实验四 复杂结构实验

一 实验目的

- 理解函数调用过程中堆栈的变化情况
- 理解数组、链表在内存中的组织形式
- 理解struct和union结构数据在内存中的组织形式

二 实验内容

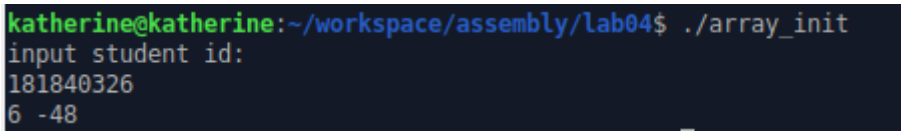
1. array_init

(1) 给出数组a、b在栈上的分布

函数g	函数f
old ebp	old ebp
%gs(14)	%gs(14)
a[9]	b[1]
a[8]	b[0]
a[7]	
a[6]	
a[5]	
a[4]	%esp
a[3]	
a[2]	
a[1]	
a[0]	

(2) 输入学号并解释输出

- 输入我的学号181840326，输出是6 和 -48



- 解释原因：

init()函数中调用fgets()函数，根据我在搜索引擎中查找的结果，C 库函数 **char *fgets(char *str, int n, FILE *stream)** 从指定的流 stream 读取一行，并把它存储在 **str** 所指向的字符串内。当读取 **(n-1)** 个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。在这里调用fgets()时n为10，所以它会读取输入流的前九个字符，也就是9位学号，又因为它以字符串形式存储在字符数组内，所以数组第10位是'\0'.

然后根据for循环的定义，a数组中存放的其实是ASCII码的差值，当temp[i]为数字是，相应的a[i]也是数字，所以a[0]-a[8]分别是9位学号的值，又因为0的ASCII码是48，'\0'的ASCII码是0，所以a[9]的值是-48.

根据第(1)问对栈的推测，b[0]和b[1]存放的地址是之前a[8]和a[9]存放的地址，又因为没有针对数组b进行初始化，所以输出的结果是学号的最后一位和a[9],即6和-48.

- 使用未初始化的程序局部变量的危害：

如果程序局部变量没有进行初始化，它在栈上的存放地址中存放的将是该地址之前存放的值，首先这使局部变量的值具有不确定性，会导致无法预测程序的结果，假如该值非常小或者非常大，甚至可能造成溢出。因此我们应该养成初始化变量的好习惯。

2. 3_d_array

(1) 地址表达式

$$addr(A[i][j][k]) = addr(A) + 4 * (k + T * (j + S * i))$$

(2) 查看寄存器的值

	%eax	%ecx	%edx
3	1	4	46
4	1	258	46
5	1	258	1
6	2	258	1
7	2	258	2
8	16	258	2
9	14	258	2
10	14	258	46956
11	14	258	46970
12	4	258	46970
13	4	258	46974
14	181840326	258	46974
15	181840326	258	46974
16	378560	258	46974

(3) 确定R S T的取值

给汇编代码加注释如下：

```
1  push   %ebp
2  mov    %esp,%ebp
3  mov    0xc(%ebp),%eax          get j
4  mov    0x8(%ebp),%ecx          get i
5  mov    %eax,%edx               j->edx
6  lea    (%edx,%edx,1),%eax      2*edx=2j->eax
7  mov    %eax,%edx               2j->edx
8  lea    0x0(,%edx,8),%eax       8*edx=16j->eax
9  sub    %edx,%eax               eax-edx=16j-2j=14j->eax
10 imul   $0xb6,%ecx,%edx         182*ecx=182*i->edx
11 add    %eax,%edx               14j+182i=14(j+13i)->edx
12 mov    0x10(%ebp),%eax         get k->eax
13 add    %eax,%edx               k+14(j+13i)->edx
14 mov    0x14(%ebp),%eax         dest->eax
15 mov    %eax,0x804a060(,%edx,4) dest->addr(A)+4(k+14(j+13i))
16 mov    $0x5c6c0,%eax           sizeof(A)=0x5c6c0=378560
17 pop    %ebp
18 ret
```

根据第15行注释得出的结果与第（1）问中的表达式对比，可以得出T=14,S=13,sizeof(A)=4RST,所以R=（378560）/(4*13*14)=520,所以A[520][13][14]

```
#define R 520
#define S 13
#define T 14
```

3. recursion

原函数如下：（见recursion.c)

```
int recursion(int x){
    if(x>2)
        return recursion(x-1)+recursion(x-2);
    else
        return 1;
}
```

检验：我将recursion.c生成的汇编代码与题目中所给的比较，发现基本相同。

4. proc

(1) 确定偏移量

偏移量	0	4	8	12
	e1.p y y[0]	e1.x y[1]	y[2]	next

(2) 补全代码

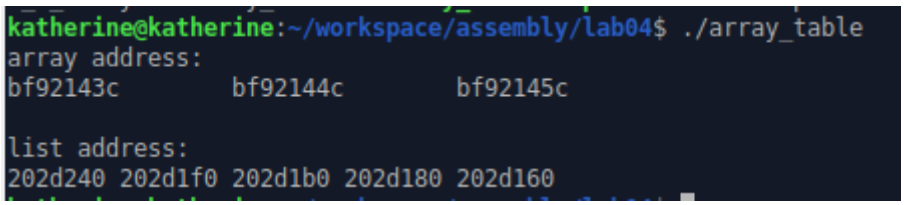
补全缺失的表达式：（见proc.c)

```
void proc(struct ele *up){
    up->next=*(up->e1.p)+up->y[2];
}
```

检验：反汇编生成的汇编代码基本相同

(3) 解释原因

程序的输出如下：



从结果可以看出，数组的地址是连续的，每一个元素的地址相差12字节，刚好是一个结构的大小；而链表的相邻地址差是等差数列，从代码可以看出构建链表的方式是从表头插入，而随着i的增大，每次构建链表后分配的int空间也增大，但即便如此地址仍是不连续的。

由于数组使用的是静态内存，所以在声明数组的时候就为数组分配空间，并且是连续的空间。而链表的空间使用malloc函数在建立链表的过程中申请的，它使用的是堆区的内存，并且由于它是边建立边分配空间，所以链表的地址不是连续的。