阅读 PL0 编译器相关文档：完成以下任务：

在 PL0 编译器中，函数 interpret()在解释指令 LOD/STO 时的语义代码如下：

```
case LOD: //  指令格式 (LOD, l, a)
    stack[++top] = stack[base(stack, b, i.l) + i.a];
    break;
case STO: //指令格式 (STO, l, a)
    stack[base(stack, b, i.l) + i.a] = stack[top];
    printf("%d\n", stack[top]);
    top--;
    break;
```

(a) 你“扩展”PL0 编译器，添加了 LEA/LODA/STOA 等指令。格式为：(LEA,l,a)，(LODA,0,0)和(STOA,0,0)。其中“取地址”指令 LEA 用来获取名字变量在“运行时栈-stack”上“地址偏移”。而“间接读”指令 LODA 则表示以当前栈顶单元的内容为“地址偏移”来读取相应单元的值，并将该值存储到原先的栈顶单元中。而“间接写”指令 STOA 则将位于栈顶单元的内容，存入到次栈顶单元内容所代表的栈单元里，然后弹出栈顶和次栈顶。给出这样的 LODA/STOA/LEA 指令的语义代码。

(b) 现在继续扩展 PL0 编译器。假设你实现了若干 C 风格的表达式、类型及其声明体系，并可编译如下程序：

```
int main()
{
    int  i;
    int* q;
    int* a[10];
    int* (*b[10])[10];
    int* (*(*p)[10])[10];

    i = 100; q = &i; a[1] = q; b[1] = &a; p = &b;

    cout << _____p[0]_____ << endl; //输出 100，待补全

    cout << _____*p_____ << endl; //输出 100，待补全

}//程序
```

- 给出变量 a 和 p 的类型表达式。
  注意，int 即为类型表达式。指针类型表示为 pointer(T1)，T1 为指针所指向对象的类型表达式，数组类型表示为 array(number,T2)，数组元素的个数 number 为常量值，T2 为数组元素的类型表达式。

- *根据 PL0 编译环境设定*，上述程序中分配的总变量空间是多少？各个变量在活动记录中“地址偏移”是多少？

- 两条输出语句中*不同的*表达式各自仅包含唯一的名字变量 p。根据你的 C 语言知识，补全这两处输出语句中的源代码。

- 给出一个上述下划线处源代码对应的 PL0 代码（两处输出语句可任选其一产生 PL0

代码)。如需使用算术运算，可直接给出，例如，加法(ADD, 0, 0)，乘法(MUL, 0, 0)，以及加载常数于栈顶 (LIT, 0, 100)等指令。

(c) 再扩展 PL0 编译器，你添加了“引用”声明及处理。一个引用变量也具有一个地址单元，其中存储着被“引用”的其他变量的地址偏移。
考虑如下程序片段：

```
int *  &r = ...  //  r 是一个引用变量，被引用对象是一个 int 指针变量。
int func(int *i, int* &j, int k); // 函数 func 声明
```

对于函数调用：`func(r, r, *r)` 分别给出计算三个实参的“值”到 stack 栈顶的 PL0 代码。假设 r 的地址偏移为 3。

(a) PL0 编译器指令扩展.

1. LEA (取地址) 指令

case LEA : stack [++top] = base (stack, b, i.l) + i.a ; break;

2. LODA (间接读)

case LODA : stack [top] = stack [stack [top]] ; break;

3. STOA (间接写)

case STOA : stack [stack [top-1]] = stack [top] ;

top -= 2; break;

(b) C 风格表达式、类型及声明体系

1. 变量类型表达式

变量 'a' 类型表达式 array (10, pointer (int)) 表示一个大小为 10 的数组，其元素为指向整型的指针.

变量 'p' 类型表达式 pointer (array (10, pointer (array (10, pointer (int))))) 的指针.
表示一个指针，指向一个大小为 10 的数组，该数组每个元素都是指向另一个大小为 10 的整型指针数组

2. 变量空间和地址偏移.

- 根据 PL0 环境设定，假设 int 类型占 1 个存储单元，pointer 类型也占 1 个存储单元.
- 总变量空间: 'i' (1) + 'q' (1) + 'a' (10) + 'b' (10) + 'p' (1) = 23 个存储单元.
- 地址偏移: 'i' (0) + 'q' (1) + 'a[0]' (2), ..., 'a[9]' (11), 'b[0]' (12), ..., 'b[9]' (21), 'p' (22).

3. PL0 代码示例（输出语句）

假设选择输出 *p 值（假定为100）

```
LIT 0, 100
STO 0, 22
LOD 0, 22
LODA 0, 0
OPR 0, 21
```

(1) 引用声明及处理

func(r, *r) 调用的PL0代码. 假设r地址偏移为3, x地址偏移为3.

```
LOD 0, 3
LODA 0, 0
LOD 0, 3
LOD 0, 3
LODA 0, 0
STO 0, 3
```

一．针对如下 C 程序及其在 i386 Linux 下的汇编代码（片段）：

```
#include<stdio.h>
union var{
        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;
}

//第一题 C 程序
```

```
        .section   .rodata
.LC0:
        .string    "%x %s\n"
        .text
.globl main
        .type      main,@function
main:

        movl       %esp, %ebp
        subl       $40, %esp
        andl       $-16, %esp
        movl       $0, %eax
        subl       %eax, %esp
        movb       $50, -24(%ebp)
        movb       $48, -23(%ebp)
        movb       $49, -22(%ebp)
        movb       $0, -20(%ebp)
        leal       -24(%ebp), %eax

        movl       %eax, -28(%ebp)
        pushl      -28(%ebp)
        pushl      -24(%ebp)

        pushl      $.LC0
        call       printf
        addl       $16, %esp

        leave
        ret
//第一题 汇编程序
```

（a） 上述 C 程序的输出是什么？ data.i 的十六进制表示和 2016.

（b） 补全 10 处划线部分的汇编代码。

二．针对如下 C 程序及其汇编代码（片段）：

```
#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));
} //第二题 C 程序
```

```
      .file "test1.c"
      .text
.globl f
      .type     f,@function
f:
      pushl     %ebp
      movl      %esp,       %ebp
      movl      $100,       8(%ebp)
      movl      $16 ,       12(%ebp)
      movb      $65,        14(%ebp)
      movl      20(%ebp),   %eax
      pushl     %eax
      pushl     _____
      pushl     _____
      pushl     _____
      call      f
      addl      $16,        %esp
      leave
      ret

//当 N=2 时，生成的汇编代码片段。
```

```
      .file "test1.c"
      .text
.globl f
      .type     f,@function
f:
      pushl     %ebp
      movl      %esp,       %ebp
      pushl     %edi
      pushl     %esi
      movl      $100,       8(%ebp)
      movl      $36,        12(%ebp)
      movb      $65,        -1(%ebp)
      subl      $8,         %esp
      movl      (%eax),     %eax
      subl      $24,        %esp
      movl      %esp,       %edi
      movl      %eax,       %esi
      cld
      movl      0(%eax),    %eax
      movl      %eax,       %ecx
      rep
      movsl
      call      f
      addl      $32,        %esp
      leal      -8(%ebp), %esp
      popl      %esi
      popl      %edi
      leave
      ret

// rep movsl 为数据传送指令，即，由源地址 esi 开始的 ecx 个字的数据传送到由 edi 指示的目的地址。

//当 N=11 时，生成的汇编代码片段
```

(a) 补全划线处的汇编代码；清理栈空间，特别是调用函数后用来移除传递给此函数的参数。

(b) 从运行时环境看，addl $16, %esp 和 leal -8(%ebp), %esp 这两条汇编指令的作用是什么？调整栈指针到局部变量的开始。

(c) 结合上述两种汇编代码，简述编译器在按值传递结构变量时的处理方式。

编译器可能会在栈上分配一个新的结构体副本，并将内容复制到这个副本中。

三．针对如下 C 程序及其汇编代码（片段）：

```
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)++; }
//第三题 C 程序
```

(a)补全下划线处的空白汇编代码；
(b)main 函数中 for 循环结束时，数组 line 各元素值是多少？

元素i值为将其学号加1.

```
.globl g
      .type     g,@function
g:
      pushl     %ebp
      movl      %esp, %ebp
      movl      8(%ebp)④    , %eax
      movl      (%eax) ⑤    , %eax
      addl      $1,  ⑥ (%edx)
      movl      %eax ⑦    , %eax
      addl      $4,  ⑧ %edx
      leave
      ret

//第三题函数 g 的汇编代码片段
```

```
      .file  "p.c"
      .text
.globl main
      .type     main,@function
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  $9, ① -60(%ebp)
      jle   .L5
      jmp .L3
.L5:
      movl      -64(%ebp), %edx
      movl      -60(%ebp), %eax
      movl      %eax, (%edx)
      subl      $12, %esp
      leal      -64(%ebp), %eax
      pushl     %eax
      call      g
      add  $1,  ②  -60(%ebp)
      leal      -60(%ebp), %eax
      incl      (%eax)
      movl  -60(%③ebp), %eax
.L3:
      movl      $0, %eax
      leave
      ret

//第三题函数 main 的汇编代码片段
```

四． 针对如下 C 程序及其汇编代码（片段）：

(1) 补全下划线处的空白汇编代码；

(2) 描述所用编译器对 C 分程序所声明变量的存储分配策略：用栈来为局部变量分配内存.

```
#include <stdio.h>
int main()
{
  int a=0, b = 0;
  { int a = 1;  }
  { int b = 2;
    {  int a = 3;  }
  }
  return 0;
} //第四题 C 程序
```

```
main:
    pushl  %ebp
    movl %esp, %ebp
    subl   $24, %esp
    andl   $-16, %esp
    movl   $0, %eax
    subl   %eax, %esp
    movl   $0, -4(%ebp)
    movl   $0, -8(%ebp)
    movl   $1, -16(%ebp)
    movl   $2, -12(%ebp)
    movl   $3, -20(%ebp)
    movl   $0, %eax
    leave
    ret
//第四题 汇编代码片段
```

```
#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;
} //第五题 C 程序
```

```
.LC0:
    .long  0
    .long  1
    .long  2
    .long  3
    .long  4
    .long  5
.LC1:
    .string   "%d\n"
    .text
.globl main
    .type  main,@function
main:
    pushl  %ebp
    movl   %esp, %ebp
    pushl  %edi
    pushl  %esi
    subl   $48, %esp
    andl   $-16, %esp
    movl   $0, %eax
    subl   %eax, %esp
    leal   -40(%ebp), %edi
    movl   $.LC0, %esi
    cld
    movl   $6, %eax
    movl   %eax, %ecx
    rep
    movsl
    movl   $6, -44(%ebp)
    movl   $7, -48(%ebp)
    leal   -40(%ebp), %eax
    addl   $4, %eax
    movl   %eax, -52(%ebp)
    subl   $8, %esp
    movl   -52(%ebp), %eax
    subl   $4, %eax
    pushl  4.(%eax)
    pushl  $.LC1
    call   printf
    addl   $16, %esp
    movl   $0, %eax
    leal   -24(%ebp), %esp
    popl   %esi
    popl   %edi
    leave
    ret

//第五题 汇编代码片段
```

五． 仔细阅读所给 C 程序及其汇编代码片段。

(1) 指出波浪线处的汇编代码的作用；执行了内存块的复制操作.

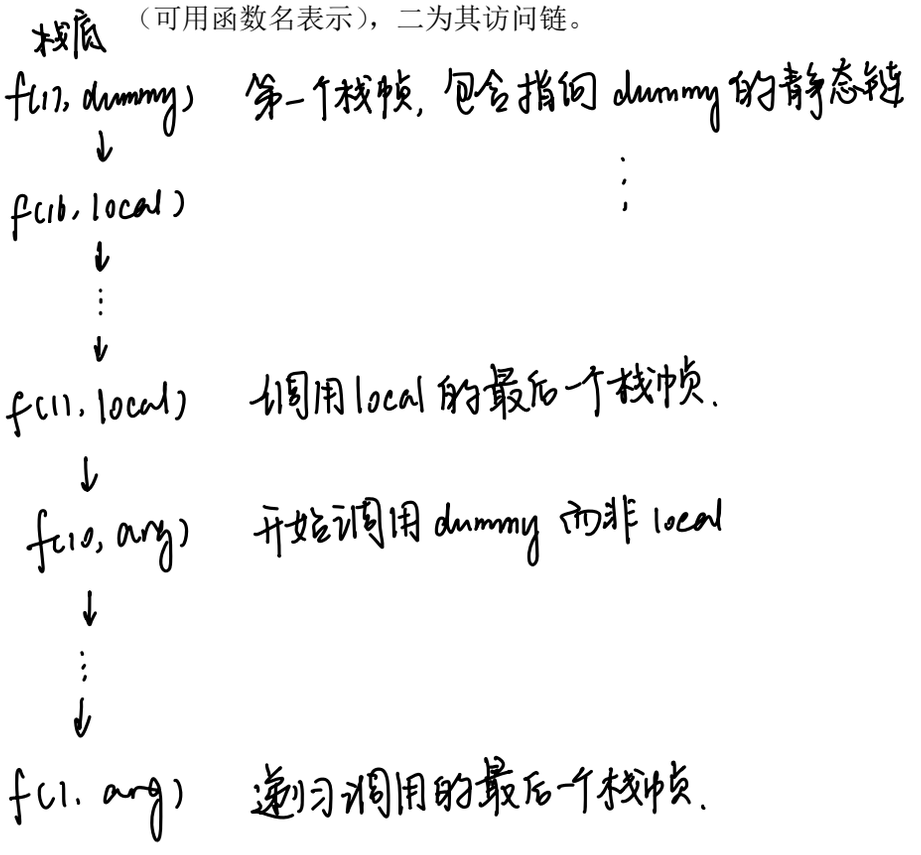
(2) 补全下划线处的空白汇编代码。

六． 假设以下假想的程序采用静态嵌套作用域规则：

```
program    staticLink
      procedure f(level, arg())
      //函数 f 有两个参数，整型变量 level，无参函数 arg

            procedure local() //  嵌套在 f 中的函数
            begin //无参函数 local，返回一个整型值。
                  return level;
            end

      begin //f 函数体
            if (level > 10)    return f(level-1, local);
            else    if (level > 1)    return f(level-1, arg);    else return arg();
      end

      procedure dummy()
      begin /*空的函数体*/end

begin //staticLink  函数体
      print(f(17, dummy));
end
```

（a） 给出该程序运行结果； None

（b） 给出函数调用 f(17, dummy)执行时运行栈上包含活动记录最多时的相关图示。假设按照逆序方式传递参数；函数作为参数传递时，需要两个单元，一为函数入口地址（可用函数名表示），二为其访问链。



3. (1)

Name	Category	Def	Ref	Diff
g	Variable	0	2	2
f	Variable	1	2	1
s	Variable	2	2	0
pf	Parameter	1	1	0
ps, pt	Parameter	2	2	0
u, v	Parameter	2	2	0
w	Parameter	2	2	0

(2)

grandpa的活动记录将包括返回地址和局部变量g。
father的活动记录将包括返回地址、控制链接（指向grandpa）、参数p_f和局部变量f。
son和uncle的活动记录将包括返回地址、控制链接（指向father）、它们的参数（p_s, p_t对son；u, v, w对uncle）、以及son的局部变量s。

(3)

pushq %rbp：保存旧的基指针，为新的函数调用准备。
movq %rsp,%rbp：设置新的基指针，这是当前函数堆栈帧的开始。
lea -32(%rsp),%rsp：为局部变量分配空间。
movq %rdi,-24(%rbp)：保存传入的第一个参数（父指针）到局部变量。
movw %si,-8(%rbp)和movw %dx,-16(%rbp)：保存其他传入的参数。
接下来的几行执行变量之间的赋值，更新g, f, s的值。