計算機程式設計
C語言 Program in Large

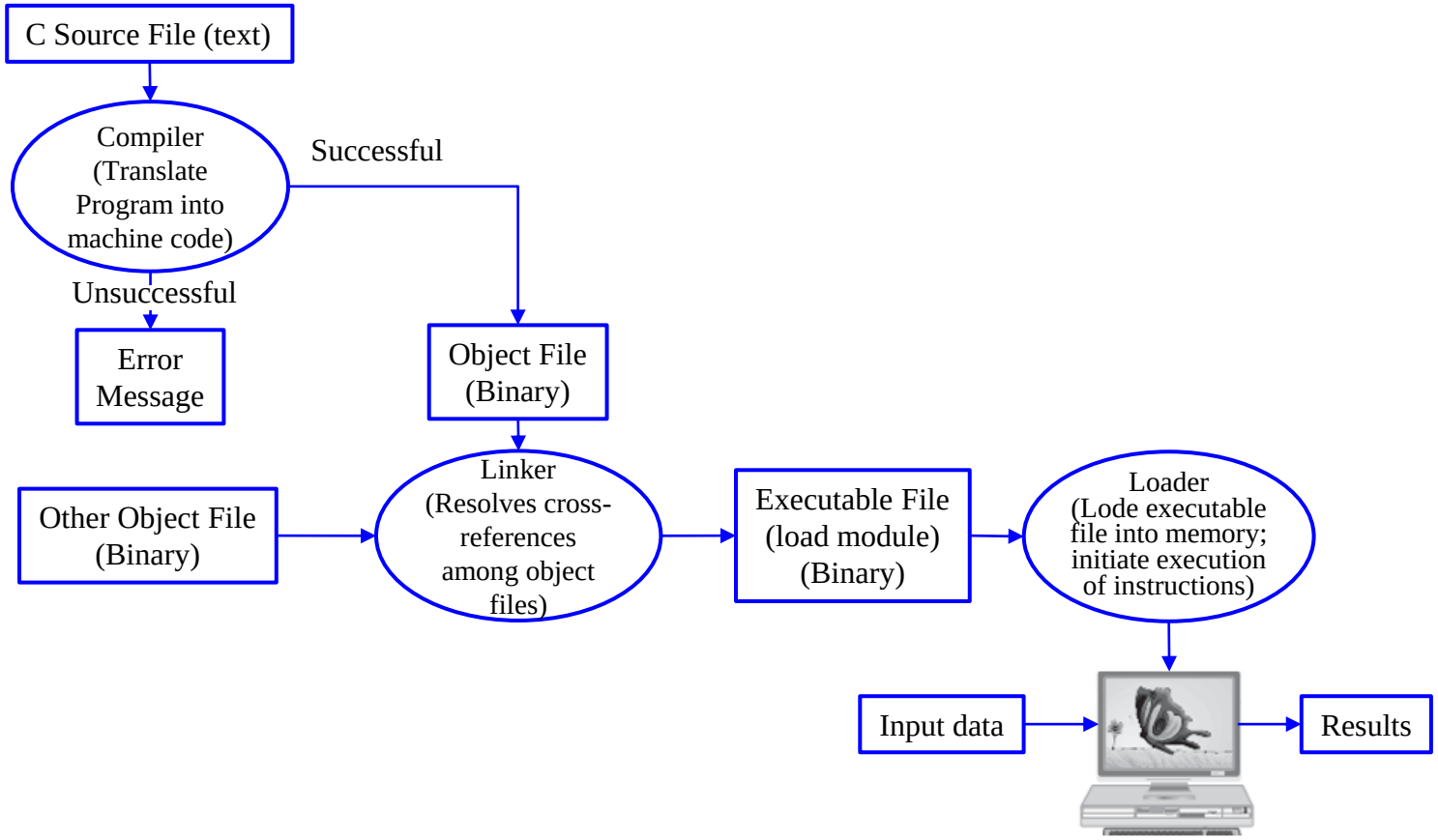
郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

程式執行

- 程式撰寫、編譯、連結



自訂函式庫

- extern
  - 宣告變數、函式在其他地方定義，"其他地方"可能是同一個檔案，或其他檔案。
- 步驟
  - (Code:Block)造一個project檔案，加入 main.c 和 some.c
  - 編譯執行

```
// some.c
double some_var = 1000.0;
```

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var;
    printf("%f\n", some_var);
    return 0;
}
```

自訂函式庫

- extern
  - 在使用 extern 同時指定值，則視為在該位置宣告定義變數，將造成重覆定義錯誤。
  - 須先宣告 extern 找到變數，再重新指定其值。

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var = 100;  // 錯誤
    printf("%f\n", some_var);
    return 0;
}
```

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var;
    some_var  = 100;  // 正確
    printf("%f\n", some_var);
    return 0;
}
```

自訂函式庫

- 標頭檔案 Header
  - 註解區塊，說明函式庫的功能目的
  - #define 宣告、命名、常數和巨集
  - 資料型別定義 typedef
  - 針對函式庫的每一個函式，有一個註解區塊，說明該函式的功能，以及使用外部宣告(extern)定義的函式

自訂函式庫

- 標頭檔案 Header

```
// planet.h - abstract data type planet
// Type planet_t has these components:
// name, diameter, moons, orbit_time, rotation_time
// Operators:
// print_planet, planet_equal, scan_planet
#define PLANET_STRSIZ 10
typedef struct {          // planet structure
    char name[PLANET_STRSIZ];
    double diameter;    // equatorial diameter in km
    int moons;              // number of moons
    double orbit_time, // years to orbit sun once
    rotation_time;       // hours to complete one revolution on axis
} planet_t;
// Displays with labels all components of a planet_t structure
extern void print_planet(planet_t pl); // input - one planet structure
```

自訂函式庫

- 標頭檔案 Header

```
// Determines whether or not the components of planet_1 and planet_2 match
// input - planets to compare

extern int planet_equal(planet_t planet_1, planet_t planet_2);
// Fills a type planet_t structure with input data. Integer returned as
// function result is success/failure/EOF indicator.
// 1 => successful input of planet
// 0 => error encountered
// EOF => insufficient data before end of file
// In case of error or EOF, value of type planet_t output argument is  undefined.
extern int
scan_planet(planet_t *plnp); /* output - address of planet_t structure to fill */
```

自訂函式庫

- 實做 Implementation

```
// planet.c
#include <stdio.h>
#include <string.h>
#include "planet.h"
// Displays with labels all components of a planet_t structure
void print_planet(planet_t pl) {              // input - one planet structure
    printf("%s\n", pl.name);
    printf(" Equatorial diameter: %.0f km\n", pl.diameter);
    printf(" Number of moons: %d\n", pl.moons);
    printf(" Time to complete one orbit of the sun: %.2f years\n", pl.orbit_time);
    printf(" Time to complete one rotation on axis: %.4f hours\n", pl.rotation_time);
}
```

自訂函式庫

- 實做 Implementation

```
// Determines whether or not the components of planet_1 and planet_2 match
// input - planets to  compare
int planet_equal(planet_t planet_1, planet_t planet_2) {
    return (strcmp(planet_1.name, planet_2.name) == 0 &&
    planet_1.diameter == planet_2.diameter &&
    planet_1.moons == planet_2.moons && planet_1.orbit_time ==
    planet_2.orbit_time &&  planet_1.rotation_time == planet_2.rotation_time);
}
```

自訂函式庫

- 實做 Implementation

```
// Fills a type planet_t structure with input data. Integer returned as
// function result is success/failure/EOF indicator.
// 1 => successful input of planet
// 0 => error encountered
// EOF => insufficient data before end of file
// In case of error or EOF, value of type planet_t output argument is undefined.
// output - address of planet_t structure tofill
int scan_planet(planet_t *plnp)  {
    int result;
    result = scanf("%s%lf%d%lf%lf", plnp->name, &plnp->diameter,
                        &plnp->moons, &plnp->orbit_time, &plnp->rotation_time);
    if (result == 5)
        result = 1;
    else if (result != EOF)
        result = 0;
   return (result);
}
```

使用自訂函式庫

- 標頭檔案 Header

```
// Beginning of source file in which a personal library and system I/O library
// are used.

#include <stdio.h> // system's standard I/O functions
#include "planet.h" // personal library with planet_t data type and operators
int main() {
    return 0;
}
```

變數等級(Storage Class)

- 全域變數 (global variable)
  - 宣告在所有函式外面。
  - 生命週期，編譯完就配置空間，直到程式結束。
  - 避免使用全域變數，因每個函式都可變更其值
  - 全域常數，可以使用。

```
// eg1.c
int global_var_x;
const size = 10;
const char *months[12] = {"January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December"};
void f() {     global_var_x = 10; }
int main() {
    global_var_x = 20;
    f();
    return 0;
}
```

變數等級(Storage Class)

- 外在全域變數 (global variable)
  - extern 全域變數，宣告在其他地方，故在此不配置記憶體空間。

```
// eg2.c
extern int global_var_x;
void g() {
    global_var_x = 5;
}
int main() {
    g();
    return 0;
}
```

變數等級(Storage Class)

- 區域變數 auto
  - many，一般區域變數，auto等級，存放於堆疊stack。
  - 生命週期，宣告開始，直到第一個Block結束(遇到右大括號)。
  - 每次呼叫f函式，many會重新配置記憶體空間，被重新初始化為 0。

```
void f() {
    int many = 0;
    many = many + 1;
    if (many>=1) {
        int many = 0;
        printf("%d\n", many); //0
    }
    printf("%d\n", many);  // 1
}
int main() {
    f();
    return 0;
}
```

變數等級(Storage Class)

- 靜態變數 static
  - once，static，編譯後程式執行之前就(只)會配置一次記憶體空間，只初始化其值一次，每次f函式變更其值，都會被記住。
  - 生命週期，程式開始執行，直到程式結束。

```
int f () {
    static int once = 0;
    once = once +1;
    return once;
}
int main() {
    printf("%d\n", f());  //1
    printf("%d\n", f());  //2
    printf("%d\n", f());  //3
    return 0;
}
```

變數等級(Storage Class)

- 暫存器變數 register
  - 使用CPU的暫存器，存取速度快

```
static double matrix[50][40];
register int row, col;
```

條件式編譯

- 追蹤 printf偵錯，#if - #endif
  - 偵錯模式，#define TRACE

```
// Computes an integer quotient (m/n) using subtraction
// #define TRACE
int quotient(int m, int n){
    int ans;
#if defined (TRACE)
    printf("Entering quotient with m = %d, n = %d\n", m, n);
#endif
    if (n > m) ans = 0;
    else  ans = 1 + quotient(m - n, n);
#if defined (TRACE)
     printf("Leaving quotient(%d, %d) with result = %d\n", m, n, ans);
#endif
    return (ans);
}
```

條件式編譯

- 追蹤 printf偵錯，#if - #elif - #endif

```
// #define TRACE_VERBOSE  #define TRACE_BRIEF
int  quotient(int m, int n) {
    int ans;
#if defined (TRACE_VERBOSE)
    printf("Entering quotient with m = %d, n = %d\n", m, n);
#elif defined (TRACE_BRIEF)
    printf(" => quotient(%d, %d)\n", m, n);
#endif
    if (n > m) ans = 0;
    else ans = 1 + quotient(m - n, n);
#if defined (TRACE_VERBOSE)
    printf("Leaving quotient(%d, %d) with result = %d\n", m, n, ans);
#elif defined (TRACE_BRIEF)
    printf("quotient(%d, %d) => %d\n", m, n, ans);
#endif
 return (ans);
 }}
```

自訂函式庫

- 標頭檔案 Header ，防止重複被包含

```
// planet.h - abstract data type planet
// Type planet_t has these components:
// name, diameter, moons, orbit_time, rotation_time
// Operators:
// print_planet, planet_equal, scan_planet
#if !defined (PLANET_H_INCL)
#define PLANET_H_INCL
#define PLANET_STRSIZ 10
typedef struct {          // planet structure
    char name[PLANET_STRSIZ];
    double diameter;    // equatorial diameter in km
    int moons;              // number of moons
    double orbit_time, // years to orbit sun once
    rotation_time;       // hours to complete one revolution on axis
} planet_t;
// Displays with labels all components of a planet_t structure
extern void print_planet(planet_t pl); // input - one planet structure
```

自訂函式庫

- 標頭檔案 Header ，防止重複被包含

```
// Determines whether or not the components of planet_1 and planet_2 match
// input - planets to compare

extern int planet_equal(planet_t planet_1, planet_t planet_2);
// Fills a type planet_t structure with input data. Integer returned as
// function result is success/failure/EOF indicator.
// 1 => successful input of planet
// 0 => error encountered
// EOF => insufficient data before end of file
// In case of error or EOF, value of type planet_t output argument is  undefined.

extern int
scan_planet(planet_t *plnp); // output - address of planet_t structure to fill

#endif
```

argc & argv

- 命令列引數，只有main可以使用；是字串型別
  - argc:引數的個數(整數)，argv: 指向字元指標陣列的指標
  - 命令列引數必須由空格分開
  - argv[0]表示程式名稱，argv[1]表示第一個引數

```
// Test.c       Test.c Tom John
#include <stdio.h>
int main(int argc, char * argv[]) {
    int t,i;
    if (argc<2) {
        printf("you forgot to type your name\n");
        exit(0);
    }
    for (t=0; t<argc; t++) {
        printf("Hi~ %s\n", argv[t]);
    }
    return 0;
}
```

```
Hi~ Test.c
Hi~ Tom
Hi~ John
```

巨集(macro)

- 巨集處理
  - 前置處理器，將程式中巨集定義的符號，換成巨集定義後面的符號

```
#include <stdio.h>
#define LABEL_PRINT_INT(label, num) printf("%s = %d", (label), (num))
int main() {
   int r = 5, t = 12;
   LABEL_PRINT_INT("rabbit", r);
   //  變成 printf("%s=%d", ("rabbit"), (r));
   printf(" ");
   LABEL_PRINT_INT("tiger", t + 2);  // t+22; 當成 num，換成 (t+2)
   //  變成 printf("%s=%d", ("tiger"), (t+2));
   printf("\n");
   return(0);
}

// rabbit = 5 tiger = 14
```

巨集(macro)

- 巨集與函式
  - 巨集純粹使用符號取代，不是函式的呼叫
  - 巨集會使程式碼變大，函式會使用堆疊stack，呼叫執行讓程式較慢
- 巨集換行

```
#include <stdio.h>
#define INDEXED_FOR(ct, st, end) \
for ((ct) = (st); (ct) < (end); ++(ct))

int main() {
    int i = 0;
    INDEXED_FOR(i, 0, X_MAX)
        printf("x[%2d] = %6.2f\n", i, x[i]);
    return(0);
}
```

```
for ((i) = (0); (i) < (X_MAX); ++(i))
    printf("x[%2d] = %6.2f\n", i, x[i]);
```

巨集(macro)

- 巨集處理要加括號

```
#define SQUARE(n) n * n     // 應該要寫成 #define SQUARE(n) ((n) * (n))
int main() {
    double x = 0.5, y = 2.0;
    int n = 4, m = 12;
    printf("(%.2f + %.2f) squared = %.2f\n\n", x, y, SQUARE(x + y));
    printf("%d squared divided by ", m);
    printf("%d squared is %d\n", n, SQUARE(m) / SQUARE(n));
    return 0;
}

// SQUARE(x + y)                變成  x + y * x + y
// SQUARE(m) / SQUARE(n))變成 m * m / n * n
//         印出 (0.5 + 2.0)squared = 3.5, 12 squared divided by 4 squared is 144
// 應該印出 (0.5 + 2.0)squared = 6.25, 12 squared divided by 4 squared is 9
```

可變參數巨集

- gcc 前置處理器

```
#ifdef DEBUG
#define dbgprint(format, args...) \
  fprintf(stderr, format, ##args)
#else
#define dbgprint(format, args...)
#endif
int main() {
    dbgprint("%s", __FILE__);
    return 0;
}
```

- code:block

```
#include <stdio.h>
#define debug(...) printf(__VA_ARGS__)
int main(){
    char name[][5]={"Tom", "John"};
    debug("Hello\n");
    debug("Hello %s\n", name[0]);
    debug("Hello %s, %s\n", name[0], name[1]);
    return 0;
}
```

```
Hello
Hello Tom
Hello Tom, John
```