CSCA48

Introduction to Computer Science II

导师： VC

UTSC Week 13 Final Review (Unit 6)

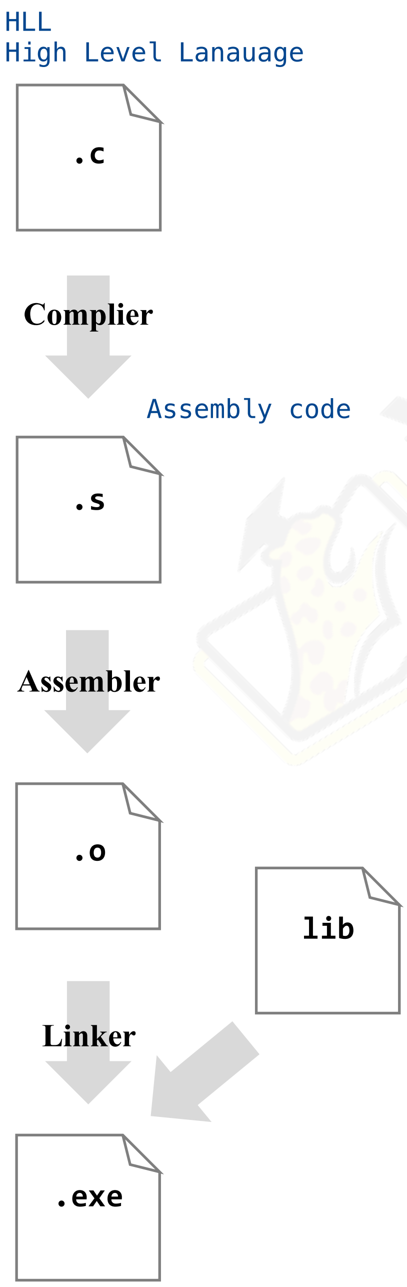
S P E E D   U P   E D U C A T I O N

How gcc works?

Consider the following program, `csca48.c`, what actually happened when we run `gcc csca48.c`

```
#include <stdio.h>
int main () {
    int a = 0;
    int b = a + 5;
    printf("%d\n", b);
}
```

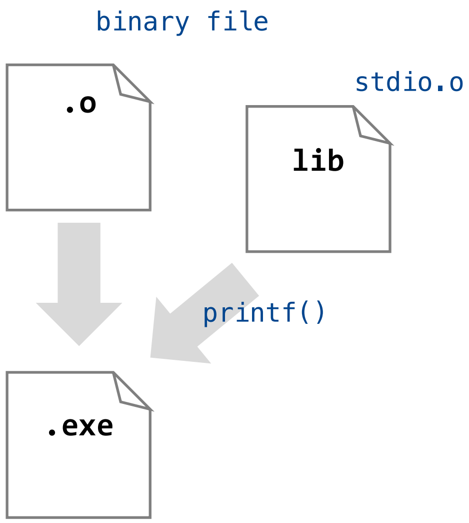
csca48.*c*

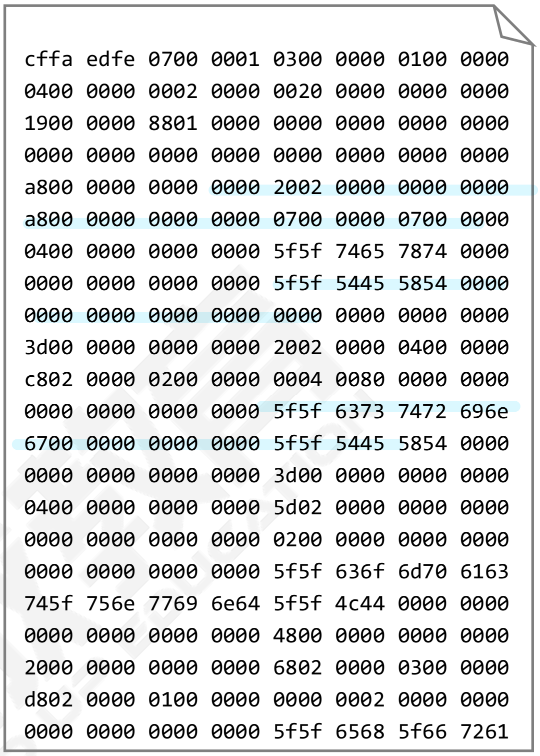


csca48.*s*   CSCB58

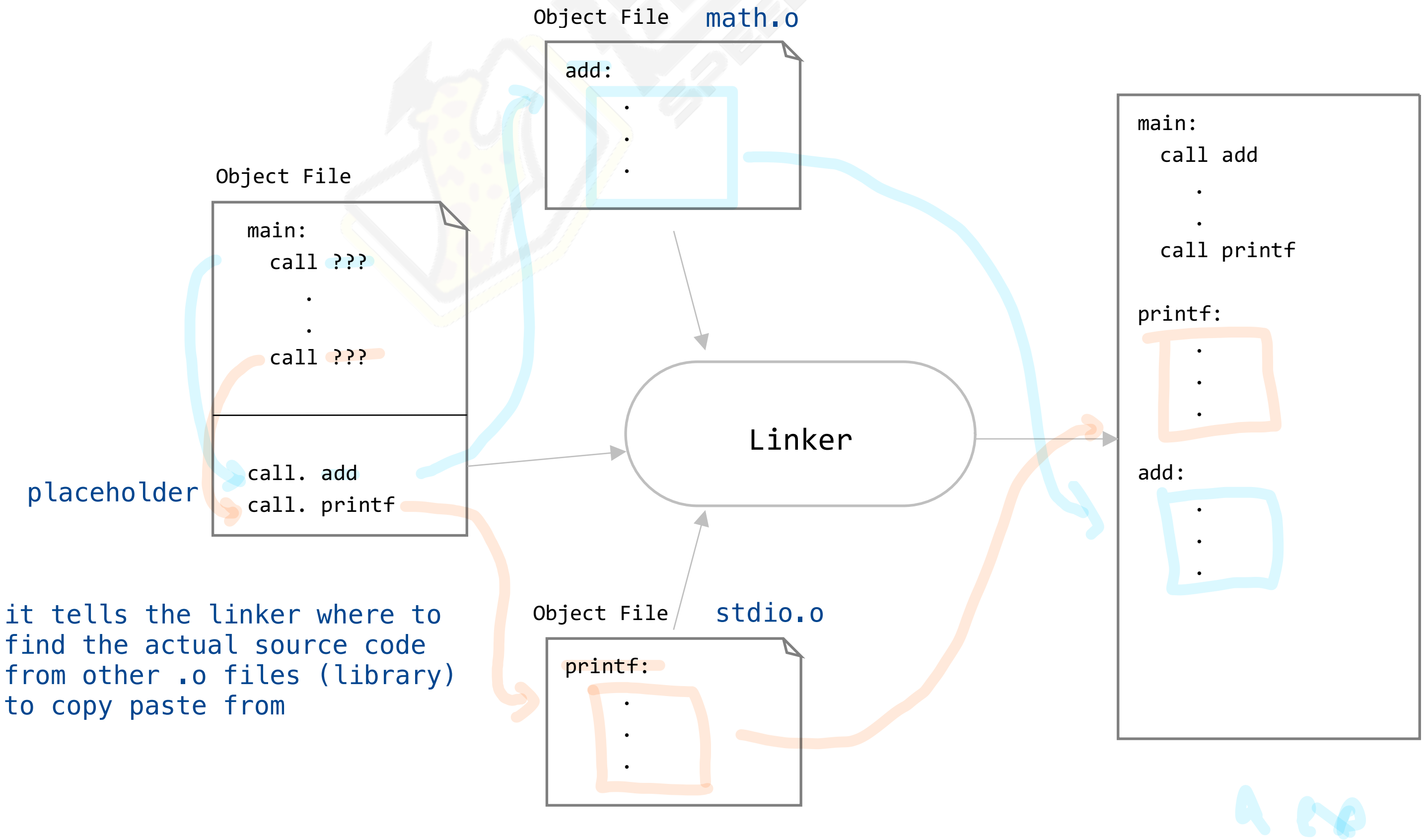
```
     .section __TEXT,__text,regular,pure_instructions
     .macosx_version_min 10, 13
     .globl   _main
     .p2align 4, 0x90
_main:                               ## @main
     .cfi_startproc
## BB#0:
     pushq    %rbp
Lcfi0:
     .cfi_def_cfa_offset 16
Lcfi1:
     .cfi_offset %rbp, -16
     movq %rsp, %rbp
Lcfi2:
     .cfi_def_cfa_register %rbp
     subq $16, %rsp
     leaq L_.str(%rip), %rdi
     movl $0, -4(%rbp)
     movl $0, -8(%rbp)
     movl -8(%rbp), %eax
     addl $5, %eax
     movl %eax, -12(%rbp)
     movl -12(%rbp), %esi
     movb $0, %al
     callq    _printf
     xorl %esi, %esi
     movl %eax, -16(%rbp)        ## 4-byte Spill
     movl %esi, %eax
     addq $16, %rsp
     popq %rbp
     retq
     .cfi_endproc

     .section __TEXT,__cstring,cstring_literals
L_.str:                              ## @.str
     .asciz   "%d\n"
```

Place holder & Symbols





```
cffa edfe 0700 0001 0300 0000 0100 0000
0400 0000 0002 0000 0020 0000 0000 0000
1900 0000 8801 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
a800 0000 0000 0000 2002 0000 0000 0000
a800 0000 0000 0000 0700 0000 0700 0000
0400 0000 0000 0000 5f5f 7465 7874 0000
0000 0000 0000 0000 5f5f 5445 5854 0000
0000 0000 0000 0000 0000 0000 0000 0000
3d00 0000 0000 0000 2002 0000 0400 0000
c802 0000 0200 0000 0004 0080 0000 0000
0000 0000 0000 0000 5f5f 6373 7472 696e
6700 0000 0000 0000 5f5f 5445 5854 0000
0000 0000 0000 0000 3d00 0000 0000 0000
0400 0000 0000 0000 5d02 0000 0000 0000
0000 0000 0000 0000 0200 0000 0000 0000
0000 0000 0000 0000 5f5f 636f 6d70 6163
745f 756e 7769 6e64 5f5f 4c44 0000 0000
0000 0000 0000 0000 4800 0000 0000 0000
2000 0000 0000 0000 6802 0000 0300 0000
d802 0000 0100 0000 0000 0002 0000 0000
0000 0000 0000 0000 5f5f 6568 5f66 7261
```



it tells the linker where to find the actual source code from other .o files (library) to copy paste from

Application User Interface (API)　　使用者介面 菜單？

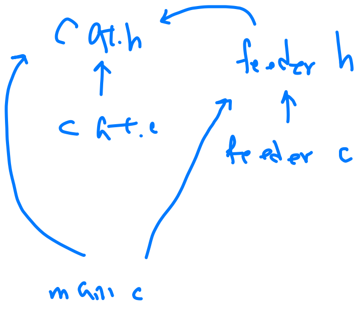
Header file

if no defined

```
#ifndef __cat_header
#define __cat_header

#include<stdlib.h>
#define MAX_STR_LEN 1024

typedef struct cat {
    char name[MAX_STR_LEN];
    float hunger;
} Cat;

cat* create_cat(char name[MAX_STR_LEN], float hunger);
void feed_cat(Cat *catp);
#endif
```

gaurding

public function signature

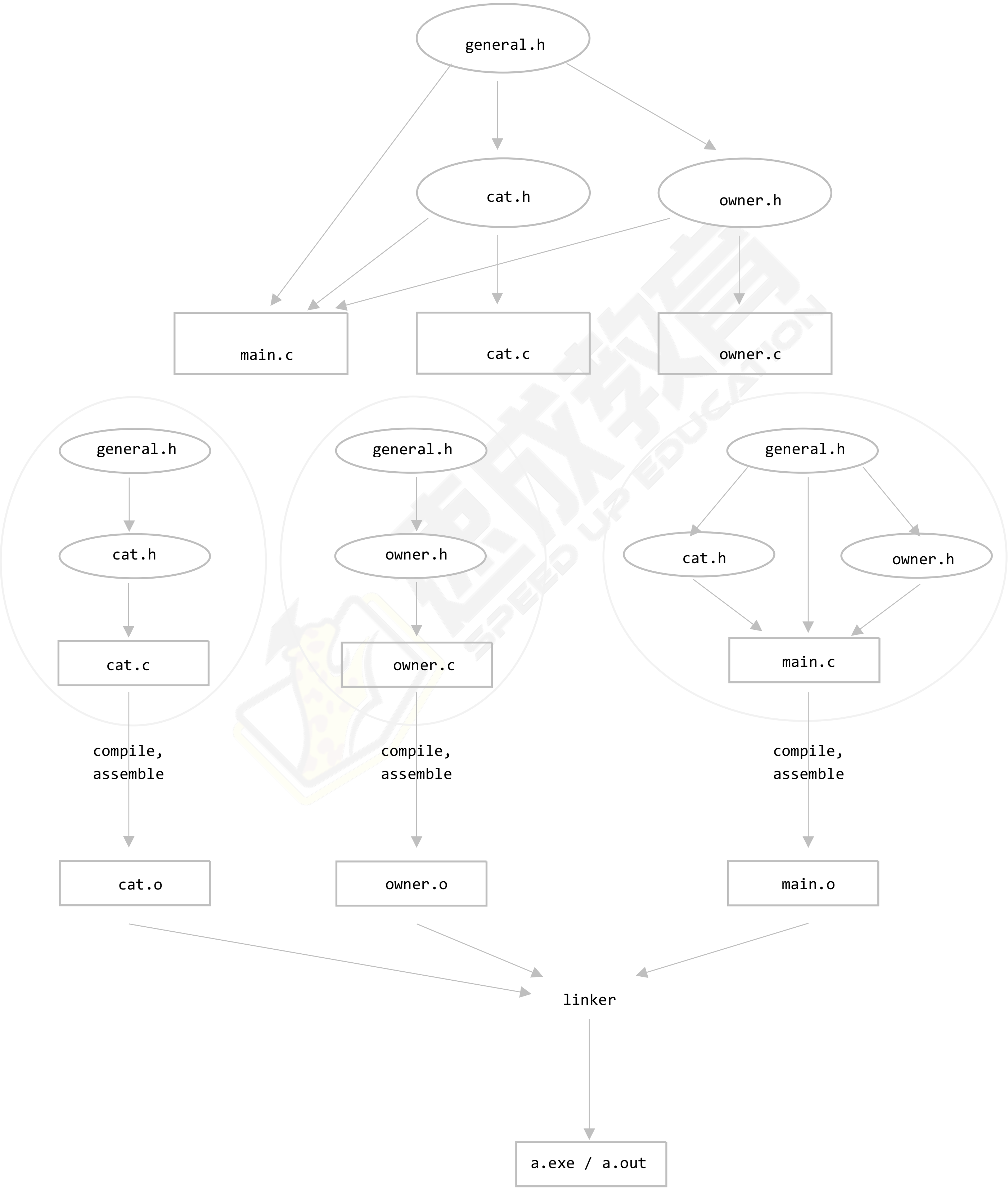


```
#include "cat.h"

#define FOOD 10

Cat create_cat(char name[MAX_STR_LEN], float hunger) {
    Cat *catp = calloc(1, sizeof(Cat));
    strcpy(catp->name, name);
    catp->hunger = hunger;
    return catp;
}

void kill_cat(Cat *catp) {
    free(catp);
}

Cat* feed_cat(Cat *catp) {
    Cat *fake_catp = create_cat(catp->name, catp->hunger);
    kill_cat(catp);
    fake_catp->hunger += FOOD;
    return fake_catp;
}
```

private function

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

int main() {
    Cat *catp = create_cat("Brian", 2);
    catp = feed_cat(catp);
    printf("%s %f\n", catp->name, catp->hunger);
}
```

HOW GCC compile with self-defined header files



Header

- **Things to NOT put in header files**
  - Private function prototypes and constants
  - Global variable

- **Things to put in header files**
  - Include guard
  - Other include file
  - New data type
  - Public function prototypes and constants

Software Design

```
cat.c
stdio.c/h
feeder.c
main.c
```

- **Modularity**
  - Our software is composed of separate modules each of which has one particular task, and each of which is mostly self-contained, can be understood, tested, and maintained independently of the others.
  - A well thought modular program will help:
    - Reduce replication of code.
    - Improve the chance that code you write will be reused.
    - Make it easier to test your code and verify it is correct.
    - Help you see the big picture of how your software is structured and how it works.

- **Reusability**
  - Ensure any modules we write for a specific task can be re-used by any other application that requires that specific task solved. For instance, if we develop a module that to find the shortest path between two nodes in a graph (which is a very common problem in many application domains). We want our implementation to be such that anyone needing to find the shorted path between graph nodes can take our module and build it into their application.

- **Extendibility**
  - We want our software to be easy to extend and improve. This allows us to build on software over time by improving and expanding its usability and functionality.

- **Maintainability**
  - Our software must be organized, easy to understand, well documented, and be free of unnecessary complexity. This improves our ability to test it, debug it, and upgrade it as needed over time. A competent developer not familiar with your code should be able to quickly get to the point where they can work on/with it.

- **Correctness**
  - Any software we develop and release must have been thoroughly tested and made as close to bug-free as possible. Where appropriate, suitable tools should be used to determine correctness, code should have been reviewed by experienced developers not related to its implementation, and a suitable process must be in place for documenting, keeping track, and taking care of bugs found after the software is released

- **Efficiency**
  - We have spent a good amount of time thinking about complexity and how to study the efficiency of our algorithms. We expect good code to be efficient both in terms of the algorithm chosen to solve a problem, and also in terms of how that algorithm is implemented (remember, at the end of the day, the constant terms will make a difference between different implementations of the same algorithm).

- **Openess**
  - When possible (e.g. when we're not developing software for a company that has a stake on what we develop), we should consider contributing to the open source software community. There is a lot of good work done for no other gain than to provide something useful for others. And we can contribute to this effort. Open-source software projects are a good way to make sure your work directly benefits others!

- **Privacy and security**
  - More and more, the software you write will be part of a system running over a network (possibly hosted somewhere else than the user's computer). Therefore, it's important to pay close attention to the best current practice in secure data exchange, as well as having a reliable solution for safely storing a user's personal information.

Application Programming Interface (API)

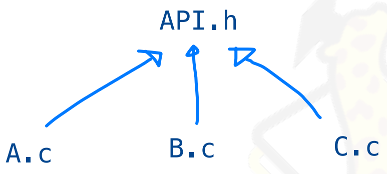
Suppose you are writing a function that finds the most similar string to a given string from a list of strings.

Provide the function declaration so that the users will know what are the expected inputs and output.

```
char *most_simliar(char strings[N][1024], int length, char target[1024]);
```

What do the users need to do, in order to use (call) the function?

What do we need to be careful when creating API and why?



```
1.  API Update

   every file that includes this API
   need to re-complile

2. Even Worst

   API 1.0 has function takes int args

   API 2.0 changed to float args

   They all need to change their codes
   in order to use the new version
```

Object Oriented Programming (OOP)

Encapsulation means wrapping together all the components required to implement the functionality of a specific data type, data structure, or software module. This includes all the data as well as the functions that manipulate it. It requires access control to data and functions that manipulate it in such a way that the designer of the module can determine what data and functions will be accessible to the user, and what legitimate use-cases will be supported. Very importantly, encapsulation requires that we be able to hide data and functionality from the user thus preventing accidental or intentional misuse.

Encapsulate 包裝！！！

Consider the following program in C.

```
#include "CatNode.h"

 typedef struct cat_node {
     char *name;
     float hunger;
     struct cat_node *next;
 } CatNode;


 CatNode *insert(CatNode *head, char *name, float hunger) {
     ...
 }

 CatNode *search(CatNode *head, char *name) {
     ....
 }
```

Cat

clean_cat()

feed_cat()

```
 #include "CatNode.h"

 int main () {
     CatNode *head = NULL;
     Head = insert(head, "Brian", 1);

     CatNode *newnode = calloc(1, sizeof(CatNode));
     newnode->next = head;
     head = newnode;
 }
```

cat
clean_cat(cat)

自己喂cat吃屎

貓死了

How do we solve this in C++

```
#include "cat.h"

typedef struct cat_node {
    char *name;
    float hunger;
    struct cat_node *next;
} CatNode;
                                                      s.upper()
                                feilds / attributes
class CatList {                                       L.append()
    private:
        CatNode *head;

        kill_cat(CatNode p) { ... };

    public:
        CatList () { head = NULL;}   constructor: auto call when a new object is created

        ~CatList() { ... }           destructor: auto call when a object is being deleted
method
        void insert(char *name, float hunger) { ... }

        void delete(char *name) { kill_cat(search(name)) ... }

        CatNode search(char *name) { ... }

        void print(); //defined inside class
}


// Definition of print using scope resolution operator ::
void CatList::print()
{
    ...
}
```

```
#include "cat.h"

int main() {
    CatList cl;
    cl.insert("Brian", 1);
    cl.insert("Vicent", 2);
    CatNode brian = cl.search("Brian");
    cl.print();

    printf("%s\n", cl.head->name); <- Error: CatList has no member: "head"
    cl.kill_cat(brian);
}
```

OOP Inheritance & Method Overloading

Consider the follow class for a basic human being.

```
// Base class
class Human {

    private:
        char name[1024];              feilds / attributes
        int weight;
        int height;

    public:
        Human() { weight = 0; height = 0; name = "skr"; }
        Human(char *n) { stcpy(name, n); weight = 0; height = 0;}
        Human(char *n, int w, int h) { strcpy(name, n), weight = w, height = h;}
        void setWeight (int w) { weight = w; }
        void setHeight(int h) { height = h; }
        void setName(char *n) { strcpy(name, n); }                    method overload
        int getWeight() { return width; }
        int getHeight() { return height; }                            1. same method name
        char* getName() { return name; }                              2. same return type
};                                                                    3. diff params (數量, type)

int main() { vc("Vincent"); }           vc.getName() -> "Vincent"
             paco("Paco", 3, 100);      vc.setName("VC")
```

Now, how would you create a class for a student.

Human class and Student class

```
// Base class           Parent Class         Super Class
class Human {

    protected:
        char name[1024];
        int weight;
        int height;

    public:
        Human() { weight = 0; height = 0; name = "skr"; }
        Human(char *n) { stcpy(name, n); weight = 0; height = 0;}
        Human(char *n, int w, int h) { strcpy(name, n), weight = w, height = h;}
        void setWeight (int w) { weight = w; }
        void setHeight(int h) { height = h; }
        void setName(char *n) { strcpy(name, n); }
        int getWeight() { return width; }
        int getHeight() { return height; }
        char* getName() { return name; }
        void get_info() { printf(name) };

};

// Derived class          Child Class          Sub Class
class Student: public Human {
    protected:
        char student_number[1024];

    public:
        Student(char *n, int w, int h, char* sn) {strcpy(name, n), width = w, height = h;
                                                  strcpy(student_number, sn)}

        void setStudentNumber(char *sn) { strcpy(student_number, sn); }
        char* getStudentNumber() { return student_number; }
        void failCourses() { ... WTF? ... }
        void get_info() { printf(name, student_number) };
}
```

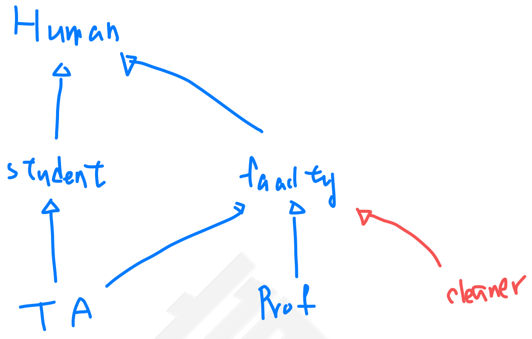
method override

1. child 覆盖 parent
2. same function name, return type, params

```
#include "human.h"
int main(void) {
   Human vc("Vincent");
   Student brian;
   printf("%s\n", brian.getStudentNumber());


   return 0;
}
```

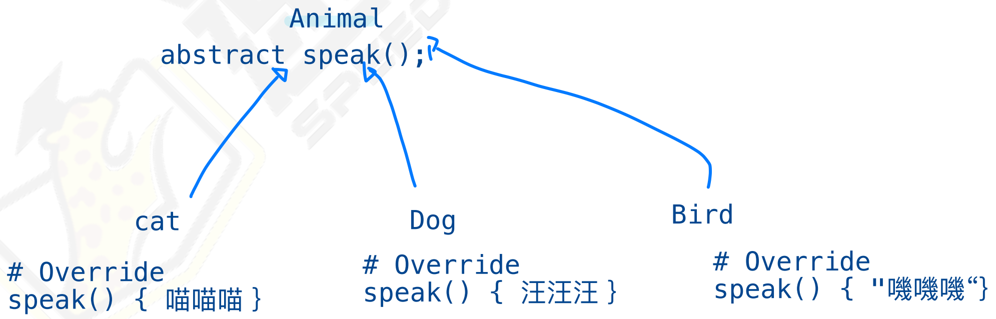
OOP Multiple Inheritance

Now, you want to add two more classes: TA and Professor.



OOP Abstract Class & Abstract Method

Now, you want to add peekemon and trainer into our system.



Animal
abstract speak();

cat
# Override
speak() { 喵喵喵 }

Dog
# Override
speak() { 汪汪汪 }

Bird
# Override
speak() { "嘰嘰嘰“}

Abstract Class
- same as a regular class (can have variables & methods)
- cannot be instantiated (created)
- can have abstract methods
    - every child classes MUST implement (complete) the body of this methods.
      (MUST mothod override)

```
int main () {
  ~~Animal vc(...);~~

  Cat vc(...);

  Dog paco(...);
```

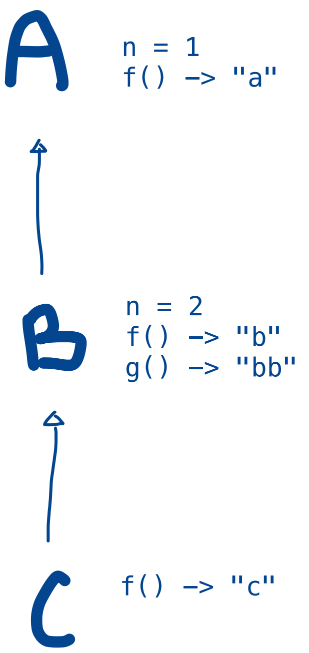
Designing classes

Consider the following, design a system using OOP and draw a diagram.

We need to represent books and authors. Authors should have a first and last name and a mailing address.
Sometimes books change their titles and number of pages, and it we often need to add just one extra page,
so it would be good to have a simple way of doing that. Given a book, should be able to get all of its authors, and given an author I want to be able to get all the books they've written.

Books can be paperback or hardcover, and hardcover books can either be regular or special edition. Paperbacks cost less, and a special edition's price depends on whether or not it has a certificate with it. The number of copies we print will also depend on the type of book, there's a formula for paperback based just on the number of pages, but for hardcover it depends on the author, and for special editions, we always print a fixed number.

Authors are either contractors (who have an agent and are paid solely on the number of pages they've written), or salaried (who have an annual salary based on the number of years they've worked with the company). We need to be able to see for any given author how much they've been paid over the year.



n = 1
f() -> "a"

n = 2
f() -> "b"
g() -> "bb"

f() -> "c"

polymorphism

```
int main() {
    A x;        B y;        C z;
referece type   A arr[10] = { x, y, z };   actual object

    printf("%d\n", arr[0].n); => 1
    printf("%d\n", arr[1].n); => 1
    printf("%d\n", arr[2].n); => 1

    printf("%d\n", ((B) arr[0]).n); => Error
    printf("%d\n", ((B) arr[1]).n); => 2
    printf("%d\n", ((B) arr[2]).n); => 2
    printf("%d\n", ((C) arr[1]).n); => Error
    printf("%d\n", ((C) arr[2]).n); => 2


    printf("%s\n",          arr[0].f())  => a
    printf("%s\n",          arr[1].f())  => b
    printf("%s\n",          arr[2].f())  => c
    printf("%s\n", ((B)     arr[2]).f()) => c

    printf("%s\n",          arr[1].g())  => Error
    printf("%s\n", ((B)     arr[1]).g()) => bb

    printf("%s\n",          arr[2].g())  => Error
    printf("%s\n", ((B)     arr[2]).g()) => bb
    printf("%s\n", ((C)     arr[2]).g()) => bb
}
```

1. Compile (gcc)

```
    if 檢查 ref type 有沒有 method / variable

       a. variable 看 ref type

       b. method: run time -> invoke (調用) actual object override 的 method
    else
       error
```

Polymorphism Example:

```
Cat vc;
Dog paco;
Bird alice;

Animal pets[N] = {vc, paco, alice }

for (int i = 0; i < N; i++){
   pets[i].speak();
}
```