Powered by Easy Group

Easy Edu

Cmput201
期末复习

导师：祝航

basic shell commands

command	example	description
ls	ls <tgt>	列举当前<目标>文件夹的内容
cd	cd /home	进入目标文件夹
mkdir	mkdir assgn1	创建目标文件夹
mv	mv src dst	移动目标文件
cat	cat file.txt	读取目标文件，在terminal中展示内容
gcc		调用gcc编译器
cp	cp src dst	复制目标文件
rm	rm file.txt	删除目标文件
…		

Data types

Data Type	Memory (bytes)	Range	Format Specifier
int	4	-2,147,483,648 to 2,147,483,647	%d
char	1	-128 to 127 *	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	2.3E-308 to 1.7E+308	%lf
Bool	1	0/1	n/a

binary representation

1 byte == 8 bit

int为32bit： 1位sign bit和31位value bit
char为8bit： 1位sign bit和7位value bit

float为32bit： 1位sign bit， 8位exponent bit， 23位mantissa bit
double为64bit： 1位sign bit， 11位exponent bit， 52位mantissa bit

input/output

printf: 一个包含在stdio.h里的基本input method，用于向stdin/terminal screen显示格式化输出。其用法为：

printf（”<format string>”, expr1, expr2…）

例：printf（”I love %s %d\n”, “cmput”, 201 ）

Printf的format string有如下规则：

%<flag><width><.prec>d

flag: 可省，为‘-’时右对齐，为‘+’时正数显示正号

width:可省，表示输出内容的最小长度，包括小数点正负号，默认用空格补位，如有leading zero则用‘0’补位

.prec: 可省，表示输出长度的精度，对于整数和string表示具体保留长度（正负号不算一位），对于float和double表示小数点后保留的位数

input/output

scanf: 一个包含在stdio.h里的基本output method，用于从stdout/keyboard读取符合格式的输出。其用法为：

scanf（”<format string>”, &expr1, &expr2…）

例：scanf（”%d/%d/%d”, &day, &month, &year ）

Scanf的format string有如下规则：

%<*><width>d

*: 可省，有*时表示跳过当前匹配，不用分配变量来存储，如例 scanf(“%*d, %d”, &sec), 只会将第二个匹配到的整数写入sec中

width:可省，表示可以匹配的最大长度，包括小数点和正负号

Scanf的返回值是成功发生匹配并写入的变量个数，如果一个scanf要match三个整数但只有两个成功匹配并存储，scanf的返回值为2

UofA Powered by Easy Group Easy Edu

Expressions

数学运算符号		
符号	表达式	值
+，-（正负号）	+5	5
*，/，%	5%3	2
+，-（运算符）	1-3	-1

赋值符号		
符号	表达式	值
=	c = 5	5
+=，-=	c += 6	11
*=，/=	c /= 4	2

？：i+=j+=k如何理解

关系符号		
符号	表达式	值
<, <=, >, >=	3<=5	True
==	1==2	False

关系符号		
符号	表达式	值
&&（逻辑与/and）	True && False	False
||（逻辑或/or）	True || False	True
！（逻辑非/not）	!True	False

Expressions

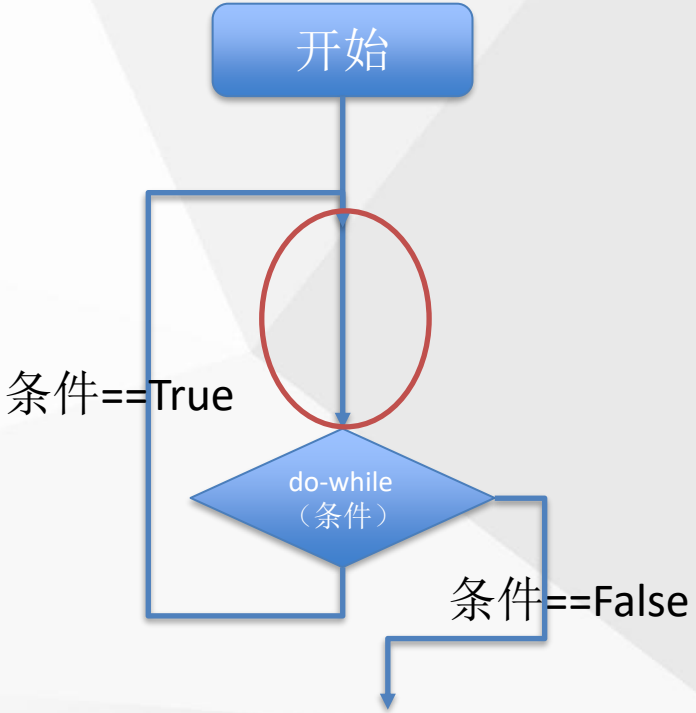
逗号是一个特别的operator，它的特性是，前面的值会被忽略，后面的值作为整个expression的值
比如（false,true） == true
int i = (0, 7) //i ==7
虽然前面的值忽略了，但是它的操作确实进行了
比如（i++, i++）
这里i++进行了两次
逗号也可以连用，如int i = (0， 7， 11) //i ==11

loops & selection statements

留意判断条件，以免越界，不要用goto。

右图do while loop画圈位置是loop body。再未经过循环条件判断之前一定会执行一次，谨慎使用

Switch statement的每个case结束之后一定要加break！



#include, #define

以上是两个常用的preprocessor command，作用于程序的compilation的过程中。

#include <*lib-name*> 用于引入标准库以及外源程序，类似于python的import。常见的有#include <stdio.h>, #include <stdlib.h>

#define *NAME* *<VALUE>* 用于定义一个替换组合。在该文件下的c程序中，所有的*NAME* 会被替换为*VALUE*。常见的有#define PI 3.14,
#define MAX_LEN 256

arrrays

1. Array的长度需要在定义时已知。
2. Array的整体赋值只能发生在定义的同时
3. 如果整体赋值长度不足，余下的每一个元素为0
4. 精确赋值时会改变当前index*
5. Array的各个元素可以通过index来随机访问，常见的array遍历：

```
for(int i=0; i<10;i++){
        ary[i]= i;}
```

6. Array的长度可以用sizeof(ary)/sizeof(ary[0])来获取，对多维数组同样有效，但这个方法对于从function以外传进来的array无效。
7. String是每一位都是字符char，并且以‘\0’结尾的array，在string.h中有自己对应的各种function，比如计算长度strlen，复制内容strcpy，比较strcmp

Function用法

C functions 用法	语法
function definition	Return_type function_name (arguments list) { Body of function; }
function call	function_name (arguments list);
function declaration/prototype	return_type function_name (argument list); /*此时argument list中argument name可省，只要type即可*/

Function特殊点

exit(): 是一个system call，作用是无论在什么位置，终止当前的进程，换句话说就是终止main。

右图的结果并不会print “not yet”， 因为over（）中的exit已经从main的外部结束了main的进程

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

void over(){
    exit(0);
    return;
}

int main(){


    over();
    printf("not yet\n");
    return 0;
}
```

main function, argc and argv

每个c程序都要有一个main function作为主程序，作为程序入口标志着程序运行的起点。主程序的返回值通常以0作为正常结束的标志，必不可少。

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

    //some code


    return 0;
}
```

```
>>>  ./area    5
```

argc == 2, argv[0]==“./area”, argv[1]==“5”,
注意这个5是string/char形式的5，不是整数的5

argc：argument count

计数整个命令行共有多少个以空格分段的变量。其中最开始的变量一定是可执行文件的文件名。

argv：argument value

以一个array of string的形式将每个变量存起来。argv[i]一定是一个string，如果想用数字的值，要将argv[i] 通过atoi或者atof转化成int或float再用于计算

Pointer

Pointer是指代地址的变量，地址是指内存中byte所对应的位置，地址在内存中顺序排列。其本质是数字，所以可以进行运算。

Pointer可以指向任何type的变量（包括pointer本身），因为他们都是存在内存之中。

关于pointer有两个非常重要的符号：*和&

&：根据本体找对应的内存地址

*：根据内存地址找地址上存的值

```
int i;
int *p;
i = 5;
p = &i;
(*p)++;

//i==*p==6
```

Pointer

*和&的作用相反，同时用会抵消掉

```
***&&&a == a
```

Pointer未赋值之前不要用，否则会出现segment fault

```
int *p;
(*p)+=12;              //错！
```

Pointer as parameter

之前我们知道，当一个变量作为参数传入一个function时，实际上是function复制了该变量的值，然后对复制体进行function中的操作，所以function中无法直接改变作为参数的外部变量的值(array除外)。

倘若我想让他修改呢？
方法一，利用return值，老办法，不多说。
方法二，利用pointer

```c
void myfunc1(int i){
    i = 0;
    printf("%d\n", i);
    i++;
    return;
}
```

```c
int main()
{
    int x = 10;
    myfunc1();
    //x==10

    return 0;
}
```

Pass by value

Pointer as parameter

如果我们将pointer作为参数传入function，相当于我们告诉function了一条线索，一个地址，虽然在传入过程中发生了赋值，但是无伤大雅，因为我们需要的是pointer的值，也就是地址的值。之后function中可以通过“*”操作进而进行读写指令

```
void myfunc2(int *p){
    *p = 0;
    printf("%d\n", *p);
    (*p)++;
    return;
}

int main()
{
    int x = 10;
    myfunc2();
    //x==0

    return 0;
}
```

Pass by reference

Pointer as return value

```
int* swap_if_larger(int *a, int *b){
  int temp;
  if(*a > *b){
    temp = *a;
    *a = *b;
    *b = temp;
  }
  return b; //returns the larger number
}
```

Pointer作为一个type，也可以作为一个function的返回值出现，但是要十分小心。因为scope和duration的存在，指针另一头的实体可能会消失掉。

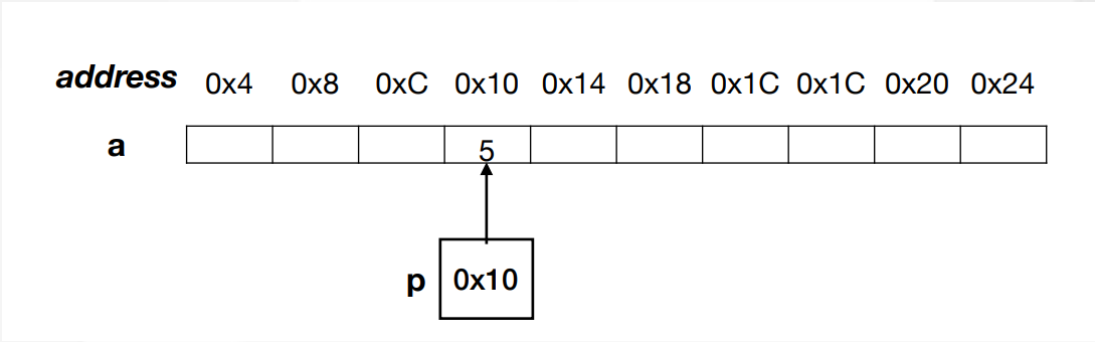
Pointer作为返回值的原则：

·对于function中的local变量只允许指向static变量

·可以是以pointer形式作为参数传入function的变量的相关指针

Pointer and arrays

Array与pointer关系密切，实际上array是由pointer衍生而来，继承了pointer在内存中连续存储的特性，并加入了一些更加完备的信息。



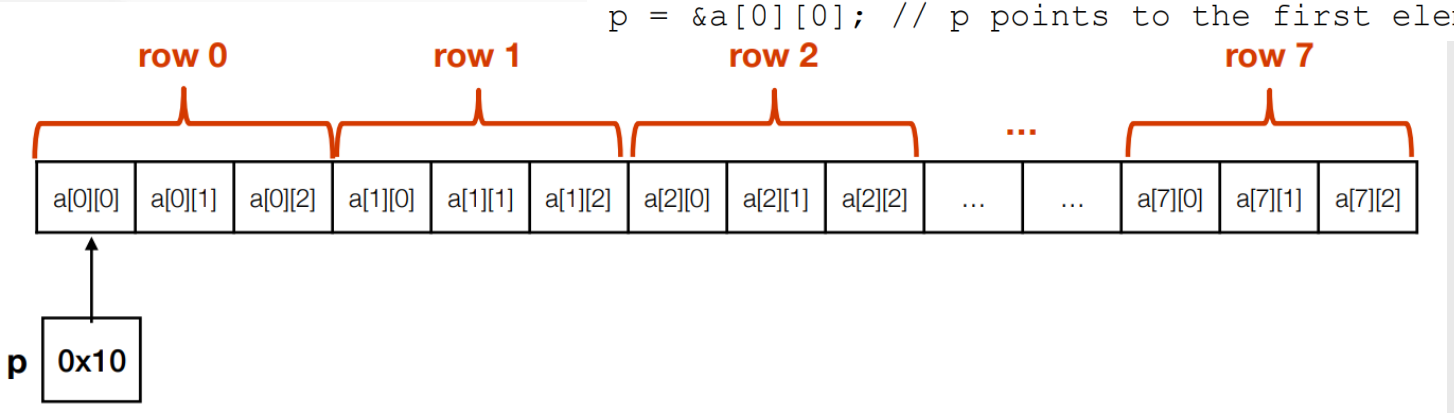
Pointer and arrays

```
11  int a[10];
12  int *p;
13
14  p = &a[5];
15  *p = 7;  //=> a[5]=7
16  p = a; //=> p=&a[0]
17  p++;  //=> p=&a[1]
```

上图中，arrayname “a”也可以当作pointer来使用，数学关系和p相同，然而a不能改变本身的值，会报错，因为根据定义，a是const type。

Pointer and arrays

```
int a[8][3];
int *p, *q;
p = &a[0][0]; // p points to the first element
```



对于多维array，关系如上图所示。注意此时p或q与a的type不同，不能进行p = a的操作，因为a的type是const int **。

Scope/Duration

Scope是指变量生效的范围，通常分为block和file两种，分别对应local和global变量。

Block：指该变量自定义之后只生效于本身定义的block（{...}）中。

例：常见的在{}中定义的无前缀变量，function的parameters

File：指该变量自定义之后生效于所在的文件中所有的block中。

例：global variable

Scope/Duration

Duration是指变量的值储存周期，通常分为automatic，dynamic和static。

automatic：指变量的值在其所在scope执行结束后自动重置为默认值（如果存在）

例for(int i=0;;){}

Dynamic：指通过memory allocation得到的内存地址，需要在恰当的时候手动移除

Static：指变量的值不会随着本身scope结束而重置，一直持续到本次主程序结束

preprocessor

预处理指令指的是以#开头的指令，这些指令会在编译过程中执行，打到预期的效果。

引用库指令　#include ：区分“ ”和<>
宏定义　　#define ：简单的文字替换，可带变量，#和##
条件指令　　#if, #ifdef 等

#：convert to string literal
##：attach to other token

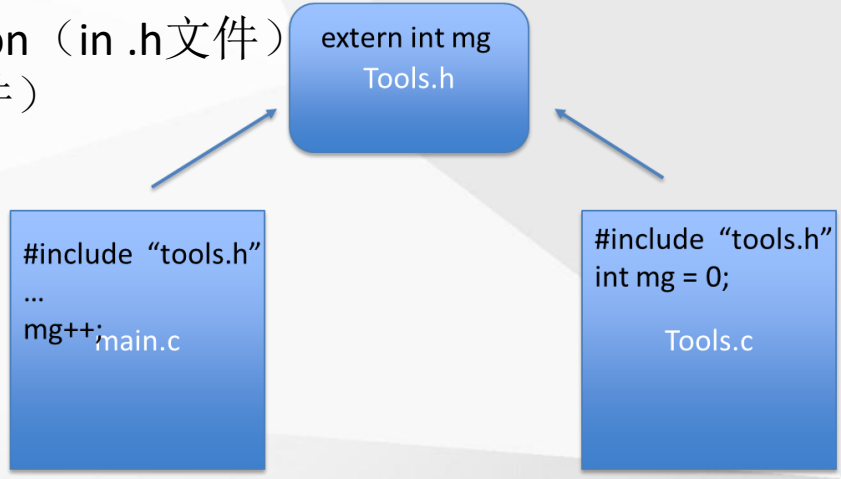
基本规则：
1.所有预处理指令必须以#开头
2.空格敏感，但空格数量不敏感
3.所有指令除非特殊用‘\’换行，否则都在本行内结束
4.可以出现在程序的任意位置

header file

Header file是指以.h后缀的文件，相当于与之同名.c文件的目录，它的作用是将.c文件中的内容要点分享给调用这些功能和内容的其他文件。

Header file中具体分享的内容有

1. Macro definition and Type definition（in .h文件）
2. Function prototype（in 同名.c文件）
3. 全局变量（in 同名.c文件）



多文件项目的建立

Makefile的格式

Target： dependency list
　　unix command

例：
All： main.c tool.c tool.h
　　gcc –o test main.c tool.c -Wall

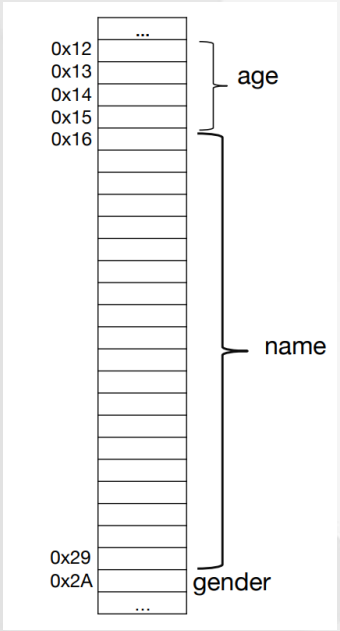
多文件项目的建立

```
All：  main.o  tool.o
       gcc –o test  main.o  tool.o –Wall

main.o:  main.c tool.h
       gcc –c main.c –Wall

tool.o:  tool.c tool.h
       gcc –c tool.c -Wall
```

struct



Struct是一种打包的数据结构，能将任何数据结构整合成一种数据类型。其子项在memory中连续排列。理论上说一个struct的size应该是所有element的size的和。实际上因为memory alignment的存在，实际size应当大于等于理论size。

其格式为：

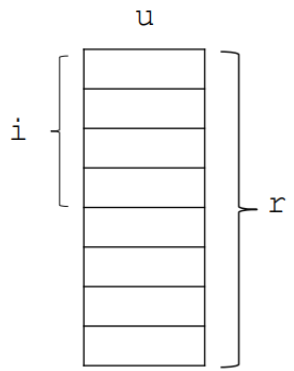
```
struct (name){
   ….//elements
} (variables)…;
```

两个括号最多只能省略一个

```
struct{
  int age;
  char name[20];
  char gender;
} person1, person2;
```

Union

```
union {
  int i;
  double r;
} u;
```



Union是一种压缩的数据结构，能将任何数据结构压缩成一种数据类型。其子项在memory中重叠排列。Union type的size取决于子项中size最大的项的值。注意：一个union type变量同一时间只能取其一个子项的值，对某个子项赋值会覆写其他子项的值。

Union的赋值规则与struct类似

Enum

Enum是一种枚举的数据结构，能将这种数据类型下的变量规定为给定的值。其本质是一个unsigned int。所以size是4 bytes。机制是将给定的选项简化为不同的数字用以替代。

```
enum suit {CLUB, DIAMONDS, HEARTS, SPADES};
typedef enum {club, diamonds, hearts, spa} Suit;
enum suit s1, s2;
Suit s1, s2;
```

上图中club/CLUB的值为0，diamonds/DIAMONDS的值为1，以此类推

本质上与#define club 0无异

Memory allocation

Memory allocation是c语言中手动为程序分配内存空间的操作，常用的function有：

- malloc: 分配，不清空
- calloc: 分配，清空
- realloc: 再分配，保留原有data，free之前的，不清空

对于所有手动分配的内存空间，在程序结束之前需要通过free来释放，否则内存一直被占用直到重启。这种特性使得手动分配的空间可以从function内return到上层程序。

Storage classes

Storage class分为四种：
auto: 自动识别，周期与scope相同，默认类
static: 静态变量，周期与main相同
extern: 外部变量，指外部（其他源文件）的全局变量，周期与main相同
register: 寄存器变量，自动识别，周期与scope相同，没有地址无法进行&操作，无法与指针关联

Type Qualifiers

Const：只读变量，不可修改

Volatile： 易变变量，值可能会发生改变，告知compiler不要优化，务必从内存读取实时值。（不考）

Restrict：只作用于pointer，表示某个pointer独家指向某变量，不会被其他pointer access

```
void sum(int * a, int * b)
{
    *a = 5;
    *b = 6;
    printf("%d", *a + *b);
    //未必是11
    return;
}
```

```
void sum(int *restrict a, int * restrict b)
{
    *a = 5;
    *b = 6;
    printf("%d", *a + *b);
    return;
}
```

input/output redirection

Stdin可以通过‘<‘ 来将一个文件的内容作为键盘输入

ex： ./hello.c < name.txt

Stdout可以通过’>’来将输出导入一个新文件

可以通过’>|’来将输出覆写到一个文件

可以通过’>>’来将输出append到一个文件结尾处

Stderr可以通过在stdout的redirect command前加’2’来将

error msg与stdout分开

./text.c <input.txt >output.txt 2>error.txt

对于text.c，以input.txt作为输入，将输出导出到output.txt，将错误信息导出到error.txt

FILE Pointers

File pointer 包含标准io数据流和customize的临时文件通道，标准数据流可以通过fscanf和fprintf加上通道名称（stdin，stdout，stderr）来访问。文件通道要先经由fopen创建file pointer，再进行访问，访问的方法同标准数据流。打开文件通道后，在退出程序之前要用fclose关闭通道

Bit-wise operator

包括：

Not (~)
And (&)
Xor (^)
Or (|)
Shift left (<<)
Shift right (>>)

Bit类题的四个步骤：

1. 提取
2. 擦除
3. 移位
4. 写回


UofA
Powered by Easy Group
Easy Edu