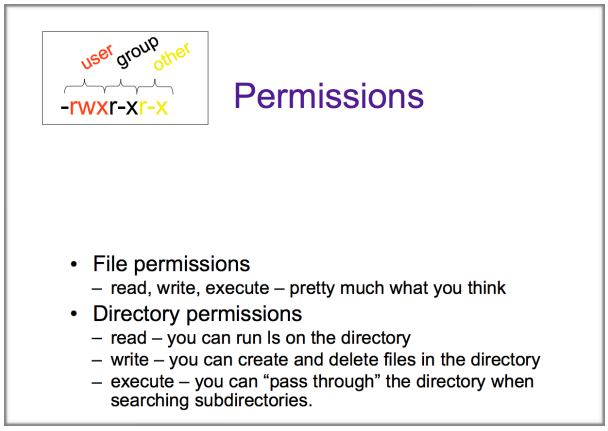
Tuesday, February 20, 2018

CSC209 week one

Program: a sequence of instructions to a computer.

The permission of a file using ls -l -rwx-wxr-x就是如下图显示的。




```
int temperature = 32;
    temperature = 98.1;
```

即使我们这样assignment了temperature， 这个数值会变成98 而不是98.1.
引文 temperature 的type是 int 不是double or float.

```
9%4 = 1;
9/4 = 2;
```

char 可以用Ascii code 去assignment， char letter =97; 和char letter = 'a'; 是一样的。

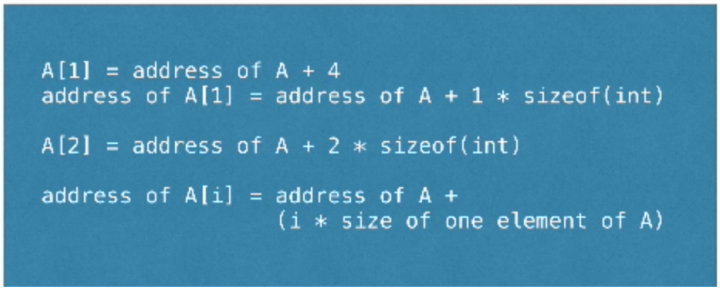
"%.3f"代表了float point 后面取三位。

```
gcc -Wall -std=gnu9 -o hello -g hello.c
```

```
scanf("%lf",&cm);scanf single variable.
scanf("%lf %lf",&cm,&good); scanf multiple variable.
```

week two

int 一般是4byte。 address of A 和 A[0]的address 一样。



```
A[1] = address of A + 4
address of A[1] = address of A + 1 * sizeof(int)

A[2] = address of A + 2 * sizeof(int)

address of A[i] = address of A +
                  (i * size of one element of A)
```

要assign 一个pointer 的值， 要dereference first 然后在赋值。

```
type k;
type *p = &k;

int n;
p = p + n;        // Read as:
                  // p = p + (sizeof(type) * n)
```

```
p[k] == *(p + k)
```

st

Stack is a LIFO (last in first out) data structure.

argv[0] is the name of the executable
argv[0] = ./hallo

string:
1.char array with a null character '\0', null terminate 也会占一个位置.
2.char text[20]="hello"; 这个里面可以改 text[0]= 'j'; jello
3.char *text = "hello" 是string literal 可以change string variable 但是不能change content。会导致runtime unexpected behaviour。
4.char text[20];这样initialize 之后不能直接让 text = "good"；

sizeof(text)给的是这个 array 占的多少size 而不是里面这个 string 有多少个长度 sizeof(text)=20；
用string function 必须 加上#include <string.h>

strlen(text) = 5； 这个是看这个string content 有多少个characters 不包括'\0'.

char total[20] = strcpy(text,goog); 不可以这样写。。
char *total = strcpy(text,goog); 破费

strcpy 里面的 a 不能是char * text只能是 char text[],b 可以是任意
strcpy(a,b)相当于把b 弄到a 里面去 如果a的length 小于b的话 会出错误
不同的compiler 会有不同的表现。

strncpy(a,b,n); n 表示了最多 a可以hold 多少个 characters 包括了 null terminator from b to a，如果没有那么要手动加上去 '\0'.

string 相加：
strcat
strncat(a,b,n) n是多少个b 加到a 里面不包括'\0' always add '\0' to a;
strncat(a,b,sizeof(a)-strlen(a)-1)减1确保了 有'\0'的位置。

char*p=strchar(a,b);return b 是不是在 a里面 b是一个 char 然后return 的是b的address 要知道 b的size 要p-a；
char*p=strstr(a,b)就是找b在a里面第一次出现的位置
p-a就可以知道位置了

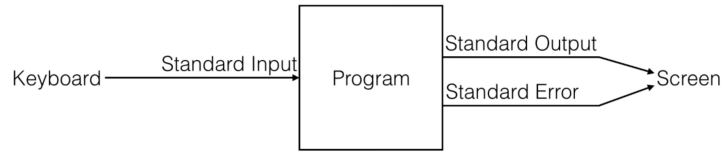
struct：

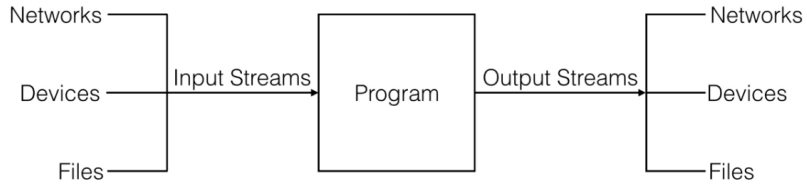
```
struct name {
 char first_name[20];
 char last_name[20];
 int year;
 float gpa;
}
struct name name1;
```

array 丢进function 的时候不是用copy 是用的自己。
struct 使用copy 当他 pass 进 function。
所以我们创建的时候 用 struct name *name1；来initialize。

Stream：c写东西output 东西都是用stream

Stream	Default Location
Standard Input	Keyboard
Standard Output	Screen
Standard Error	Screen





Files：
FILE *fopen(char* filename,const char* mode);(r read)(w write)(a append)

File * good_file;
good_file = fopen(“file.txt”,”r”);这个地方的r 是双引号。

```
if(good_file == NULL) {
        fprintf(stderr,”Error open the file\n”);
        return 1;
```

}这个是检查能不能打开file的 如果等于NULL 就不能打开file 要 fprintf到 stderr 然后一个error message。

```
if(fclose(good_file)!=0) {
        fprintf(stderr,”fclose failed”);
        return 1;
```

} fclose return value 不等于0的时候就是不能正确的close file。

char* fgets(char* a, int n, FILE *name); n 包括了’\0’; 如果pass 了 n= 5 那么最多4个 characters。

```
while(fgets()!=NULL) {
……
}
```

fgets 以\n 为一个终结

scanf 以空格为一个终结

int fscanf(FILE *stream,const char format..);
return value 是 fscanf 一次读了多少个。

fprintf(FILE * stream,”dddd%d”,integer);

Binary file:

和一般file 差不多的 但是mode 后面加个b like “rb” “wb” 什么的

write to binary file
size fwrite(const *ptr,size,number,FILE *stream);
return value is number of character success write into file.

read from binary file
size fread(*ptr,size,number, FILE * steam);
fread 可以直接写一个array
return value is number of characters success read

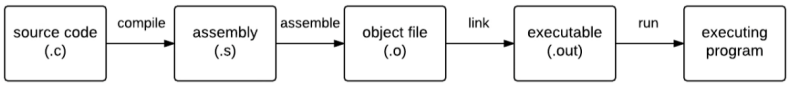
WAV FILE
44 byte 是这个file的信息。 header
od -A d -j 44 -t d2 short.wav

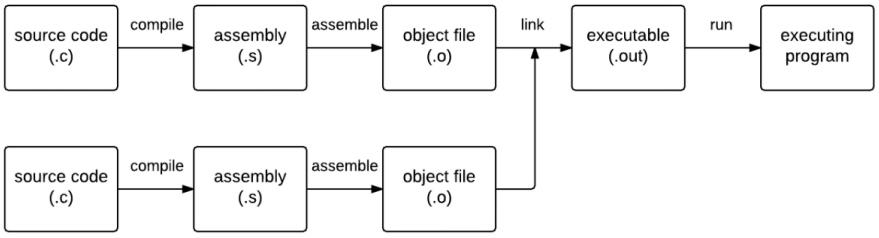
同样也可以把struct 写进binary file 直接丢进去 用fwrite

int fseek(FILE *stream, long int offset, int whence);
SEEK_SET 从头
SEEK_CUR 从当前位置
SEEK_END 从最后
第二个是移动多少个byte
用负数表示回去移动
rewind(FILE *stream); 从头开始

Compiler:
gcc create an executable file。
gcc -S hello world.c 转换成 assembly language
//as hello world.c -o hello world .o 把assembly code 转换成.o file 这一步还差一步 才能用.out run gcc -S 直接两步一起执行了。
每个.ou file都是特定的mechine 特定的操作系统的 不能随便换

gcc hello world.s
装换成 helloworld.o





header can contain more than prototype, typedef 也可以包括进去。

每一个rule 包括这些东西
前面是tab 不是空格 在recipe前面

底下这样写的时候是 任何一个 dependencies 变了 都会执行底下的recipe。

如果没有 argument in make file 然后我们进行 make 他只会执行第一个rule

如果 make compare_sorts
这个会看 compare_sorts.o 和 sorts.o
如果这两个也是target 那么就会去看 他们那个target的 dependencies
然后进行变化

Makefiles:
compare_sorts:compare_sorts.o sorts.o
	gcc compare_sorts.o sorts.o -o compare_sorts

```
compare_sorts.o: compare_sorts.c sorts.h
    gcc -c compare_sorts.c -o compare_sorts.o

sorts.o: sorts.c sorts.h
    gcc -c sorts.c -o sorts.o

compare_sorts: compare_sorts.o sorts.o
    gcc compare_sorts.o sorts.o -o compare_sorts


>rm *.o
>make
gcc -c compare_sorts.c -o compare_sorts.o
>make compare_sorts
gcc -c sorts.c -o sorts.o
gcc compare_sorts.o sorts.o -o compare_sorts
>touch compare_sorts.c
>make compare_sorts
gcc -c compare_sorts.c -o compare_sorts.o
gcc compare_sorts.o sorts.o -o compare_sorts
```

```
%.o: %.c sorts.h
    gcc -c $< -o $@

compare_sorts: compare_sorts.o sorts.o
    gcc compare_sorts.o sorts.o -o compare_sorts
```

这个第一行 replace 所有的 source file to object file

```
.PHONY: clean
clean:
    rm compare_sorts *.o
```

这个.PHONY makes clean is not a actually file, it is just a legal target.

```
.PHONY: clean
clean:
    rm compare_sorts *.o
```

```
>make clean
rm compare_sorts *.o
>make
gcc -c compare_sorts.c -o compare_sorts.o
gcc -c sorts.c -o sorts.o
gcc compare_sorts.o sorts.o -o compare_sorts
```

更多视频

```
OBJFILES = compare_sorts.o sorts.o

%.o: %.c sorts.h
    gcc -c $< -o $@

compare_sorts: $(OBJFILES)
    gcc $(OBJFILES) -o compare_sorts

.PHONY: clean
clean:
    rm compare_sorts *.o
```

就相当于make 里面可以带 variable

typedef unsigned int size_t:

```c
#include <stdio.h>

typedef unsigned int size_t;
typedef unsigned int age_t;
typedef unsigned int shoe_size_t;

void print_boot_size(shoe_size_t shoe_size) {
    printf("boot size:%d\n", shoe_size + 2);
}
```

```c
#include <stdio.h>

typedef unsigned int size_t;
typedef unsigned int age_t;
typedef unsigned int shoe_size_t;
typedef struct {
    char first_name[20];
    char last_name[20];
    int year;
    float gpa;
} Student;

void print_boot_size(shoe_size_t shoe_size) {
    printf("boot size:%d\n", shoe_size + 2);
}

int main() {
    age_t my_age = 5;
    print_boot_size(my_age);
    print_boot_size('5');

    Student s;
    Student *p;
    return 0;
}
```

Tuesday, February 20, 2018

Macros:

```
#define CLASS_SIZE = 80
```

用macro 的时候不需要加=号 直接用 上面那个是错的 应该是#define CLASS_SIZE (80)

```
#include <stdio.h>

#define TAX_RATE .80
#define WITH_TAX(x) ((x) * 1.08)

int main() {
    double purchase = 9.99;
    printf("%f\n", WITH_TAX(purchase));

    return 0;
}
```

No Space Here

```
#include <stdio.h>

#define TAX_RATE .80
#define WITH_TAX(x) (x * 1.08)

int main() {
    double purchase = 9.99;
    double purchase2 = 12.49;
    printf("%f\n", WITH_TAX(purchase + purchase2));
    printf("%f\n", (purchase + purchase2 * 1.08));
    return 0;
}
```

Useful C Features Video 2: Macros

```
#include <stdio.h>

#define TAX_RATE .80
#define WITH_TAX(x) ((x) * 1.08)

int main() {
    double purchase = 9.99;
    printf("%f\n", WITH_TAX(purchase));        printf("%f\n", ((purchase) * 1.08));

    return 0;
}
```

PREPROCESSOR:

prep processor 会把 42 当成一个string， 他只是把 text 转换成text

ex1.c

```
#define MY_INT (42)

int x = MY_INT;      // Assigning defined value MY_INT
```

Command Line

```
>cpp ex1.c
```

更多视频

这些line 啊 FILE 啊什么的 是predefine 的 MACRO 他必须用_LINE_ 这种形式表现

ex2.c

```
#include <stdio.h>

int main() {
  printf("Line %d: %s compiled on %s at %s.\n", __LINE__, __FILE__, __DATE__, __
TIME__);
  return 0;
}
```

Command Line

```
# 2 "ex2.c" 2

int main() {
  printf("Line %d: %s compiled on %s at %s.\n", 4,
 "ex2.c", "Jun  8 2016", "21:39:28");
  return 0;
}

17 errors generated.
```

更多视频

ex4.c

```c
#include <stdio.h>

#if __APPLE__
const char OS_STR[] = "OS/X";
#elif __gnu_linux__
const char OS_STR[] = "gnu/linux";
#else
const char OS_STR[] = "unknown";
#endif

int main() {
  printf("Compiled on %s\n", OS_STR);
  return 0;
}
```

这种macro 相当于只能true false 来看不能直接printf 一些信息。

ex4.c

```c
#include <stdio.h>

#if __APPLE__
const char OS_STR[] = "OS/X";
#elif __gnu_linux__
const char OS_STR[] = "gnu/linux";
#else
const char OS_STR[] = "unknown";
#endif

int main() {
  printf("Compiled on %s\n", OS_STR);
  return 0;
}
```

ex5.c

```c
#include <stdio.h>

#ifdef __APPLE__
const char OS_STR[] = "OS/X";
#elif defined(__gnu_linux__)
const char OS_STR[] = "gnu/linux";
#else
const char OS_STR[] = "unknown";
#endif

int main() {
  printf("Compiled on %s\n", OS_STR);
  return 0;
}
```

Function Pointers:

aaaa

```
}

double time_sort(int size, void (*initializer)(int *, int)) {
    int arr[size];
    initializer(arr, size);
```

```
double time_sort(int size, void (*sort_func)(int *, int), void (*initializer)(int *, int)) {
    int arr[size];
    initializer(arr, size);

    clock_t begin = clock();
    sort_func(arr, size);
    clock_t end = clock();

    check_sort(arr, size);

    return (double)(end - begin) / CLOCKS_PER_SEC;
}


int main() {
    srand(time(NULL));

    for (int size = 1; size <= 4096; size *= 2) {
        double time_spent = time_sort(size, insertion_sort, max_to_min_init);
        printf("%d: %f\n", size, time_spent);
    }

    return 0;
}
```

they just have the addı

these are system calls and also include exit();

```c
// File management system calls

    int open(const char *pathname, int flags, mode_t mode);
    ssize_t read(int fd, void *buf, size_t count);
    ssize_t write(int fd, const void *buf, size_t count);
    int close(int fd);

// Process management system calls
    pid_t fork(void);
    pid_t getpid(void);
    pid_t wait(int *status);
    int kill(pid_t pid, int sig);
    int execv(const char *path, char *const argv[]);

// Interprocess communication system calls
    int pipe(int pipefd[2]);
    int socket(int domain, int type, int protocol);
    int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
    int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
    int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Which of the following functi

- [ ] exit()
- [ ] fgets()
- [ ] fopen()
- [ ] printf()
- [ ] read()
- [ ] scanf()
- [ ] write()

除了 exit write read 以外都是library call

Errors and Errno