akaedu.org-1

xwp

March 29, 2014

Contents

1 linux c 基础复习

1.1 基本命令

1.2 vim 操作

1.3 c 基础

1.3.1 基本类型

1.3.2 循环控制语句

1.3.3 数组

1.3.4 指针

1.3.5 结构体

2 Ubuntu 系统安装软件方法

2.1 Ubuntu 服务器软件源安装

- Ubuntu 软件中心 –》编辑 –》软件源
- 更新本地软件源列表 sudo apt-get update
- 搜索软件 sudo apt-cache search thunderbird
- 安装软件 sudo apt-get install thunderbird
- 卸载软件 sudo apt-get autoremove thunderbird

2.2 deb 包安装

- 网上下载相关软件 deb 包
- 安装 sudo dpkg -i filename.deb
- 卸载 sudo dpkg -r filename

2.3 源代码安装

- 网上下载相关软件源代码包（.tar.gz/.zip/.tar.bz2/.tar）
- 进入源码包，编译源代码包，并且安装
  - ./configure

  - make
  - sudo make install
- 卸载源代码包安装的软件, 进入到源代码包里执行
  - sudo make distclean

2.4 ubuntu 使用小技巧

- 显示桌面，Ctrl+win+d
- 切换工作区，Ctrl+Alt+up/down/left/right

3 git 使用教程

4 计算机组成原理

4.1 寄存器

4.2 CPU

4.3 内存

4.4 总线

4.5 MMU

5 X86 汇编

5.1 学习汇编的目的

学习汇编不是为了掌握用汇编去写程序，而是为了更好的透视理解C语言行为，理解处理器的工作方式，了解程序背后底层的东西，相当于习武中的内功心法，以便你日后写出高效的代码和调试诡异的BUG。

5.2 第一个汇编程序

```
#PURPOSE: Simple program that exits and returns a
#
status code back to the Linux kernel
#
#INPUT:
none
```

```
#
#OUTPUT: returns a status code. This can be viewed
#
by typing
#
#
echo $?
#
#
after running the program
#
#VARIABLES:
#
%eax holds the system call number
#
%ebx holds the return status
#
.section .data
.section .text
.globl _start
_start:
movl $1, %eax # this is the linux kernel command
#number (system call) for exiting
#a program
movl $4, %ebx # this is the status number we will
          # return to the operating system.
         # Change this around and it will
        # return different things to
       # echo $?
int $0x80 # this wakes up the
```

5.2.1 编译链接过程

.c –> .i –> .s –> .o –> a.out(ELF)

文件类型	文件属性	工具链	目标文件类型
.c	源代码	cpp 预处理器	.i
.i	预处理后的文件	gcc 编译器	.s
.s	编译后的文件	as 汇编器	.o
.o	汇编后的文件	ld 链接器	a.out
a.out	链接后的可执行文件	loader 加载器	进程

5.3 X86 寄存器

寄存器	功能
eax	通用
ebx	通用
ecx	通用
edx	通用
esi	通用
edi	通用
ebp	桢指针
esp	栈指针
eip	程序计数器
eflag	程序状态
浮点寄存器	浮点数运算

5.4 汇编语法

去我的 nfs 服务器上下载汇编电子教材

5.5 寻址方式

```
int a = 3, d = 5;
int b[5] = {1,2,3,4,5};
struct STU {
int id;
char name[20];
char sex;
}c;
int *p;
```

寻址方式	含义	对应 C 语法
直接寻址	movl ADDRESS, %eax	a = *p
变址寻址	movl $\text{data}_{\text{items}}$(,%edi,4), %eax	a = b[3']'
间接寻址	movl(%eax), %ebx	a = *p
基址寻址	movl 4(%eax), %ebx	a = *(p+1)
立即数寻址	movl $12,%eax	a = 12
寄存器寻址	movl %eax, %ebx	a = d

5.6 E L F 文件格式

ELF 文件格式是一种开放的标准，unix/linux 可执行程序都采用 ELF 标准。

- 可重定位的目标文件 (Relocatable, 或者 Object File) 如.o 文件
- 可执行文件 (Executable)
- 共享库 (Shared Object, 或者 Shared Library)

5.6.1 绝对地址和相对地址辨析

- 类别绝对路径和相对路径
- 相对地址：内部偏移地址，两条指令间的相对距离
- 绝对地址：虚拟地址 4G 空间内的唯一地址

5.6.2 Section 和 Segment 辨析

- Section：站在链接器角度的链接单位，如：

段名	含义	解释
Section Headers	段符号表	保存了各个段的起始地址和大小
.text	代码段	存放程序执行的机器码指令
.ro.data	只读数据段	存放只读常量
.data	数据段	存放已初始化的全局变量
.Bss	0 数据段	存放未初始化全局变量
.shstrtab		保存着各 Section 的名字
.strtab		保存着程序中用到的符号的名字
.rel.text	重定位符号表	告诉链接器指令中的哪些地方需要做重定位
.symtab	符号表	保存了符号内部的相对地址

- Segment：站在加载器角度的加载单位，如：

PrProgram Headers:

Type	Offset	VirtAddr	PhysAddr	FileSize	MemSiz	Flg	Align
LOAD	0x000000	0x08048000	0x08048000	0x0009e	0x0009e	R E	0x1000
LOAD	0x0000a0	0x080490a0	0x080490a0	0x00038	0x00038	R W	0x1000

Section to Segment mapping:
Segment Sections...

segment 编号	segment 名
00	.text
01	.data

5.7 查看反汇编调试程序

- gcc -g test.c -o test
- objdump -dSsx test > file
- vim file
- 找到 main 函数的位置，然后开始分析

6 C 和汇编的关系

6.1 重点指令讲解

指令	示例	作用
push		
pop		
call		
leave		
ret		

6.1.1 栈的类型

栈的类型	工作方式
满递减	
满递增	
空递减	
空递增	

6.1.2 call

call 0x80a0000<foo>

- 把 call 的下一条指令的地址压到栈上保存
- 修改 eip，跳到 foo 函数的地址上去执行

6.1.3 ret

call 指令的逆向操作

- 把当前 esp 里保存的返回地址值给 eip
- 由于修改了 eip，所以跳向返回地址

6.1.4 leave

是函数开头的 push %ebp 和 mov %esp,%ebp 的逆操作

- 把 ebp 的值赋给 esp
- 现在 esp 所指向的栈顶保存着 foo 函数栈帧的 ebp , 把这个值恢复给 ebp , 同时 esp 增加 4

6.1.5 总结

- x86 平台函数调用，利用栈来传递参数
- 每个函数都有自己的桢指针，称为函数的栈桢，通过函数的栈桢向上可以取到调用函数的参数，向下可以取到函数定义的局部变量
- 函数调用时从右到左去压栈

6.2 变量存储布局

6.2.1 存储区域

存储区域	存储元素
栈	局部变量
data 段	已初始化全局变量，static 修饰并初始化的变量
bss 段	未初始化全局变量，static 修饰未初始的变量
rodata	只读数据，const 修饰的变量
text	存放对局部变量初始化的数据

6.2.2 存储的生命周期

存储区域	生命周期
栈	函数调用结束
data 段	整个程序结束前，都有效
bss 段	整个程序结束前，都有效
rodata	整个程序结束前，都有效
text	整个程序结束前，都有效

6.2.3 作用域

定义方式	作用域
在语句块{ 定义的变量，for（）{int a;}	本语句块内
在函数内定义的变量，fun{int a;}	本函数体内
在文件里, 其他函数外定义的 static int a;	本文件内
在文件里, 其他函数外定义的 int a;	所有文件

6.2.4 延长局部变量生命周期

```
  void foo(void)
{
static int a;
printf("%d\n", a);
a = 3;
printf("%d\n", a);
}
int main(void)
{
foo();
foo();
}
```

6.2.5 命名空间

6.2.6 链接属性

符号	作用域
globel/外部链接	全局
local/内部链接	局部
no link	不能用于链接

6.2.7 其他

7 高级指针

7.1 指针数组

```
struct STU {
    int id;
    char name[20];
    char sex;
};
```

定义	申请字节数	含义	可以赋值吗？
char *a^1	40	数组有 10 个 char * 指针	a^2 = ‘x’;*a^2=’x’
int *a^1	40	数组有 10 个 int * 指针	a^2 = 123
struct STU *a^1	40	数组有 10 个 struct STU* 指针	a^2={1,”hi”,’m’}

可不可以对上述元素进行赋值？a^2 = ‘x’;

7.2 数组指针

```
struct STU {
    int id;
    char name[20];
    char sex;
};
```

定义	申请字节数	sizeof(*p)	p+1 含义
char (*p)[1]	4	10	10
int (*p)[1]	4	40	40
struct STU (*p)[1]	4	280	280

辨析以下 p+1，分别加过多少字节？（注：加 1 表示加过 1 个元素）

```
char p[10];   p+1
char *p[10];  p+1
char (*p)[10];  p+1
```

7.2.1 数组指针与二维数组可等价转换

```
int foo(char s[][10], int n)
{
int i;
for (i = 0; i < n; i++)
printf("%s\n", s[i]);
}
  int main(void)
{
char str[3][10] = {"hello", "world", "akaedu"};
foo(str);
return 0;
}
```

7.3 二重指针

```
struct STU {
    int id;
    char name[20];
    char sex;
};
```

定义	含义	申请字节数	辨析
char **p			
int **p			
struct STU **p			

```
  int main(void)
{
char a = 'x';
char *q = &a;
char **p = &q;
*q = 'd';
**p = 'e';
printf("%c\n", a);
}

//void foo(char *argv[], int n)
void foo(char **argv, int n)
{
int i;
for (i = 0; i  < n; i++)
printf("%s\n", argv[i]);
}
  int main(void)
{
char *buf[3] = {"hello", "world", "aka"};
foo(buf, 3);
}
```

7.3.1 二重指针和数组指针的关系

命令行传参

```
  int main(int argc, char *argv[])
{
int i;
printf("argc = %d\n", argc);
for (i = 0; i < argc; i++)
printf("argv[%d]=%s\n", i, argv[i]);
}

  int main(int argc, char *argv[])
```

```
{
if (argc < 3) {
printf("./a.out num str\n");
return -1;
}
int n = atoi(argv[1]);
while (n--)
printf("%s\n", argv[2]);
return 0;
}
```

7.3.2 二重指针作为函数的参数

7.4 函数指针

7.4.1 基本定义

函数指针就是用来保存函数地址的变量。

```
 void foo(void)
{
    printf("helloworld\n");
}
int p_str(char *str, int n)
{
     if (n < 0)
          return -1;
     while (n--)
        printf("%s\n", str);
     return 0;
}
char *maxlen_str(char *str[], int n) //char *(*p)(char **, int);
{
    int maxlen = 0, i;
    char *maxstr = NULL;
    if (str == NULL)
          return NULL;
    for (i = 0; i < n; i++) {
       if (str[i] == NULL)
              continue;
       if (maxlen < strlen(str[i])) {
```

```
            maxlen = strlen(str[i]);
            maxstr = str[i];
       }
    return maxstr;
}
int main(void)
{
    void (*p)(void);
    int (*q)(char *, int);
    char *(*m)(char **, int) = maxlen_str;
    char *s[5] = {"hello", NULL, "world", "aa", "abcdefg"};
    printf("%s\n", m(s, 5));
    //q = p_str;
    //q("hello", 3);


    //p = foo;
 //   p = foo();
    //*p();
    // p();
    //foo();
    return 0;
}
```

7.4.2 函数指针作为函数的参数

```
int p_str(char *str, int n)
 {
      if (n < 0)
           return -1;
      while (n--)
         printf("%s\n", str);
      return 0;
 }
int p_str_cat(char *str, int n)
{
      if (n < 0)
           return -1;
      while (n--)
         printf("akakedu %s\n", str);
```

```
      return 0;
}
void show_str(char *str, int n, int (*fun)(char *, int))
{
     fun(str, n);
}
int main(void)
{
   //show_str("abc", 5, p_str);
   show_str("abc", 5, p_str_cat);
}
```

练习：动物走秀练习

```
  #include <stdio.h>
#include <string.h>
struct animal {
    char name[20];
    void (*say_what)(void);
    int level;
};

typedef struct animal ANIMAL;

void cow_say(void)
{
    printf("men men\n");
}
void dog_say(void)
{
    printf("wang wang\n");
}
void man_say(void)
{
    printf("OH YEAH\n");
}
void init_animal(char *name, void (*say)(void), int level, ANIMAL *p)
{
    strcpy(p->name, name);
    p->say_what = say;
```

```
    p->level = level;
}
int level_cmp(ANIMAL a, ANIMAL b)
{
    if (a.level < b.level)
        return 1;
    else if (a.level == b.level)
        return 0;
    else
        return -1;
}
int name_cmp(ANIMAL a, ANIMAL b)
{
    return strcmp(a.name, b.name);
}
void swap(ANIMAL *a, ANIMAL *b)
{
    ANIMAL tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
void show_animal(ANIMAL *a, int n)
{
    int i;
    for (i = 0; i < n; i++) {
        printf("%s\tsay\t", a[i].name);
        a[i].say_what();
    }
}
void sort_animal(ANIMAL *a, int n, int (*cmp)(ANIMAL, ANIMAL))
{
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n-i-1; j++)
            if (cmp(a[j], a[j+1]) > 0)
                swap(&a[j], &a[j+1]);
}
int main(void)
{
```

```
    ANIMAL a[10];

    init_animal("cow", cow_say, 2, &a[0]);
    init_animal("dog", dog_say, 1, &a[1]);
    init_animal("man", man_say, 3, &a[2]);
    sort_animal(a, 3, level_cmp);
    show_animal(a, 3);
    printf("**************\n");
    sort_animal(a, 3, name_cmp);
    show_animal(a, 3);
    return 0;
}
```

7.4.3 泛性函数设计

设计一个冒泡排序，可以排任意数据类型的数组，当用户调用此冒泡排序时，只需要提供一个比较函数即可。泛性函数设计时，通常借助 void 或 void *。

```
#include <stdio.h>
#include <string.h>

void bsort(void *a, int nmemb, int nsize, int compar(void *, void *))
{
    int i, j;
    char tmp[nsize];
    for (i = 0; i < nmemb; i++)
        for (j = 0; j < nmemb-i-1; j++)
            if (compar((void *)((char *)a+j*nsize),
                        (void *)((char *)a+(j+1)*nsize)) > 0) {
           memcpy((void *)(tmp), (void *)((char *)a+j*nsize),nsize);
           memcpy((void *)((char *)a+j*nsize), (void *)((char *)a+(j+1)*nsize), nsize);
           memcpy((void *)((char *)a+(j+1)*nsize), (void*)tmp, nsize);
            }
}
int int_cmp(void * a, void * b)
{
    int m = *((int*)a);
    int n = *((int*)b);
    if (m > n)
```

```
        return 1;
    else if (m == n)
        return 0;
    else
        return -1;
}
void show_int(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d\n", a[i]);
}
void show_str(char *b[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%s\n", b[i]);
}
int str_cmp(void *a, void *b)
{
    //char *m = (char *)(*((int *)a));
    //char *n = (char *)(*((int *)b));
    char *m = *((char **)a);
    char *n = *((char **)b);
    return strcmp(m, n);
}
int main(void)
{
    int a[5] = {33,1,5,9,2};


    char *b[5] = {"abc", "world", "aka", "hello", "good"};
    bsort((void*)a, 5, sizeof(a[0]), int_cmp);
    show_int(a, 5);
    bsort((void*)b, 5, sizeof(b[0]), str_cmp);
    show_str(b, 5);
    return 0;
}
```

7.4.4 函数指针作为函数的返回值

7.4.5 函数指针作为函数的参数（回调函数）

7.4.6 函数指针数组

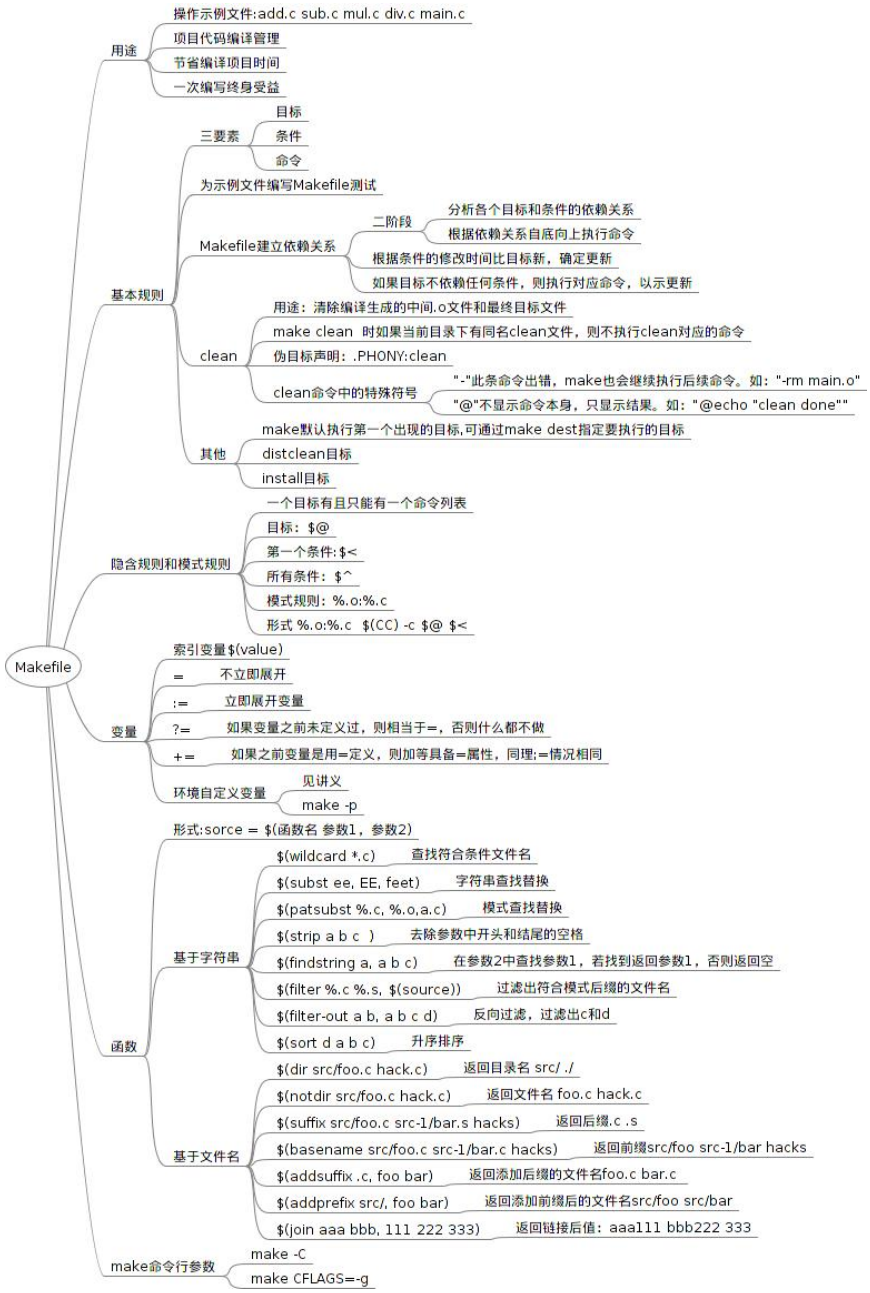
```
#include <stdio.h>
#include <string.h>
void fun_cat(char *s1, char *s2)
{
    printf("%s\n", strcat(s1, s2));
}
void fun_cmp(char *s1, char *s2)
{
    printf("%d\n", strcmp(s1, s2));
}
void fun_cpy(char *s1, char *s2)
{
    printf("%s\n", strcpy(s1, s2));
}
/*
typedef void (*FUN)(char *, char *);
FUN p[3];
*/
/*
   练习：定义以下函数的函数指针数组?
   char *foo(int *n, char **m)
   解答：typedef char *(*FUN)(int *, char **)
         FUN p[3];
   练习：如果 foo(int n) 的函数返回值是一个函数指针
   char (*)(char *, int) 类型,foo 函数如何定义?
   解答：typedef char (*FUN)(char *, int)
         FUN foo(int n)
   练习：上题 foo 函数的函数指针数组如何定义?
   解答：FUN (*p[3])(int);
 */
int main(int argc, char *argv[])
{
    void (*p[3])(char *, char *) = {fun_cat, fun_cmp, fun_cpy};
    char s1[100] = "hello";
    char s2[100] = "world";
```

```
    p[0](s1, s2);
    p[1](s1, s2);
    p[2](s1, s2);
    return 0;
}
```

7.4.7 可变参函数 printf 实现

8 预处理课件

9 Makefile 课件



10 文件操作

10.1 刷新终端文件缓冲区

1. ‘’
2. fflush（文件流指针）
3. 缓冲区满，默认 8192Bytes
4. 程序运行结束时，调用 exit 函数

10.2 FILE 结构体定义

查看 stdio.h 和 libio.h

```
/* Standard streams.  */
extern struct _IO_FILE *stdin;      /* Standard input stream.  */
extern struct _IO_FILE *stdout;     /* Standard output stream.  */
extern struct _IO_FILE *stderr;    /* Standard error output stream.  */
```

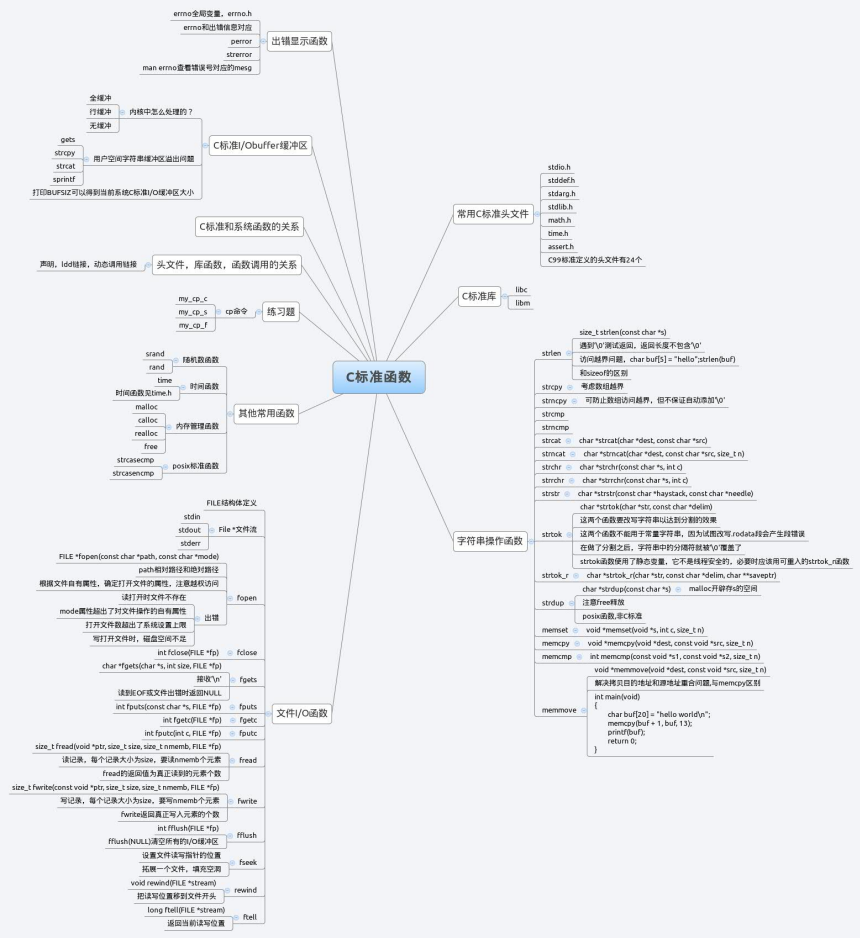
10.3 查看 errno 错误号的头文件

请大家查看此文件，加深印象

```
/usr/include/asm-generic/errno-base.h
```

errno 是全局变量，只记录最近一次函数调用时的出错原因，如果函数调用成功，不会修改 errno 的值

10.4 文件操作课件



10.5 文件操作练习

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct STU {
    int id;
    char name[20];
    char sex;
};
int read_file(FILE *fp, struct STU *a, int n)
```

```
{
    int i;
    char buf[100];
    for (i = 0; i < n; i++) {
        fgets(buf, sizeof(buf), fp);
        a[i].id = atoi(strtok(buf, "\n"));

        fgets(buf, sizeof(buf), fp);
        strcpy(a[i].name, strtok(buf, "\n"));

        fgets(buf, sizeof(buf), fp);
        a[i].sex = buf[0];
    }
    return i;
}
int cnt_stu(FILE *fp)
{
    int n = 0;
    char buf[100];
    while (fgets(buf, sizeof(buf), fp))
        n++;
    return n/3;
}
void show_stu(struct STU *a, int n)
{
    int i;
    for (i = 0; i < n; ++i)
        //printf("%d\t%s\t%c\n", a[i].id, a[i].name, a[i].sex);
     fprintf(stdout, "%d\t%s\t%c\n", a[i].id, a[i].name, a[i].sex);


}
void swap(struct STU *m, struct STU *n)
{
    struct STU tmp;
    tmp = *m;
    *m = *n;
    *n = tmp;
}
void sort_stu(struct STU *a, int n)
```

```
{
    int i, j;
    for (i = 0; i < n; ++i)
        for (j = 0; j < n-i-1; ++j)
            if (a[j].id > a[j+1].id)
                swap(&a[j], &a[j+1]);
}
int write_file_bin(FILE *fp, struct STU *a, int n)
{
    int i;
    //how many student num
    fwrite(&n, sizeof(int), 1, fp);
    for (i = 0; i < n; ++i)
        fwrite(&a[i], sizeof(struct STU), 1, fp);
    return i;
}
int read_file_bin(FILE *fp, struct STU *a, int n)
{
    int i;
    for (i = 0; i < n; ++i)
        fread(&a[i], sizeof(struct STU), 1, fp);
    return i;
}
int read_file_bin_head(FILE *fp)
{
    int n = 0;
    fread(&n, sizeof(int), 1, fp);
    return n;
}
int main(int argc, char *argv[])
{
    if (argc < 2) {
        printf("./a.out stufile stufile_bin\n"
                "or ./a.out stufile_bin\n");
        exit(-1);
    }
    int n;
    if (strcmp(argv[1], "stufile") == 0) {
        FILE *fp = fopen(argv[1], "r");
        if (fp == NULL) {
```

```
            perror(argv[1]);
            exit(-1);
        }
        FILE *fp_bin = fopen(argv[2], "w");
        if (fp_bin == NULL) {
            perror(argv[2]);
            exit(-1);
        }
        n = cnt_stu(fp);
#ifdef DEBUG
        printf("n = %d\n", n);
#endif
        struct STU a[n];
        rewind(fp);
        read_file(fp, a, n);
        sort_stu(a, n);
        show_stu(a, n);
        write_file_bin(fp_bin, a, n);
        fclose(fp);
        fclose(fp_bin);
    }
    else if (strcmp(argv[1], "stufile_bin") == 0) {
        FILE *fp = fopen(argv[1], "r");
        int n = read_file_bin_head(fp);
        struct STU a[n];
        read_file_bin(fp, a, n);
        show_stu(a, n);
        fclose(fp);
    }

    return 0;
}
```

11 C 编程知识树

11.1 数据类型

11.1.1 基础数据类型

类型	size
char	1
unsigned char	1
short	2
unsigned short	2
int	4
unsigned int	4
long	4/8
unsigned long	4/8
float	4
double	8
long long	8

11.1.2 类型转换

- 隐式类型转换
  - 类型提升
  - 类型截断
- 显式类型转换

11.2 循环控制语句

控制语句	
if	if 语句里只有两种值真/假
if . . . else	
if . . . else if	
if . . . if . . .	
switch () {case . . . break;}	等价于 if . . . else if
循环语句	
while()	语句 1，2，3 分别出现在什么地方
for()	continue 时和 while 循环区别
do{while()	宏定义时常使用，主要是因为;

11.3 数组

11.3.1 数组定义

相同数据类型的集合

11.3.2 一维数组

- 数组存储布局
  连续的线性的开辟存储空间，首元素位于数组低地址
- 数组下标的含义
  从数组首地址开始偏移元素的个数

11.3.3 二维数组

- 数组存储布局
  连续的线性的开辟存储空间，首元素位于数组低地址, 类似于一维数组
- 数组下标的含义
  行和列，int a[3, 4]; a[5, 3]含义

11.4 函数接口设计

11.4.1 为什么要设计函数

- 代码和功能可以复用
- 代码结构清晰简洁
- 分工分模块开发
- 调试修改方便
- 调用灵活
- 可以把函数封装成库，方便的提供给第三方使用

[3]DEFINITION NOT FOUND: 3
[4]DEFINITION NOT FOUND: 5
[5]DEFINITION NOT FOUND: 2

11.4.2 函数定义 4 要素

要素	用法
函数名	英文单词描述函数功能
参数	调用函数时提供哪些参数
返回值	一个函数只能返回一个值
函数体	大括号内{ 函数实现

11.4.3 函数声明 3 要素

11.4.4 函数声明和函数定义的关系

- 函数声明可否有多份
- 函数定义可有多份
- 什么时候用声明，什么时候用定义，为什么要用函数的声明

11.5 字符串数组和字符串操作的系列函数

11.5.1 字符串结尾标志

11.5.2 字符串操作函数

函数	功能
char *strstr(const char *haystack, const char *needle)	
$size_t$ strlen(const char *s)	
char *strcpy(char *dest, const char *src)	
strncpy	
strcmp	
strncmp	
strcat	
strncat	
char *strtok(char *str, const char *delim)	
char *$strtok_r$(char *str, const char *delim, char **saveptr)	
strchr	
sizeof(宏函数)	
atoi	只包含数字

11.6 结构体枚举联合

11.6.1 结构体定义

不同类型的元素的集合

11.6.2 结构体存储空间

```
struct STU {
   int id;
   char name[20];
   char sex;
}a;
```

- 考虑 4 字节对齐
- 首元素 id 位于结构体的低地址
- 成员索引符，a.id

11.6.3 typedef 类型别名

```
typedef struct STU STUDENT;
typedef struct STU * STUDENT_P;
STUDENT s;  sizeof(s)= ?
STUDENT_P q;  sizeof(q)=?  sizeof(*q)=?
```

11.6.4 联合（共用体）

```
  union ITEM {
    char ch;
    int id;
    struct STU s;
}b;
sizeof(b)= ?
```

索引联合成员

```
b.s.sex
```

11.6.5 枚举

类似于宏定义

```
enum ITEM {AA, BB=5, CC};
```

11.7 位操作

11.7.1 位操作运算符

运算符号	
&	
或	
~	
»	分为算术右移最高位填充符号位，逻辑右移最高位填 0
«	最高位丢弃，低位填充 0
»=	
«=	

11.7.2 计算机数的表示

编码
源码
反码
补码

11.8 预处理

- 头文件展开
- 宏替换
- 宏函数，宏函数里参数用小括号括起来
- 过滤注释，空白符
- 条件预处理
  - #ifdef
  - #ifndef
  - #endif
  - #elseif
- gcc -E file.c -o file.i

11.9 Makefile

- 目标 -顶格后面跟冒号
- 依赖（条件）-紧跟在目标冒号后面，如有多个依赖，用空格分割
- 命令 -开头有一个 Tab 制表符

11.9.1 Makefile 两阶段工作

- 第一阶段，建立目标和依赖之间的关系树
- 第二阶段，自底向上去执行命令

11.9.2 Makefile 变量

变量	
$@	目标
$<	第一依赖
$^	所有依赖
CPPFLAGS	-I./ 预处理器参数
CFLAGS	-c -g 编译器参数
LDFLAGS	-L./ -lm 链接器参数

11.9.3 Makefile 函数

11.10 链接 -库的制作

`a.c b.c c.c`
制作 `libabc.a` 和 `libabc.so`

11.10.1 静态库

```
gcc a.c b.c c.c -c
ar rs libabc.a a.o b.o c.o
```

- 主程序在链接静态库时，静态库代码实现会链入到主程序中
- 主程序代码体积增大
- 主程序可以放到不同的电脑上运行

11.10.2 共享库

名字	
real name	库真正版本号，包含主次版本号
so name	只记录库的主版本号，用于加载器加载阶段，是 real name 的符号链接
link name	不包含库的版本号，用于链接器链接阶段，是 real name 的符号链接

```
gcc a.c b.c c.c -c -fPIC
gcc -shared -Wl,-soname,libabc.so.1 -o libabc.so.1.10 a.o b.o c.o
sudo ldconfig        -> 生成 so name
ln -s libabc.so libabc.so.1.10        -> 生成 link name
```

11.10.3 共享库运行时加载路径

- 加载库临时路径 export $LD_{LIBRARYPATH}$=$$LD_{LIBRARYPATH}$:/newpath
- 把库拷贝到系统默认库加载路径，如：/lib ， /usr/lib
- 把库所在路径地址添加到/etc/ld.so.conf 文件中, 然后 sudo ldconfig 更新生效

11.11 汇编

11.11.1 ELF 文件格式

段名	存储
text	代码
rodata	只读数据
data	已初始全局变量，static 修饰的已初始的变量
bss	未初始化全局变量，static 修饰的未初始化的变量

- ELF 三种文件
  - 可重新定位
  - 可执行
  - 共享库

11.11.2 X86 寄存器

寄存器	
eax	保存函数调用的返回值
ebx	
ecx	
edx	
edi	
esi	
eip	程序计数器
ebp	桢指针，向上索引实参，向下索引局部变量
esp	栈指针
eflags	保存程序的运行状态（符号位，零标志位，进位，溢出位）

- 溢出：两个数相加，最高位进位和次高位进位不同则产生溢出，如相同，则没有溢出。

11.11.3 x86 汇编语法

- 见教材
- gcc a.c -g -o app
- objdump -dSsx app > file

11.11.4 函数调用栈桢创建释放的过程

- 函数传参，从右到左依次压栈
- call 命令的含义
- 进入到 foo 函数后 push %ebp 保存上一级函数的桢指针，mov %esp, %ebp 创建 foo 函数的桢指针
- 函数返回时 leave，是函数一开始两条指令的逆操作
- ret 指令是 call 指令的逆操作
- 函数的返回值放在了 eax 寄存器

11.12 指针

11.12.1 指针存储空间 -4Byte

11.12.2 指针是保存地址的变量

11.12.3 指针类型

指针类型	函数传参等价类型	sizeof(p)	sizeof(*p)
char *p	char s[]	4	1
int *p	int s[]	4	4
struct STU *p	struct STU s[]	4	28
char *p[1]	char **s	40	4
char **p	char *s[]	4	4
char (*p)[1]	char s[4, 1]	4	10
int (*p)(char *,int)	int foo(char *a,int b)	4	(*p)();
int (*p[1])(char *,int)		40	
void foo(int (*p)(char *))			
函数指针作为函数的返回值			

11.13 文件 I/O

11.13.1 FILE 结构体内容

11.13.2 文件 I/O 函数

函数	
fopen	mode:r w r+ w+ a a+
fclose	释放 buffer
fgetc	基于字符读，文件末尾返回 EOF
fputc	
fgets	基于行读，文件末尾返回 NULL
fputs	
fread	基于记录读，读成功返回记录个数，文件末尾返回 0
fwrite	
fprintf	格式化输出到指定文件，输出整型时为可见字符，区分 fwrite
fseek	设置读写指针位置，$SEEK_{SET}$ $SEEK_{END}$ $SEEK_{CUR}$
fflush	刷新文件指针缓冲区（8192）里的内容
rewind	把读写指针移到文件开头
ftail	返回读写指针位置

11.13.3 errno 全局变量 perror

11.14 数据结构

11.14.1 链表

- 动态内存申请

函数
malloc
free
calloc
reclloc

  - 栈的存储空间较小，容易耗尽
  - 堆的存储空间大，不容易耗尽，定义大存储变量时，优先在堆上开辟

- 单链表
  - 头指针 size 为 4
  - 头节点
- 双链表
  - 哨兵节点（头节点，尾节点）
  - 头指针尾指针
- 循环链表

11.14.2 排序

排序
冒泡
选择
插入
归并
快速
二分查找

11.15 函数接口设计训练

- 统计一个文件里单词个数
- 统计一个字符串中某个字符出现的次数
- 把一个字符串转序（逆序）
- 根据 (+-*/) 等字符，返回调用哪个运算函数，如 int add(int a, int b)
- 查找一个 int 数组里, 最大的数是多少
- 约瑟夫环，根据用户指定的 n(第一个人开始)m(步长)，从一个链表中找到最后活着的一个人
- 在一个字符串里查找子串 str，找到后替换成字符串 word
- 对一个字符串指针数组，按字典序进行排序

11.16 约瑟夫环问题

- 手动创建 persion 文件，里面录入人的信息，姓名，年龄，性别，爱好
- 用循环链表找到最后一个活着的人
- 打开 persion 文件，读出每一个人的信息，构成一个节点，插入到双向链表里
- 可以指定按姓名或年龄排序
- 友善的用户界面，用户可以指定从第 n 个人开始，步长为 m
- 每次删除节点（被杀死的人），按顺序记录在 delete.log 文件里
- 游戏有重新开始和退出选项