实验 4　复杂结构实验

姓名：林荣恩　　学号：191220060　　院系：计算机科学与技术系

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、给定如下 array_init.c 文件，使用命令 gcc -fstack-protector-all -ggdb array_init.c –o array_init 编译代码，使用命令 objdump -d array_init > array_init.s 反汇编二进制文件，分析反汇编后代码，并完成以下要求：

(1) 查看函数 g 和 f 的反汇编代码,分别给出函数 g 和 f 中数组 a,b 在栈上的分布,在下图中给出 a[0]-a[9]以及 b[0]、b[1]位置。

函数 g:

<table>
<tr><td colspan="2">old rbp</td></tr>
<tr><td colspan="2">%fs(28)</td></tr>
<tr><td>a[9]</td><td>a[8]</td></tr>
<tr><td>a[7]</td><td>a[6]</td></tr>
<tr><td>a[5]</td><td>a[4]</td></tr>
<tr><td>a[3]</td><td>a[2]</td></tr>
<tr><td>a[1]</td><td>a[0]</td></tr>
<tr><td colspan="2"></td></tr>
</table>

函数 f:

<table>
<tr><td colspan="2">old rbp</td></tr>
<tr><td colspan="2">%fs(28)</td></tr>
<tr><td>b[1]</td><td>b[0]</td></tr>
<tr><td colspan="2"></td></tr>
</table>

2) 运行程序，程序的输入为 9 位学号（此处为 191220060），观察输出。解释为什么 b[0]和 b[1]是这两个值。说明使用未初始化的程序局部变量的危害。

```
input student id :
191220060
0 -48
```

输出为“0 -48”.

b[0]=0, b[1]=-48 的原因：在对 g()与 f()的调用中，栈中 b[0]的位置与 a[8]重合，b[1]的位置与 a[9]重合。在 g()调用后，a[8]=’0’-’0’=0，其中第一个 0 来源于学号的末位；a[9]=’\0’-’0’=-48；这些值也反映在栈中的对应位置上。但是，在调用结束后，尽管栈指针发生了变化，原来栈中对应位置的值仍然保留。同时，由于程序并未为 b[]赋初值，因此在 print 被调用时，b[1]和 b[0]位置上的值继承了 a[9]与 a[8]，故在 print 中，b[0]=0,b[1]=-48.

从上例可以看出，未初始化的程序局部变量会使程序的运行情况和结果随着其它部分（如先前执行的程序）的变化而变化，是不稳定的、难以预测的，在许多情形下是有害的。

2、给定如下三维数组 A 的定义以及 store_ele 函数，其中 R,S,T 是用#define 定义的常量。又给定 3_d_array 这个可执行文件，在 3_d_array 的 main 函数中仅调用了一次 store_ele 函数，使用命令 objdump -d 3_d_array > 3_d_array.s 反汇编二进制文件，观察 store_ele 函数。

1) 将数组地址计算扩展到三维，给出 A[i][j][k]地址的表达式。(A 的定义为 int A[R][S][T], sizeof(int)=4,起始地址设为 addr(A))

与二维的情形类似，三维有符号整型数组 A[i][j][k]地址的表达式为

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

2) 使用命令 gdb ./3_d_array 启动 gdb 调试。在 store_ele 函数入口设置断点，以自己的 9 位学号为输入，运行程序。在 store_ele 函数中，单步执行，并打印出每步汇编指令执行后寄存 eax、ecx、edx 的值。上面给出了 store_ele 函数的汇编指令及其指令编号，根据自己的实验结果填写每条指令运行后的结果。

	%eax	%ecx	%edx
3	7	4	8
4	7	352	8
5	7	352	7
6	14	352	7
7	14	352	14
8	112	352	14
9	98	352	14
10	98	352	64064
11	98	352	64162
12	4	352	64162
13	4	352	64166
14	191220060	352	64166
15	191220060	352	64166
16	378560	352	64166

3) 根据以上内容确定 R、S、T 的取值。

对汇编代码进行分析可以看出，S*T=182，T=14，sizeof(A)=378560=4*S*T*R. 从中可以解出 R、S、T 的值，即

R=520，S=13，T=14.

3、函数 recursion 是一个递归调用函数。其原函数存在缺失，试根据其汇编代码确定原函数，保存为 recursion.c.

见 recursion.c.

4、给定结构体:

```
struct ele{
    union {
        struct{
            int* p;
            int x;
        }e1;
        int y[3];
```

```
    };
    struct ele *next;
};
```

1) 确定下列字节的偏移量：e1.p、e1.x、y、y[0]、y[1]、y[2].

- e1.p 偏移量为 0;
- e1.x 偏移量为 8（64 位系统中指针 (e1.p) 长度为 8 字节）;
- y 偏移量为 0;
- y[0]偏移量为 0;
- y[1]偏移量为 4;
- y[2]偏移量为 8;

2) 过程对链表进行操作，链表是以上述结构作为元素。现有 proc 函数主体的汇编码，查看汇编代码，并根据汇编代码补全 proc 函数中缺失的表达式. (不需要进行强制类型转换)

见 proc.c.

3) main 函数中声明了一数组和一链表并打印了每个元素的地址，查看地址，并解释产生原因，体会数组与链表分别使用静态内存和动态内存的差异。

```
array address:
e2da9e40        e2da9e58        e2da9e70

list address:
27f277f0        27f27770        27f27710        27f276d0
        27f276b0
```

每个元素的地址如图。

对数组，由于分配的是静态内存，故相邻两个元素的地址相差 sizeof(struct ele)=24 字节（内存对齐使一个 ele 占 24 字节，具体可参考本小实验的第一问）。

对链表，由于 malloc 下 struct ele 和 int 都对齐至 32 字节，同时 malloc 的整型变量数目由 0 开始每次增加一，故链表相邻元素（依加入顺序）分别相差 32，64，96，128 字节。