

实验 4 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、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

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

```
#define M 2
```

```
#define N 10
```

```
void init(int a[N]){
```

```
    int i;
```

```
    char temp[N];
```

```
    printf("input student id : \n");
```

```
    fgets(temp,N,stdin);
```

```
    for(i=0;i<N;i++){
```

```
        a[i]=temp[i]-'0';
```

```
    }
```

```
}
```

```
void g(){
```

```
    int a[N];
```

```
    init(a);
```

```
}
```

```
void print(int b[M]){
```

```
    int i;
```

```
    for(i=0;i<M;i++){
```

```
        printf("%d ",b[i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
void f(){
```

```
    int b[M];
```

```
    print(b);
```

```
}
```

```
int main(){
    g();
    f();
    return 0;
}
```

实验要求：

(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	函数 f
old rbp	old rbp
%fs(28)	%fs(28)

(2) 运行程序，程序的输入为 9 位学号，观察输出。请详细解释为什么 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。

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。

```
int A[R][S][T];

int store_ele(int i,int j,int k,int dest){
    A[i][j][k] = dest;
    return sizeof(A);
}
```

```
1  push    %ebp
```

```

2  mov    %esp,%ebp
3  mov     0xc(%ebp),%eax
4  mov     0x8(%ebp),%ecx
5  mov     %eax,%edx
6  lea     (%edx,%edx,1),%eax
7  mov     %eax,%edx
8  lea     0x0(%edx,8),%eax
9  sub     %edx,%eax
10 imul    $0xb6,%ecx,%edx
11 add     %eax,%edx
12 mov     0x10(%ebp),%eax
13 add     %eax,%edx
14 mov     0x14(%ebp),%eax
15 mov     %eax,0x804a060(,%edx,4)
16 mov     $0x5c6c0,%eax
17 pop     %ebp
18 ret

```

实验要求

(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)`)

(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器 `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。

(3) 根据以上内容确定 R、S、T 的取值

	%eax	%ecx	%edx
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

16			
----	--	--	--

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

```
int recursion (int x){
    if(_____)
        return ____;
    else
        return ____;
}
```

0000000000000000 <recursion>:

0:	55	push	%rbp
1:	48 89 e5	mov	%rsp,%rbp
4:	53	push	%rbx
5:	48 83 ec 18	sub	\$0x18,%rsp
9:	89 7d ec	mov	%edi,-0x14(%rbp)
c:	83 7d ec 02	cmpl	\$0x2,-0x14(%rbp)
10:	7f 07	jg	19 <recursion+0x19>
12:	b8 01 00 00 00	mov	\$0x1,%eax
17:	eb 1e	jmp	37 <recursion+0x37>
19:	8b 45 ec	mov	-0x14(%rbp),%eax
1c:	83 e8 01	sub	\$0x1,%eax
1f:	89 c7	mov	%eax,%edi
21:	e8 00 00 00 00	callq	26 <recursion+0x26>
26:	89 c3	mov	%eax,%ebx
28:	8b 45 ec	mov	-0x14(%rbp),%eax
2b:	83 e8 02	sub	\$0x2,%eax
2e:	89 c7	mov	%eax,%edi
30:	e8 00 00 00 00	callq	35 <recursion+0x35>
35:	01 d8	add	%ebx,%eax
37:	48 83 c4 18	add	\$0x18,%rsp
3b:	5b	pop	%rbx
3c:	5d	pop	%rbp
3d:	c3	retq	

4、给定以下结构定义

```
struct ele{
    union {
```

```

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

```

实验要求

(1) 确定下列字节的偏移量。

```

e1.p
e1.x
y
y[0]
y[1]
y[2]
next

```

(2) 下面的过程（省略一些表达式）是对链表进行操作，链表是以上述结构作为元素的。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，并保存为 `proc.c`。（不需要进行强制类型转换）

```

void proc(struct ele *up){

    up->_____ = *(up->_____) + up->_____;

}

```

0000000000000000 <proc>:

0:	55	push	%rbp
1:	48 89 e5	mov	%rsp,%rbp
4:	48 89 7d f8	mov	%rdi,-0x8(%rbp)
8:	48 8b 45 f8	mov	-0x8(%rbp),%rax
c:	48 8b 40 10	mov	0x10(%rax),%rax
10:	48 8b 55 f8	mov	-0x8(%rbp),%rdx
14:	48 8b 12	mov	(%rdx),%rdx
17:	8b 0a	mov	(%rdx),%ecx
19:	48 8b 55 f8	mov	-0x8(%rbp),%rdx
1d:	8b 52 08	mov	0x8(%rdx),%edx
20:	01 ca	add	%ecx,%edx
22:	89 10	mov	%edx,(%rax)
24:	90	nop	

```

25:    5d                                pop    %rbp
26:    c3                                retq

```

(3) 有以下 main 函数，该 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。

```

int main(){
    struct ele a[5];
    struct ele * head, * p;
    head=NULL;
    for(int i=0;i<5;i++){
        p=(struct ele *)malloc(sizeof(struct ele));
        p->next=NULL;
        if(head==NULL){
            head=p;
        }else{
            p->next=head;
            head=p;
        }
        for(int j=0;j<i;j++){
            malloc(sizeof(int));
        }
    }

    printf("array address:\n");
    printf("%x\t%x\t%x\n",(unsigned int)&a[0],(unsigned int)&a[1],(unsigned int)&a[2]);
    printf("\nlist address:\n");
    p=head;
    while(p!=NULL){
        printf("%x\t", (unsigned int)p);
        p=p->next;
    }
    printf("\n");
}

```

实验报告要求：

见之前实验