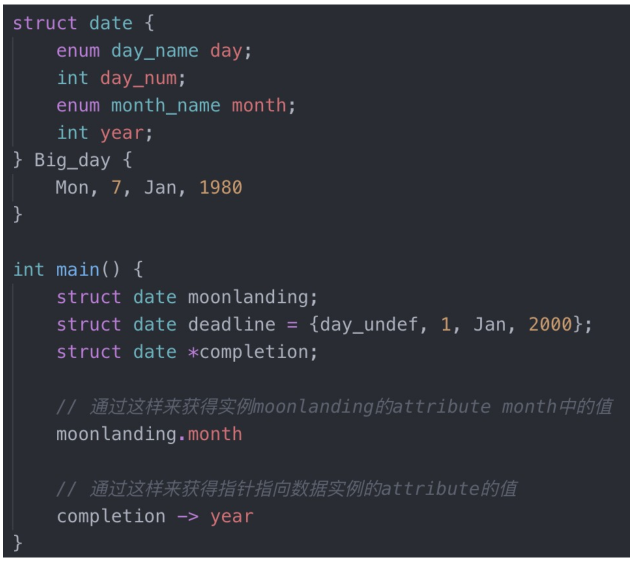
COMP2017 / COMP9017 Tutorial 4

Structs, Unions, Bitfields and Files

Structs and Unions

Within C you can declare a new 'type' using structs. A struct is a collection of existing types with a constant size in memory defined at the time of compilation, and with a packing that is consistent within the architecture of the system and the version of the compiler.

Each field within the struct has its own name and can be called using the dot operator "."



```
struct date {
    enum day_name day;
    int day_num;
    enum month_name month;
    int year;
} Big_day {
    Mon, 7, Jan, 1980
}

int main() {
    struct date moonlanding;
    struct date deadline = {day_undef, 1, Jan, 2000};
    struct date *completion;

    // 通过这样来获得实例moonlanding的attribute month中的值
    moonlanding.month

    // 通过这样来获得指针指向数据实例的attribute的值
    completion -> year
}
```

```
struct coordinate {
        int x;
        int y;
};

struct coordinate point;
point.x = 0;
point.y = 0;
printf("%d %d\n", point.x, point.y);
```

Similarly to arrays, structs can be initialised using curly braces.

```
struct coordinate {
        int x;
        int y;
};

struct coordinate point = { 0, 0 };
//or, to specify fields
struct coordinate point = { .x = 0, .y = 0 };
```

As with all other types in C, we can have pointers to struct types.

```
struct coordinate* point_ptr = &point;
```

Naively accessing fields of a pointer to a struct is somewhat arduous, but C has the arrow operator '->' to simplify this. The following statements are equivalent.

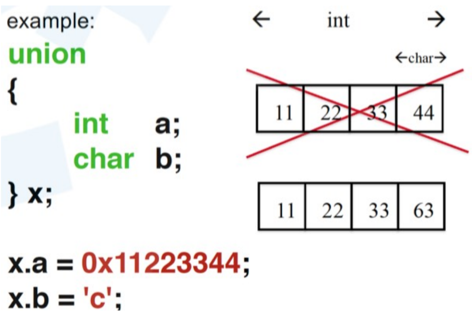
```
(*point).x;
point->x;
```

You may also be wondering about why we brought up that structs need to have a constant size in memory at compile time; this limits the objects that can be stored within a struct. For instance a struct cannot contain an array of non-constant size, though it can contain a pointer to an array located elsewhere in memory. Similarly a struct cannot contain another struct that hasn't been previously declared, though it can contain a pointer to a struct that has yet to be declared. This is possible as the size of a pointer is constant and does not depend on the type of the object it is pointing to.

Unions are a 'special' type of struct that can be used as one of multiple types listed within it. The size of the union is the size of the largest possible type it can take on.

```
union number
{
        int i;
        float f;
        double d;
};

union number n;
n.i = 10;
n.f = 10.05;
n.d = 12.02;
```

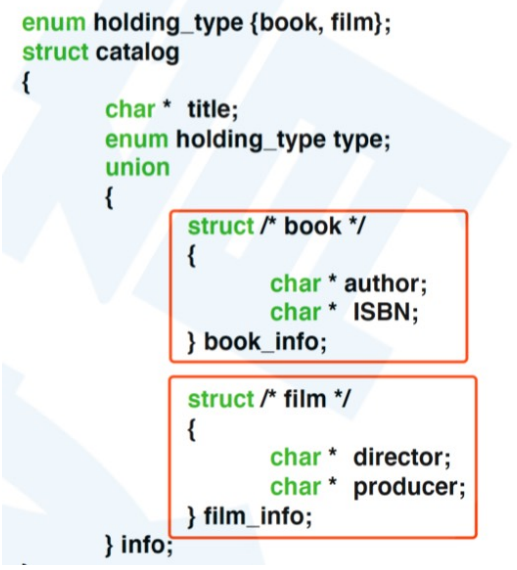
我们只能在定义时声明attribute，其特点在于在union中的所有attribute共享同一块memory。union的size是基于所有其中size最大的attribute来决定的。



example:

```
union
{
    int   a;
    char  b;
} x;

x.a = 0x11223344;
x.b = 'c';
```

Note that unlike a struct, each of these are stored in exactly the same region of memory. When you change the value using the double representation and then attempt to read it as an integer, it will simply interpret the binary data within the first four bytes of this region of memory as an integer. This is **not** the same as casting and will result in garbage.



```
enum holding_type {book, film};
struct catalog
{
        char *  title;
        enum holding_type type;
        union
        {
                struct /* book */
                {
                        char * author;
                        char *  ISBN;
                } book_info;

                struct /* film */
                {
                        char *  director;
                        char *  producer;
                } film_info;
        } info;
```

Q：什么时候要使用union?

A：当我们有多种case，但是每次我们只需要使用一种的时候，注意⚠️确保我们只会access其中一种的数据类型

使用union的话，book和film是共享memory的，不会造成内存浪费

Pre-tutorial Work

Question 1: Sizeof

What is the size in bytes of each of the following objects? Do any of these change depending on the architecture of the computer they are used on?

```
int a;                  4 bytes
int* b = &a;            size of pointer
int* c = NULL:          size of pointer

unsigned d;             4 bytes
short e;                2 bytes
long f;                 8 bytes
size_t g;               8 bytes
long long h;            8 bytes

uint8_t i;              1 byte
uint32_t j;             4 bytes

struct quoll
{
        char name[20];
        uint8_t age;
};

struct quokka
{
        char* name;
        struct quokka* quokka_father;
        struct quokka* quokka_mother;
};

union mammal
{
        struct quoll l;
        struct quokka a;
};
```

struct内存对齐的Note:

- 在struct里，会有内存对齐发生.
- 内存对齐的单位，是以里面的最大member为标准
- 但是，最大member，是按type算的，数组不算. 例如object5里面有一个char c[19]，它贡献的对齐单位是1，里面最大的type是int，于是，这个struct的对齐单位大小是4.

- What is the size of int, short, long? 4 2 8
- What is the size of union mammel, struct quokka and struct quoll 24 24 21
- What is the size of struct quoll* and struct quokka*? 8 8
- Compile with the -m32 flag and report what the differences in sizes.

32bit machine will have pointer size of 4 bytes
64bit machine will have pointer size of 8 bytes

- Does the size of `uint8_t` and `uint32_t` change due to the `-m32` flag? What can be said about the portability of `stdint.h` types?

unit8_t is 8 bit (1 byte), unit32_t is 32 bit (4 byte), it will not change due to -m32 flag

Question 2: Structs Properties

The following questions are based on the following code snippet:

```
enum TYPE { FIRE, WATER, FLYING, ROCK, ELECTRIC };
struct pokemon {
    const char* name;
    enum TYPE type;
};
```

the size of enum is usually 4

1. Which of the following code snippets compile? C D E

   (a) `pokemon pikachu = { "Pikachu", ELECTRIC };` miss struct in front

   (b) `struct pokemon pikachu = { ELECTRIC, "Pikachu" };` order is wrong

   (c) `struct pokemon pikachu = { "Pikachu", ELECTRIC };`

   (d) `struct pokemon pikachu = { .type = ELECTRIC, .name = "Pikachu" };`

   (e) `struct pokemon blank = { 0 };` char array[20] = {{0}}, means assign all the value in array to 0

2. What assumptions can you make about `sizeof(struct pokemon)`? 16

3. What does the following code do?

```
struct pokemon pikachu = { "Pikachu", ELECTRIC };
struct pokemon *ptr = &pikachu;
ptr->name = "Raichu";
ptr->type = ELECTRIC;
```

   modified the value by having the pointer to point that memory

4. What would the following code do?

   local variable copy

```
void evolve(struct pokemon mon) {
    ptr.name = "Raichu";
    ptr.type = ELECTRIC;
}

int main() {
    struct pokemon pikachu = { "Pikachu", ELECTRIC };
    evolve(pikachu);
}
```

   copy of struct pokemon pass in the function, the original object isn't modified

5. Based on the outcome from the previous code segment, what changes could you to the `evolve` function to ensure they modify the object.

   pass in pointer instead of the object itself

Question 3: Greeter

You are to write a program that will play the role as a shop greeter, the program will record all customers that show up by asking them a few questions.

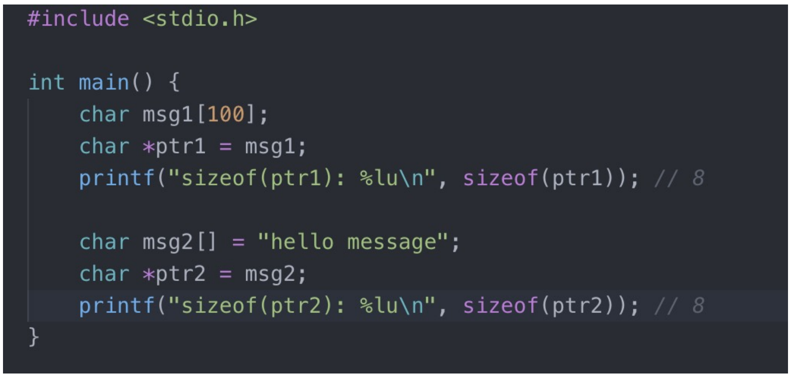
The greeter will ask them their name, age and what they are looking for, afterwards the greeter will allow them to continue into the store.

At the end of the day, the greeter will output all user data to the screen and close, ready for another adventure the following day.

```
$ ./greeter
Welcome to ShopaMocha,
Could you please tell me your name, age and what you looking for?
Lionel 25 Bees

Hrmm, I think you should talk to a ShopaMocha assistant to find "Bees"
Have a good day!
^D
Customer 0, Name: Lionel, Age: 25, Looking for: Bees
```

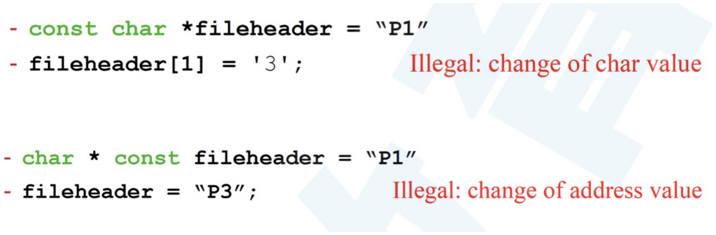
To make testing your code easier, create a few files that contain your keyboard input and use bash redirection when running your program.

记得从右往左读
char *p[3] 一个size为3的array，存的是char pointer
char (*p)[3] 使用括号提高了优先度，一个pointer指向了一个char array



```
#include <stdio.h>

int main() {
    char msg1[100];
    char *ptr1 = msg1;
    printf("sizeof(ptr1): %lu\n", sizeof(ptr1)); // 8

    char msg2[] = "hello message";
    char *ptr2 = msg2;
    printf("sizeof(ptr2): %lu\n", sizeof(ptr2)); // 8
}
```

const char* name —> you cannot change the value
char* const name —> you cannot change the address



```
- const char *fileheader = "P1"
- fileheader[1] = '3';
```

Illegal: change of char value

```
- char * const fileheader = "P1"
- fileheader = "P3";
```

Illegal: change of address value

Question 4: Files

Although piping and redirection on the Unix command line is very useful, this is a feature of Unix operating systems rather than C. The C standard library defines a simple way of interacting with files as shown below.

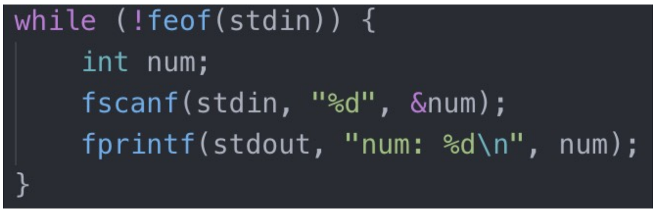
There are four steps to reading from and writing to text files in C.

1. Open the file using `FILE* fopen(const char* path, const char* mode);`
2. Read text from the file using `int fscanf(FILE* stream, const char* format, ...);`
3. Write text to the file using `int fprintf(FILE* stream, const char* format, ...);`
4. Close the file using `int fclose(FILE* fp);`

The `mode` specifies what operations are allowed as well as the behaviour of `fopen` if the file does not exist.

MODE	STREAM POSITION	RESULT
`r`	beginning of file	open file for reading
`r+`	beginning of file	open file for reading and writing
`w`	beginning of file	open for writing, creates an empty file if none exist truncates file to zero length if file exists
`w+`	beginning of file	open for reading and writing, creates an empty file if none exist truncates file to zero length if file exists
`a`	end of file	open for appending, creates an empty file if none exist
`a+`	beginning of file	open for reading and appending, creates an empty file if none exist output is always appended to the end of the file

The C11 standard introduces the `x` specifier that can be appended to `w`, for example: `wx` and `w+x` which causes `fopen` to fail if the file already exists. Now, use your knowledge of file I/O in C to extend the following code.



```c
while (!feof(stdin)) {
    int num;
    fscanf(stdin, "%d", &num);
    fprintf(stdout, "num: %d\n", num);
}
```

```c
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
    // attempt to open a file that does not exist
    FILE* file = fopen("missingno", "r");
    if (file == NULL) {
        perror("unable to open file");
        return 1;
    }
    fclose(file);
    // TODO: write to a file of your choice using fopen and fprintf
    // TODO: read from a file of your choice using fopen and fscanf
    // TODO: write to stdout and stderr using fprintf
    return 0;
}
```

r：只读（文件必须存在）
w：只写（如果文件存在，新的内容会覆盖旧的内容，如果没有文件，则创建新的文件
r+：读和写（文件必须存在）
w+：读和写（不用必须）

fscanf(stdin, …) 等同于 scanf()
fprintf(stdout, …) 等同于 printf()

Tutorial Work

$./printnum < twonum.txt
在这里，我们把txt文件在程序运行中输入进去。这时候stdin就等同于这个文件的file pointer。如果没有文件输入，则键盘就是输入。

Question 5: Discussion

Discuss with your tutor the answer to questions 1 and 2. Consider the struct alignment and if fields have been padded.

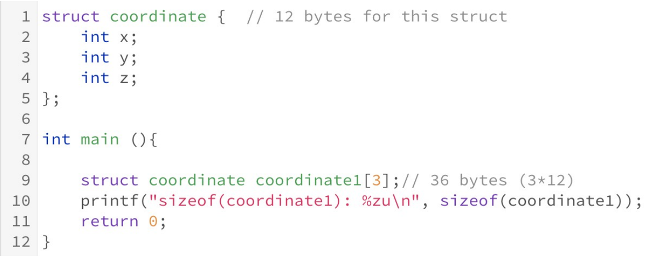
在多次scanf的时候，每一次我们需要使用fflush来清空stdin，不然的话，每一次scanf都只会使用第一次scan的值
fflush(stdin);

Question 6: wc

Implement the wc program in C. Your program should behave in a similar way to the Unix wc program. If no arguments were passed to your wc, read from stdin and print the number of lines, words and characters. For every argument that was passed to your wc, it should read the arguments as files and output the number of lines, words and bytes for each file with the respective file name, and also output the total counts for the files.

Use the isspace function to determine if a character is whitespace. A contiguous string of non whitespace characters count as a word. We recommend that you use the fread and fwrite functions for this exercise.

Compare your output to the Unix wc.

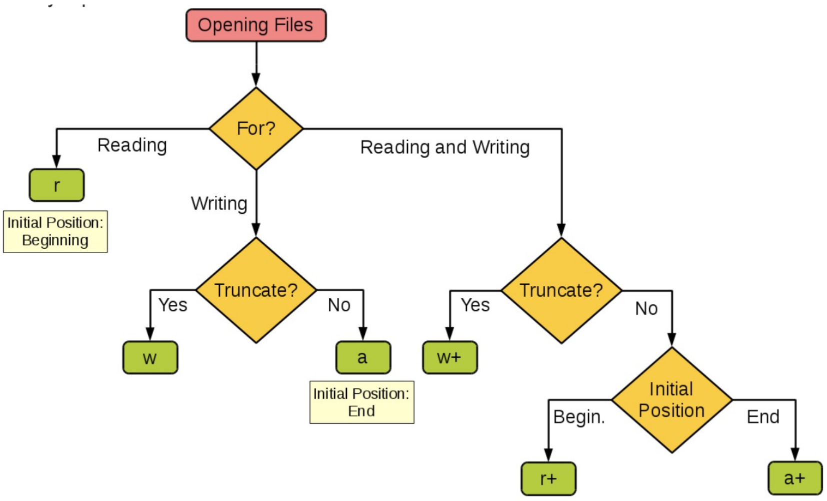


```
struct coordinate {  // 12 bytes for this struct
    int x;
    int y;
    int z;
};

int main (){

    struct coordinate coordinate1[3];// 36 bytes (3*12)
    printf("sizeof(coordinate1): %zu\n", sizeof(coordinate1));
    return 0;
}
```

```
$ echo hello | ./wc
    1    1    6

$ ./wc file1 file2 file3
    <lines>    <words>    <bytes>    file1
    <lines>    <words>    <bytes>    file2
    <lines>    <words>    <bytes>    file3
    <lines>    <words>    <bytes>    total
```

print number of lines in a file: wc -l file.txt
print number of characters in a file: wc -m file.txt
print number of bytes in a file: wc -c file.txt
print number of words in a file: wc -w file.txt

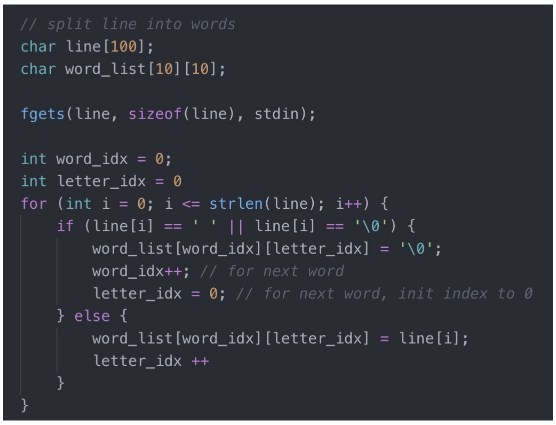
Extension: use the mmap function to directly map a file into memory instead of reading the file byte by byte.



Question 7: Replace words

Write a program that will search for words within a file and replace it with a word from command line arguments.

```
$ cat rick.txt
Always gonna give you up
Always gonna let you down
Always gonna run around and desert you
Always gonna make you cry
Always gonna say goodbye
Always gonna tell a lie and hurt you

$ ./replace_word rick.txt astley.txt Always Never
$ cat rick.txt
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
```



```
// split line into words
char line[100];
char word_list[10][10];

fgets(line, sizeof(line), stdin);

int word_idx = 0;
int letter_idx = 0
for (int i = 0; i <= strlen(line); i++) {
    if (line[i] == ' ' || line[i] == '\0') {
        word_list[word_idx][letter_idx] = '\0';
        word_idx++; // for next word
        letter_idx = 0; // for next word, init index to 0
    } else {
        word_list[word_idx][letter_idx] = line[i];
        letter_idx ++
    }
}
```

Question 8: Scores to CSV

Using the `struct batsman` definition, output the fields of an array of `struct batsman` to a comma-separated value file (CSV). You can use your solution to Question 4 from last week's tutorial to help input data and test your program.

```
struct batsman {
        char first_name[20];
        char last_name[20];
        int score[20];
};
int output_scores(struct batsman* batters, const char* filename);
```

CSV Output Example:

```
Cameron,Bancroft,40
Mitchell,Marsh,67
David,Warner,59
Ben,McDermott,Duck
Cameron,White,78
Usman,Khawja,54
...
```

method 1: fgets + sscanf

read each line which you assumed contained: first name, last name and score

method 2: scanf

scanf version works for proer input, broken input is hard to detect

Extension: Create a program that will convert a tab-separated value file to a comma-separated value file and vice versa.

first: while loop, append the batmans to the fire (handle each character)

second: ArrayList using malloc and storing all the batsman in the list

Question 9: Saving Pokemon

The following code reads the names and levels of four Pokemon from `stdin` and then dumps the contents of the `pokemons` array to a binary file using `fwrite`.

Examine the contents of binary file with the `xxd` hex dump program. Does it contain what you expect? Is `sizeof(pokemon)` larger than its components, if so, why? How are the data types represented in memory?

```
#include <stdio.h>

#define SIZE 4

typedef struct pokemon {
        char name[100];
        unsigned level;
} pokemon;

int main(void) {

        pokemon pokemons[SIZE];

        for (size_t i = 0; i < SIZE; i++) {
                if (scanf("%99s %u",
                                pokemons[i].name,
                                &pokemons[i].level) != 2) {
                        fprintf(stderr, "unable to read input\n");
                        return 1;
                }
        }

        printf("sizeof(size_t)  = %zu\n", sizeof(size_t));       8
        printf("sizeof(unsigned)  = %zu\n", sizeof(unsigned));   4
        printf("sizeof(char[100])  = %zu\n", sizeof(char[100]));100

        printf("sizeof(pokemon)  = %zu\n", sizeof(pokemon));     104
        printf("sizeof(pokemons)  = %zu\n", sizeof(pokemons));   416
        // attempt to save to file
        FILE* file = fopen("pokemon.dat", "w");
        if (file == NULL) {
                perror("unable to open file for writing");
                return 1;
        }
        fwrite(pokemons, sizeof(pokemon), SIZE, file);
        fclose(file);
        return 0;
}
```

Question 10: Working with a stream

You will need to write a program that will read from a stream. This stream emulates data provided by a gamepad. You can access the program from the resources section. This program will create a file that the process and send data to that file. Your program will need to receive those packets and decode them.

Use the data layout below to help with decoding the data.

```
id: unsigned byte
locked: unsigned byte
buttons: 2 bytes
  x: unsigned bit
  y: unsigned bit
  z: unsigned bit
  w: unsigned bit
  a: unsigned bit
  b: unsigned bit
  c: unsigned bit
  d: unsigned bit
  l: unsigned bit
  r: unsigned bit
  st: unsigned bit
  sel: unsigned bit
analog: 8 bytes
  left: float
  right: float
```

After receiving the packet, the reading process needs to output what buttons have been pressed and the current state of the analog sticks. If a button has been released it will need to show this state information between one read and another. If a button has maintained its current value for more than 3 packets, it is considered to be a `Hold`.

Example:

```
PKT0: Analog Left: 0.0000, Analog Right: 0.0000
PKT1: X: Pressed, Analog Left: 0.0000, Analog Right: 0.0000
PKT2: X: Released, Analog Left: 0.0000, Analog Right: 0.0000
PKT3: X: Pressed, Analog Left: 0.0000, Analog Right: 0.0000
PKT4: X: Pressed, Analog Left: 0.2400, Analog Right: 0.0000
PKT5: X: Pressed, Analog Left: 0.2400, Analog Right: 0.0000
PKT6: X: Hold, Analog Left: 0.0000, Analog Right: 0.0000
...
```