
  printf ("It's holiday!\n");
}
```

Structure

- ใช้กำหนดข้อมูลที่มีสมาชิกต่างชนิดกัน
- การประกาศโครงสร้าง:

```
struct student {
  int  id;
  char name[128];
  int  score;
};
```

- การประกาศตัวแปรและใช้งาน:

```
struct student rec;
struct student *p = &rec;

rec.id = 1;
strcpy (rec.name, "Julius Caesar");
rec.score = 10;
printf ("%d, %s, %d\n", rec.id, rec.name, rec.score);

p->id = 2;
strcpy (p->name, "Mark Anthony");
p->score = 6;
printf ("%d, %s, %d\n", p->id, p->name, p->score);
```

Bit Field

- เป็นวิธีบีบข้อมูลที่มีขนาดเล็กกว่าไบต์เรียงลงในไบต์หรือเวิร์ดเดียวกัน
- เมื่อระบุจำนวนบิตให้กับสมาชิกของ **struct**
  - สมาชิกจะถูกจองเท่าจำนวนบิตที่กำหนดเท่านั้น
  - สมาชิกต่อไปจะถูกจองต่อจากเดิมเป็นจำนวนบิตที่ระบุ
  - เมื่อพบสมาชิกชนิดใหม่ หรือไม่ระบุบิต ก็เลิกจองแบบบิตต่อ
- ตัวอย่างการประกาศชนิด:

```
struct box {
  unsigned char has_border   : 1;
  unsigned char border_color : 3; /* RGB */
  unsigned char is_filled    : 1;
  unsigned char fill_color   : 3; /* RGB */
  int x1, y1, x2, y2;
};
```

Union

- เก็บข้อมูลหลายชนิดไว้ในที่อยู่เดียวกัน
- การประกาศและใช้งานเหมือน **struct**
- โดยปกติจะมีแท็กบ่งชี้ชนิดของข้อมูลปัจจุบันอยู่ด้วย
- ตัวอย่างการประกาศชนิด:

```
struct value {
  enum { INT, STRING, REAL } type;
  union {
    int    v_int;
    char   v_str[10];
    float  v_real;
  } val;
};
```

Union

- ตัวอย่างการใช้งาน:

```
void print (const struct value *v)
{
  switch (v->type) {
    case INT:    printf ("%d\n", v->val.v_int);  break;
    case STRING: printf ("%s\n", v->val.v_str);  break;
    case REAL:   printf ("%f\n", v->val.v_real); break;
    default:     printf ("Unknown type!\n");     break;
  }
}
```

Type Definition

- เป็นการตั้งชนิดใหม่ขึ้นมาโดยอาศัยการประกาศ
- ตัวอย่างการประกาศ:

```
typedef struct {
   int  id;
   char name[128];
   int  score;
} Student;
```

- เมื่อจะประกาศตัวแปรก็ใช้ชื่อชนิดได้ทันที:

```
Student rec;
```

Preprocessing

- เป็นการกระทำที่อยู่นอกเหนือไวยากรณ์ภาษา C
- กระทำโดย preprocessor ก่อนส่งให้คอมไพเลอร์
- การอ่านเนื้อหาแฟ้มอื่นเข้ามารวม:

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

  - วงเล็บแหลม: แฟ้มของระบบ
  - ในเครื่องหมายคำพูด: แฟ้มในไดเรกทอรีเดียวกัน

Preprocessing

- การกำหนดแมโครค่าคงที่:

```
#define MAX_HEIGHT  10
```

  - แทนที่ **MAX_HEIGHT** ทั้งหมดที่ปรากฏในแฟ้มด้วย **10**
  - หากกำหนดค่าคงที่เป็นชุด แนะนำให้ใช้ **enum**

- การกำหนดแมโครที่มีพารามิเตอร์:

```
#define MAX(a,b)  ((a)>(b)?(a):(b))
```

  - แทนที่ **MAX(x,y)** เป็น **((x)>(y)?(x):(y))**
  - ใช้งานคล้ายฟังก์ชัน
  - ควรครอบวงเล็บในค่าที่กระจายไว้ก่อน เผื่อกรณีซับซ้อน เช่น

    MAX(t?c:a,b)

- ตรวจสอบการกำหนดแมโคร:

```
#ifndef MAX_HEIGHT
#define MAX_HEIGHT 10
#endif
```

```
#ifdef __cplusplus
... /* C++ stuffs */
#endif
```

Preprocessing

- เงื่อนไขทั่วไป

```
#if INT_MAX == 0x7fff
... /* int is 16 bits */
#elif INT_MAX == 0x7fffffff
... /* int is 32 bits */
#endif
```

```
#if defined __GNUC__ && __GNUC__ > 2
... /* Using GCC > 2 */
#endif
```