ICS习题课2

作业6

T1

The main program below calls a subroutine `F` . The `F` subroutine uses R3 and R4 as input, and produces an output which is placed in R0. The subroutine modifies registers R0, R3, R4, R5, and R6 in order to complete its task. `F` calls two other subroutines, `SaveRegisters` and `RestoreRegisters` , that are intended handle the saving and restoring of the modified registers (although we will see in question (b) that this may not be the best idea!)

```
; Main Program;
.ORIG x3000
...
...
JSR F
...
...
HALT
; R3 and R4 are input.
; Modifies R0, R3, R4, R5, and R6
; R0 is the output
;
F
JSR SaveRegisters
...
...
...
JSR RestoreRegisters
RET
.END
```

(a) Write the two subroutines `SaveRegisters` and `RestoreRegisters` .
(b) When we run the code we notice there is an infinite loop. Why? What small change can we make to our program to correct this error. Please specify both the correction and the subroutine that is being corrected.

> (a) 写出两个subroutines的具体内容
>
> ```
> SaveRegisters
> ST R0, SAVER0
> ST R3, SAVER3
> ST R4, SAVER4
> ST R5, SAVER5
> ST R6, SAVER6
> RET
> ```

```
8   ;
9   RestoreRegisters
10  LD R0, SAVER0
11  LD R3, SAVER3
12  LD R4, SAVER4
13  LD R5, SAVER5
14  LD R6, SAVER6
15  RET
16  ;
17  SAVER0 .BLKW x1
18  SAVER3 .BLKW x1
19  SAVER4 .BLKW x1
20  SAVER5 .BLKW x1
21  SAVER6 .BLKW x1
```

(b) 上述的代码为什么无法结束：F里调用了其他subroutine，R7存储的PC被修改

修改方式：调用SaveRegisters之前保存R7，并在 RET 前恢复

T2

Memory locations x5000 to x5FFF contain 2's complement integers. What does the following program do?

```
1           .ORIG x3000
2           LD R1, ARRAY
3           LD R2, LENGTH
4           AND R3, R3, #0
5   AGAIN   LDR R0, R1, #0
6           AND R0, R0, #1
7           BRz SKIP
8           ADD R3, R3, #1
9   SKIP    ADD R1, R1, #1
10          ADD R2, R2, #-1
11          BRp AGAIN
12          HALT
13          ARRAY .FILL x5000
14          LENGTH .FILL x1000
15          .END
```

上述程序实现了什么？

观察到条件分支之前，将R0的数值 AND 1，即判断R0是不是奇数，所以上述程序做的实际上就是检测ARRAY中奇数的数量

T3

Our code to compute n factorial worked for all positive integers n. Augment the iterative solution to `FACT` to also work for 0!

```
FACT    ST R1,SAVE_R1
        ADD R1,R0,#0
        ADD R0,R0,#-1
        BRz DONE
AGAIN   MUL R1,R1,R0
        ADD R0,R0,#-1
        BRnp AGAIN
DONE    ADD R0,R1,#0
        LD R1,SAVE_R1
        RET
SAVE_R1 .BLKW 1
```

上述代码是一个计算阶乘的代码，但是不能处理0的阶乘，该如何修改

增加0的特殊判断即可，下面是一个给出的例子

```
FACT    ST  R1,SAVE_R1
     ADD   R1,R0,#0
     BRnp  SKIP
     ADD   R1,R1,#1
     BRnzp DONE
SKIP ADD   R0,R0,#-1
     BRz   DONE
AGAIN MUL   R1,R1,R0
     ADD   R0,R0,#-1 ; R0 gets next integer for MUL
     BRnp  AGAIN
DONE ADD   R0,R1,#0 ; Move n! to R0
     LD  R1,SAVE_R1
     RET
SAVE_R1 .BLKW 1
```

T4

The following operations are performed on a stack:

PUSH A, PUSH B, POP, PUSH C, PUSH D, POP, PUSH E, POP, POP, PUSH F

(a) What does the stack contain after the PUSH F?

(b) At which point does the stack contain the most elements?

Without removing the elements left on the stack from the previous operations, we perform:

PUSH G, PUSH H, PUSH I, PUSH J, POP, PUSH K, POP, POP, POP, PUSH L, POP, POP, PUSH M

(c) What does the stack contain now?

> (a) 在push f之后栈空间：AF
>
> (b) 何时栈空间中元素最多：PUSH D或者PUSH E此时栈空间均有3个元素
>
> (c) 在上述操作结束后栈空间：AFM

T5

(a) What problem could occur if a program does not check the Ready bit of the KBSR before reading the KBDR?

(b) What problem could occur if the keyboard hardware does not check the KBSR before writing to the KBDR?

(c) Which of the above two problems is more likely to occur? Give your reason.

> (a) 如果在程序读取KBDR之前不检测KBSR会发生什么：程序可能反复读取同一个字符
>
> (b) 如果键盘硬件不检测KBSR直接写KBDR会发生什么：可能发生KBDR的上一个数据还未被读取就写入新的数据，用户输入的数据可能丢失
>
> (c) 上述的两个问题哪个更容易发生：a更容易发生，相较于程序的读取速度，用户的输入速度更慢

T6

Some computer engineering students decided to revise the LC-3 for their senior project.

In designing the LC-4, they decided to conserve on device registers by combining the KBSR and the DSR into one status register: the IOSR (the input/output status register). IOSR[15] is the keyboard device ready bit and IOSR[14] is the display device ready bit.

What are the implications for programs wishing to do I/O? Is this a poor design decision?

> 把KBSR和DSR合成成同一个寄存器IOSR会发生什么：
>
> 尽管寄存器的设计更简单，但是我们在判断时需要用掩码和IOSR做 AND 运算来获得具体位数的值，增加了写汇编代码的复杂性，因此不是一个好的方案

T7

The following LC-3 program is assembled and then executed. There are no assemble time or run-time errors.

What is the output of this program? Assume all registers are initialized to 0 before the program executes.

```
1           .ORIG x3000
2           LEA R0, LABEL
3           STR R1, R0, #4
4           TRAP x22
5           TRAP x25
6   LABEL   .STRINGZ "FUNKY"
7   LABEL2  .STRINGZ "HELLO WORLD"
8           .END
```

> 上述程序的输出是什么：
>
> STR修改了内存地址 LABEL+4 ， 将字符串提前结束了， 所以输出是： FUNK

T8

The program below, when complete, should print the following to the monitor:

```
1           ABCFGH
```

Insert instructions at (a)–(d) that will complete the program.

```
1           .ORIG x3000
2           LEA R1, TESTOUT
3   BACK_1  LDR R0, R1, #0
4           BRz NEXT_1
5           TRAP x21
6           ------------ (a)
7           BRnzp BACK_1
8           ;
9   NEXT_1  LEA R1, TESTOUT
10  BACK_2  LDR R0, R1, #0
11          BRz NEXT_2
12          JSR SUB_1
13          ADD R1, R1, #1
14          BRnzp BACK_2
15          ;
16  NEXT_2  ------------ (b)
17          ;
18  SUB_1   ------------ (c)
19  K       LDI R2, DSR
20          ------------ (d)
21          STI R0, DDR
22          RET
23  DSR     .FILL xFE04
24  DDR     .FILL xFE06
25  TESTOUT .STRINGZ "ABC"
26   .      END
```

> 上述代码的输出是 ABCFGH，补全代码
>
> 观察代码结构可以发现代码中只有ABC字符串，说明想要输出FGH需要额外操作，其中输出ABC的代码应当是前半部分代码，所以a处应当是R1迭代+1；下半部分是输出FGH的逻辑，观察到后部分的代码使用了DSR和DDR，可以判断是一个模拟输出字符的代码，同时由于代码中不含有ABC，而FGH刚好是ABC向后偏移得来的，所以可以填写c d 部分的代码，最后 程序应当有个退出逻辑，根据此填写b的值
>
> - ADD R1, R1, #1
> - TRAP x25
> - ADD R0, R0, #5
> - BRzp K

T9

Interrupt-driven I/O:

(a) What does the following LC-3 program do?

```
1           .ORIG x3000
2           LD R3, A
3           STI R3, KBSR
4           AGAIN LD R0, B
5           TRAP x21
6           BRnzp AGAIN
7   A       .FILL x4000
8   B       .FILL x0032
9           KBSR .FILL xFE00
10          .END
```

(b) If someone strikes a key, the program will be interrupted and the keyboard interrupt service routine will be executed as shown below. What does the keyboard interrupt service routine do?

```
1           .ORIG x1000
2           LDI R0, KBDR
3           TRAP x21
4           TRAP x21
5           RTI
6   KBDR    .FILL xFE02
7           .END
```

(c) Finally, suppose the program of part a started executing, and someone sitting at the keyboard struck a key. What would you see on the screen?

(d) In part c, how many times is the digit typed shown on the screen? Why is the correct answer: "I cannot say for sure."

> (a) 允许键盘中断，重复输出字符 2
>
> (b) 重复输出键盘两次
>
> (c) 程序运行时按了一个键会发生什么：先输出数个2 然后输出键盘键入的字符2到3次，再持续输出2
>
> (d) 为什么会输出2次到3：取决于中断发生的位置，如果中断正好发生在主程序 TRAP x21 前，中断返回后本次主程序也会输出一次键盘键入的字符

T10

What does the following LC-3 program do?

```
1                .ORIG x3000
2                LEA R6, STACKBASE
3                LEA R0, PROMPT
4                TRAP x22 ; PUTS
5                AND R1, R1, #0
6    LOOP        TRAP x20 ; IN
7                TRAP x21
8                ADD R3, R0, #-10 ; Check for newline
9                BRz INPUTDONE
10               JSR PUSH
11               ADD R1, R1, #1
12               BRnzp LOOP
13   INPUTDONE   ADD R1, R1, #0
14               BRz DONE
15   LOOP2
16               JSR POP
17               TRAP x21
18               ADD R1, R1, #-1
19               BRp LOOP2
20   DONE        TRAP x25 ; HALT
21
22   PUSH        ADD R6, R6, #-2
23               STR R0, R6, #0
24               RET
25   POP         LDR R0, R6, #0
26               ADD R6, R6, #2
27               RET
28   PROMPT      .STRINGZ "Please enter a sentence:"
29               STACKSPAC .BLKW #50
30               STACKBASE .FILL #0
31               .END
```

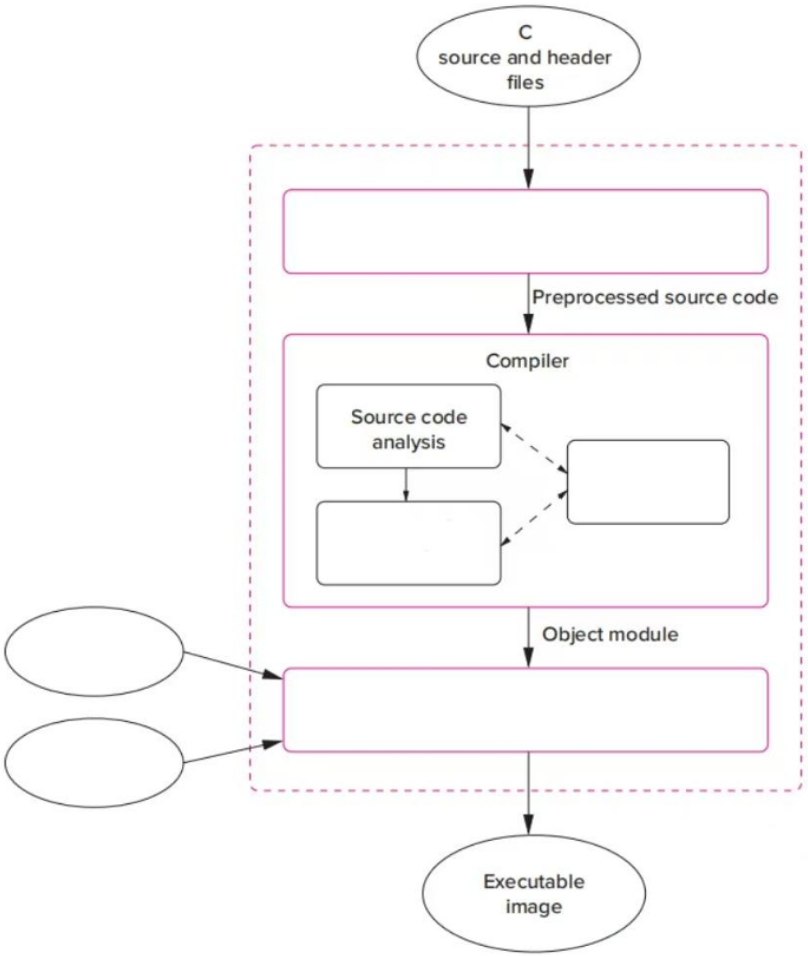
> 程序将输入字符串存到栈中，并通过POP栈的方式输出，所以程序是逆序输出输入的字符串

作业7

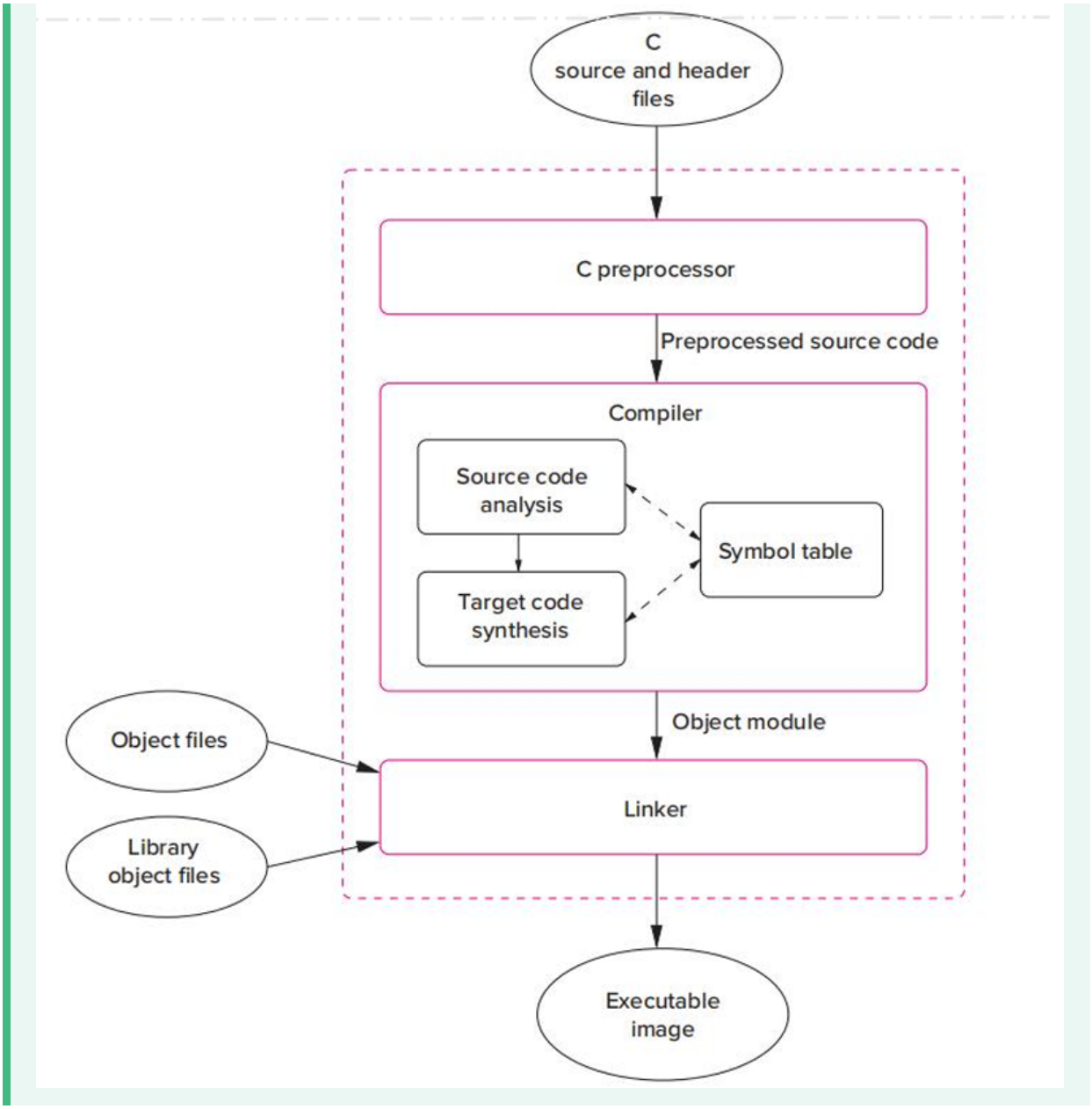
T1

Both C and C++ are compiled languages. The C or C++ compiler follows the typical mode of translation from a source program to an *executable image*. An executable image is a machine language representation of a program that is ready to be loaded into memory and executed. The compilation mechanism involves several distinct com ponents, notably the preprocessor, the compiler itself, and the linker.

Please complete the following image and illustrate the overall compilation process of C.



对于编译性的语言（C/C++为例）补全编译的过程，概念性的：预处理 --> 编译 --> 汇编 --> 链接：


C
source and header
files
C preprocessor
Preprocessed source code
Compiler
Source code
analysis
Symbol table
Target code
synthesis
Object module
Object files
Linker
Library
object files
Executable
image

T2

Another advantage of compilation over interpretation is that a compiler can optimize code more thoroughly. Since a compiler can examine the entire program when generating machine code, it can reduce the amount of computation by analyzing what the program is trying to do.
The following algorithm performs some very straightforward arithmetic based on values typed at the keyboard. It outputs a single result.
1. Get W from the keyboard
2. X<-W+W
3. Y<-X+X
4. Z<-Y+Y
5. Print Z to the screen

a. An interpreter would execute the program statement by statement. In total, five statements would execute. If the underlying ISA were capable of all arithmetic operations (i.e., addition, subtraction, multiplication, division), at least how many operations would be needed to carry out this program? State what the operations would be.

b. A compiler would analyze the entire program before generating machine code, and it would possibly optimize the code. If the underlying ISA were capable of all arithmetic operations (i.e., addition, subtraction, multiplication, division), at least how many operations would be needed to carry out this program? State what the operations would be.

> 编译性的语言可以在编译的过程中对代码进行优化，而对于解释性语言只能一步一步执行
>
> (a) 完成上述的程序至少需要多少次运算，分别是什么：
>
> - get w from the keyboard：不需要进行计算
> - X <- W + W：执行一次加法计算
> - Y <- X + X：执行一次加法计算
> - Z <- Y + Y：执行一次加法计算
> - Print Z to the screen：不需要进行计算
>
> (b) 编译优化后至少需要执行几次运算：三次加法可以变成一个乘法 Z = 8 * W

T3

初始时 a = 6 b = 9

Expression	Value of expression	Value of a afterwards	Value of b afterwards
a | b	15	6	9
a || b	1	6	9
a & b	0	6	9
a && b	1	6	9
!(a + b)	0	6	9
a % b	6	6	9
b / a	1	6	9
a = b	9	9	9
a = b = 5	5	5	5
++a + b—	16	7	8
a = (++b < 3) ? a : b	10	10	10
a <<= b	3072	3072	9

T4

Explain the differences between the following C statements:

a. j = i++;

b. j = ++i;

c. j = i + 1;

d. i += 1;

e. j = i += 1;

f. Which statements modify the value of i? Which ones modify the value of j? If i=0 and j=1 initially, what will the values of i and j be after each statement is run separately?

> a. j 等于 i 之前的值同时 i 自增
>
> b. j 等于 i 自增后的值
>
> c. j 等于 i + 1 的值，i 不变
>
> d. i 自增
>
> e. i 自增 同时 j 等于 i
>
> f. 根据上面的分析，a,b,d,e改变了 i 的值，a,b,c,e 改变了 j 的值，
>
> - i = 1, j = 0
> - i = 1, j = 1
> - i = 0, j = 1
> - i = 1, j = 1
> - i = 1, j = 1

T5

At least how many times will the statement called loopBody execute the following constructs?

a. while (condition)
loopBody;

b. do
loopBody;
while (condition);

c. for (init; condition; update)
loopBody;

d. while (condition1)
for (init; condition2; reinit)
loopBody;

e. do
do
loopBody;
while (condition1);
while (condition2);

> 上述的循环至少执行几次
>
> - 0 次 while一开始为假不会进入循环
> - 1 次 do while 至少进入一次循环
> - 0 次 for循环一开始为假不会进入循环
> - 0 次 同理
> - 1 次 同理

T6

Provide the output of each of the following code segments.

a. int x = 20;
int y = 10;
while ((x > 10) && (y & 15)) {
y = y + 1;
x = x - 1;
printf("*");
}

b. for (int x = 10; x ; x = x - 1)
printf("*");

c. for (int x = 0; x < 10; x = x + 1)
if (x % 2)
printf("*");

d. int x = 0;
while (x < 10) {
for (int i = 0; i < x; i = x + 1)
printf("*");
x = x + 1;
}

> - y&15 在 y=16时不为真，所以会输出6次
> - x=0 时循环结束，所以会输出10次
> - x是奇数的时候输出，输出5次
> - 每次进入for最多只会输出一次，所以输出9次

T7

The following C program uses a combination of global variables and local variables with different scope. What is the output?

```c
int t = 1;  // Global variable
int sub1(int fluff);
int main(void) {
    int t = 2;
    int z;
    z = t;
    z = z + 1;
    printf("A: The variable z equals %d\n", z);
    {
        z = t;
        t = 3;
```

```
        {
            int t = 4;
            z = t;
            z = z + 1;
            printf("B: The variable z equals %d\n", z);
        }
        z = sub1(z);
        z = z + 1;
        printf("C: The variable z equals %d\n", z);
    }
    z = t;
    z = z + 1;
    printf("D: The variable z equals %d\n", z);
}
int sub1(int fluff) {
    int i;
    i = t;
    return (fluff + i);
}
```

> **注意作用域**
>
> A: The variable z equals 3
>
> B: The variable z equals 5
>
> C: The variable z equals 7
>
> D: The variable z equals 4

T8

The following code reads a string from the keyboard and prints out a version with any uppercase characters converted to lowercase.

However, it has a flaw. Identify it.

```
#include <stdio.h>
#define MAX_LEN 10
char* LowerCase(char* s);
int main(void) {
    char str[MAX_LEN];
    printf("Enter a string : ");
    scanf("%s", str);
    printf("Lowercase: %s \n", LowerCase(str));
}
char* LowerCase(char* s) {
    char newStr[MAX_LEN];
    for (int index = 0; index < MAX_LEN; index++) {
        if ('A' <= s[index] && s[index] <= 'Z')
            newStr[index] = s[index] + ('a' - 'A');
        else
            newStr[index] = s[index];
    }
    return newStr;
}
```

> 代码有什么漏洞：返回了局部变量newStr的地址，在函数返回后，该部分内存会被直接释放
>
> 修改方式：使用全局数组等

T9

```c
#include <stdio.h>
int main(void) {
    int x = 0;
    int y = 0;
    char label[10];
    scanf("%d %d", &x, &y);
    scanf("%s", label);
    printf("%d %d %s\n", x, y, label);
}
```

a. What gets printed out if the input stream is 46 29 BlueMoon?

b. What gets printed out if the input stream is 46 BlueMoon?

c. What gets printed out if the input stream is 111 999 888?

> - 46 29 BlueMoon
> - 46 0 BlueMoon (y无法正确获取值)
> - 111 999 888

T10

The following C program is compiled into the LC-3 machine language and executed. The run-time stack begins at xEFFF. The user types the input abac followed by a return.

```c
#include <bits/stdc++.h>
#define MAX 4
struct Rec {
    char ch;
    struct Rec* back;
};
int main(void) {
    struct Rec *ptr, pat[MAX + 2];
    int i = 1, j = 1;
    printf("Pattern: ");
    pat[1].back = pat;
    ptr = pat;
    while ((pat[i].ch = getchar()) != '\n') {
        pat[++i].back = ++ptr;
        if (i > MAX)
            break;
    }
```

```
18        while (j <= i)
19            printf("%d ", pat[j++].back - pat);
20        // Note the pointer arithmetic here: subtraction
21        // of pointers to structures gives the number of
22        // structure elements, not the number
23        // of memory locations
24    }
```

a. Show the contents of the stack frame for main when the program terminates.

b. What is the output of this program for the input abac?

> 程序离开while循环时，i的值为5，读取了四个字符，所以j的循环遍历范围：1，2，3，4，5
>
> 最后的输出是 0 1 2 3 4
>
> 程序在退出时栈帧中即为定义的这些变量：*ptr, pat[6],i,j