

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;
}
```

```
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

[illegible]

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 _____;
}

```

00000000 <recursion>:

0:	55	push	%ebp
1:	89 e5	mov	%esp,%ebp
3:	53	push	%ebx
4:	83 ec 04	sub	\$0x4,%esp
7:	83 7d 08 02	cmpl	\$0x2,0x8(%ebp)
b:	7f 07	jg	14 <recursion+0x14>
d:	b8 01 00 00 00	mov	\$0x1,%eax
12:	eb 28	jmp	3c <recursion+0x3c>
14:	8b 45 08	mov	0x8(%ebp),%eax
17:	83 e8 01	sub	\$0x1,%eax
1a:	83 ec 0c	sub	\$0xc,%esp
1d:	50	push	%eax
1e:	e8 fc ff ff ff	call	1f <recursion+0x1f>
23:	83 c4 10	add	\$0x10,%esp
26:	89 c3	mov	%eax,%ebx
28:	8b 45 08	mov	0x8(%ebp),%eax
2b:	83 e8 02	sub	\$0x2,%eax
2e:	83 ec 0c	sub	\$0xc,%esp
31:	50	push	%eax
32:	e8 fc ff ff ff	call	33 <recursion+0x33>
37:	83 c4 10	add	\$0x10,%esp
3a:	01 d8	add	%ebx,%eax
3c:	8b 5d fc	mov	-0x4(%ebp),%ebx
3f:	c9	leave	
40:	c3	ret	

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->_____;  
}
```

00000000 <proc>:

0:	55	push	%ebp
1:	89 e5	mov	%esp,%ebp
3:	8b 45 08	mov	0x8(%ebp),%eax
6:	8b 40 0c	mov	0xc(%eax),%eax
9:	8b 55 08	mov	0x8(%ebp),%edx
c:	8b 12	mov	(%edx),%edx
e:	8b 0a	mov	(%edx),%ecx
10:	8b 55 08	mov	0x8(%ebp),%edx
13:	8b 52 08	mov	0x8(%edx),%edx
16:	01 ca	add	%ecx,%edx
18:	89 10	mov	%edx,(%eax)
1a:	90	nop	
1b:	5d	pop	%ebp
1c:	c3	ret	

(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