hw06_runtime习题解答（修正&注释）

第一题

```
union var{ // 各成员共享空间
    char c[5];
    int i;
};
int main(){
   union var data;
   char *c;
   data.c[0] = '2'; // 50
   data.c[1] = '0'; // 48
   data.c[2] = '1'; // 49
   data.c[3] = '6'; // 54
   data.c[4] = '\0'; // 0
   c = (char*)&data;
   printf("%x %s\n",data.i,c);
   return 0;
}
```

```
main:
        pushl ebp%
        movl %esp, %ebp
        subl $40, %esp
        andl $-16, %esp // 按16字节对齐(空间不变或增加)，方
便一些指令的并行操作
        movl $0, %eax
        subl %eax, %esp
        movb $50, -24(%ebp)
        movb $48, -23(%ebp)
        movb $49, -22(%ebp)
        movb $54, -21(%ebp)
        movb $0, -20(%ebp)
        leal -24(%ebp), %eax
        movl %eax, -28(%ebp)
```

第一题

逆序传参.

```
union var{ // 各成员共享空间
    char c[5];
    int i;
};
int main(){
   union var data;
   char *c;
   data.c[0] = '2';
   data.c[1] = '0';
   data.c[2] = '1';
   data.c[3] = '6';
   data.c[4] = '\0';
   c = (char*)&data;
   printf("%x %s\n",data.i,c);
   return 0;
}
```

```
movl %eax, -28(%ebp)
sub $4, $esp // esp-4, 一共就是下面的addl $16
pushl -28(%ebp) // esp-4  // 参数    第3个参数.
pushl -24(%ebp) // esp-4  // 参数    第2个参数.
pushl $.LC0 // esp-4, 返回地址       第1个参数.
call printf
addl $16, %esp // 恢复栈之前状态，清除掉printf函数参数的空间
movl $0  %eax // return 0;
leave
ret
```

输出为：

```
36313032 2016
```

第二题

```
#define N 2
// #define N 11
typedef struct POINT {
    int x, y ;
    char z[ N ];
    struct POINT *next;
    } DOT;
void f(DOT p) {
     p.x = 100;
     p.y = sizeof(p);
     p.z[1] = 'A';
     f(*(p.next));
}
```

```
f:  // 当 N=2 时，生成的汇编代码片段
      pushl %ebp
      movl %esp, %ebp
      movl $100, 8(%ebp)
      movl $16, 12(%ebp)
      movb $65, 17(%ebp) // z[0]:16(%ebp)  z[1]:17(%ebp)
      movl  20(%ebp), %eax // z[2]补到了4个字节
      pushl 12(%eax)
      pushl 8(%eax)          // 逆序传递
      pushl 4(%eax)
      pushl (%eax)
      call f
      addl $16, %esp // 同前， 清除f参数的空间
      leave
      ret
```

第二题

```c
#define N 2
// #define N 11
typedef struct POINT {
    int x, y ;
    char z[ N ];
    struct POINT *next;
    } DOT;
void f(DOT p) {
     p.x = 100;
     p.y = sizeof(p);
     p.z[1] = 'A';
     f(*(p.next));
}
```

```
f: // 当 N=11 时，生成的汇编代码片段
        pushl %ebp
        movl %esp, %ebp
        pushl %edi
        pushl %esi
        movl $100, 8(%ebp)
        movl $24, 12(%ebp) // N=11时大小为4+4+11+1+4=24
        movb $65, 17(%ebp) // z[1]依然是17(%ebp)
        subl $8, %esp
        movl 28(%ebp), %eax // 相应地到了28
        subl $24, %esp
        movl %esp, %edi
        movl %eax, %esi
        cld
```

第二题

```
#define N 2
// #define N 11
typedef struct POINT {
   int x, y ;
   char z[ N ];
   struct POINT *next;
   } DOT;
void f(DOT p) {
    p.x = 100;
    p.y = sizeof(p);
    p.z[1] = 'A';
    f(*(p.next));
}
```

```
// 接cld
movl $6, %eax // 大小为6(long)
movl %eax, %ecx
rep
movsl
call f
addl $32, %esp
leal -8(%ebp), %esp // 清除f的空间
// 为什么是 -8(%ebp)：恢复esi, edi状态
popl %esi
popl %edi
leave
ret
```

☆☆

编译器在按值传递结构变量时的处理方式:

逆序的栈传递方式，数据多时采用数据传送指令。

第三题

```c
void g(int**);
int main(){
    int line[10],i;
    int *p=line;
    for (i=0;i<10;i++){
    *p=i;  g(&p);
    }
    return 0;
}
void g(int**p){
    (**p)++;  (*p)++;
}
```

type(&p): int**

```
.globl g
        .type g,@function
g:
        pushl %ebp
        movl %esp, %ebp
        movl 8(%ebp), %eax // 参数位置
        movl (eax%), %eax // 注意(%eax)和%eax      // (**p)++
        addl $1, (%eax)
        movl 8(%ebp), %eax
        addl $4, (%eax)                  // (*p)++
        leave
        ret
```

第三题

```
void g(int**);
int main(){
    int line[10],i;
    int *p=line;
    for (i=0;i<10;i++){
    *p=i;  g(&p);
    }
    return 0;
}
void g(int**p){
    (**p)++;  (*p)++;
}
```

最后line[10]为：

1 2 3 ... 10

改成2条

```
movl $9, %eax
cmpl -60(%ebp), %eax
```

```
main:
        pushl %ebp
        movl %esp, %ebp
        subl $72, %esp
        andl $-16, %esp
        movl $0, %eax
        subl %eax, %esp
        leal -56(%ebp), %eax
        movl %eax, -64(%ebp)
        movl $0, -60(%ebp)
.L2
        cmpl -60(%ebp), $9
        jle .L5
        jmp .L3
.L5:
        movl -64(%ebp), %edx
        movl -60(%ebp), %eax
        movl %eax, (%edx)
        subl $12, %esp // esp-12
        leal -64(%ebp), %eax
        pushl %eax //esp-4
        call g
        addl $16, %esp // 12 + 4
        leal -60(%ebp), %eax
        incl (%eax)
        jmp .L2 // 回去比较
.L3:
        movl $0, %eax
        leave
        ret
```

i>=10则跳到.L3结束循环

☆☆

第四题

```
#include <stdio.h>
int main(){
    int a=0, b = 0;
    { int a = 1; }
    { int b = 2;
        { int a = 3; }
    }
    return 0;
}
```

```
main: // 存储分配策略1
        pushl %ebp
        movl %esp, %ebp
        subl $24, %esp
        andl $-16, %esp
        movl $0, %eax
        subl %eax, %esp
        movl $0, -20(%ebp)
        movl $0, -16(%ebp)
        movl $1, -12(%ebp)
        movl $2, -12(%ebp)
        movl $3, -8(%ebp)
        movl $0, %eax
        leave
        ret
```

// 分配在栈上，地址由根据局部变量的先后顺序***由低到高***，退出作用域的变量空间会被***重用***

第四题

```
#include <stdio.h>
int main(){
    int a=0, b = 0;
    { int a = 1; }
    { int b = 2;
        { int a = 3; }
    }
   return 0;
}
```

```
main: // 存储分配策略2
        pushl %ebp
        movl %esp, %ebp
        subl $24, %esp
        andl $-16, %esp
        movl $0, %eax
        subl %eax, %esp
        movl $0, (%ebp)
        movl $0, -4(%ebp)
        movl $1, -8(%ebp)
        movl $2, -12(%ebp)
        movl $3, 16(%ebp)
        movl $0, %eax
        leave
        ret
```

太合理了.

第五题

```c
#include <stdio.h>
int main(){
    int a[6]={0,1,2,3,4,5};
    int i=6,j=7;
    int *p = (int*)(&a+1);
    printf("%d\n",*(p-1));
    return 0;
}
```

&a: int[6]*

```
.LC0:
        .long 0
        ...
        .long 5
.LC1:
        ...
main:
        ...
        leal -40(%ebp), %edi
        movl $.LC0, %esi
        cld
        movl $6, %eax
        movl %eax, %ecx
        rep
        movsl
```

int a[6]={0,1,2,3,4,5}

```
movl $6, -44(%ebp)
movl $7, -48(%ebp)
leal -40(%ebp), %eax
addl $24, %eax
movl %eax, -52(%ebp)
subl $8, %esp // esp-8
movl -52(%ebp), %eax
subl $4 , %eax
pushl (%eax) // esp-4  // 注意(%eax)和%eax
pushl $.LC1 // esp-4
call printf
addl $16, %esp // -8-4-4
movl $0, %eax
leal -8(%ebp), %esp // 要恢复esi, edi
popl %esi
popl %edi
leave
ret
```