Assignment 1: Legacy Fortran (25%)

REVERSE POLISH NOTATION TRANSLATOR

INTRODUCTION

Ever heard of Reverse Polish Notation (RPN)? Probably not. Algebraic notation is often termed *infix* notation - the arithmetic operator appears *between* the two operands to which it is being applied. This may require parentheses to specify the order of operations. *Postfix* notation, on the other hand, eliminates the need for parentheses because the operator is placed directly *after* the two operands to which it applies. This form of notation is sometimes called Reverse Polish Notation after the nationality of its creator, Polish logician Jan Lukasiewicz. For many years, HP has sold handheld calculators which require input in RPN, starting with the HP-35 in 1972. It worked great, but there was a learning curve - converting from traditional expressions to RPN expressions. Here is a RPN expression:

```
AB+C*
```

The plus sign applies to the two operands to the left of it. After doing the addition, the expression is scanned looking for the next operator. When it is found, it is applied to the two quantities preceding it, which are A+B and C. So this RPN expression is equivalent to the algebraic expression `(A+B)*C`. Here are some more algebraic expressions and their RPN equivalents:

Algebraic	Polish
`A+B+C+D`	`AB+C+D+`
`A+B*C+D`	`ABC*+D+`
`(A+B)*(C+D)`	`AB+CD+*`

It affords a better way of using a calculator by using fewer keystrokes. Consider the following example:

```
(7 + 4) / (9 – 5) = ?
```

Mode	Process	Keystrokes
Algebraic	1. Calculate **9 – 5 =** 2. Store 4 in memory **m+** 3. Calculate **7 + 4 =** 4. **Divide** 11 by 4 in memory **mr =**	12
RPN	7 [ENTER] 4 + 9 [ENTER] 5 – /	9

TASK

Attached to this assignment is a reverse-polish translator written in Fortran IV. The task is to migrate the legacy Fortran IV program into a Fortran-90/95 program. This involves converting and removing any structures which are relevant/irrelevant. Your code should be clean and easy to understand (unlike the existing code). The program contains a number of structures which should be modified or removed to make the program more maintainable. Some examples include:

- unstructured statements in the form of GO TO statements, and labels.
- Hollerith characters which are replaced by a character string

Part of the process of re-engineering a piece of code involves understanding what the code does. Spend some time analyzing the code using the flowchart for the algorithm in the Appendix. This will help with understanding how the code works.

DESIGN DOCUMENT

Discuss your re-designed program in 3-4 page *design document*, explaining decisions you made in the re-engineering process. Consider the design document a synopsis of your re-engineering process. One page should include a synopsis of the approach you took to migrate the program (e.g. it could be a numbered list showing each step in the process). Identify the legacy structures/features and how you modernized them.

Some of the questions that should be answered include:

- Would it have been easier to re-write the program from scratch in a language such as C?
- What were the greatest problems faced during the re-engineering process?
- Is your program shorter or longer? Why?
- Is there a better way of writing the program?

Your program should compile using `gfortran`.

TESTING

The program can be tested using the following series of algebraic expressions.

A*(B+C)	→	ABC+*	(A+B)*(C+D)	→	AB+CD+*
(A+B)*C	→	AB+C*	A-B/C	→	ABC/-
A+B*C+D	→	ABC*+D+	(A-B)/C	→	AB-C/
A/B+C	→	AB/C+	A/(B+C+D)	→	ABC+D+/
A/B/C	→	AB/C/	(A/B)/C	→	AB/C/
A*B-C+D	→	AB*C-D+	A*B-(C+D)	→	AB*CD+-

GOING BEYOND (5%)

Extend the re-engineered program to include an exponentiation operator, represented by a ^ character, and having higher precedence than multiplication or division.

SKILLS

Fortran programming, re-engineering through migration, program comprehension, ability to review specifications

DELIVERABLES

Either submission should consist of the following items:

- The design document.
- The code (well documented and styled appropriately of course).

APPENDIX - RPN

The algorithms first task is to identify the various characters as operands or operators. The operands are single letter variables. The operators are:

```
- + * / ( )
```

Then each operator is assigned a ranking (precedence) to determine where it appears in the RPN expression. In the absence of parentheses, multiplication and division are performed before addition an subtraction. Otherwise the ranking is as follows:

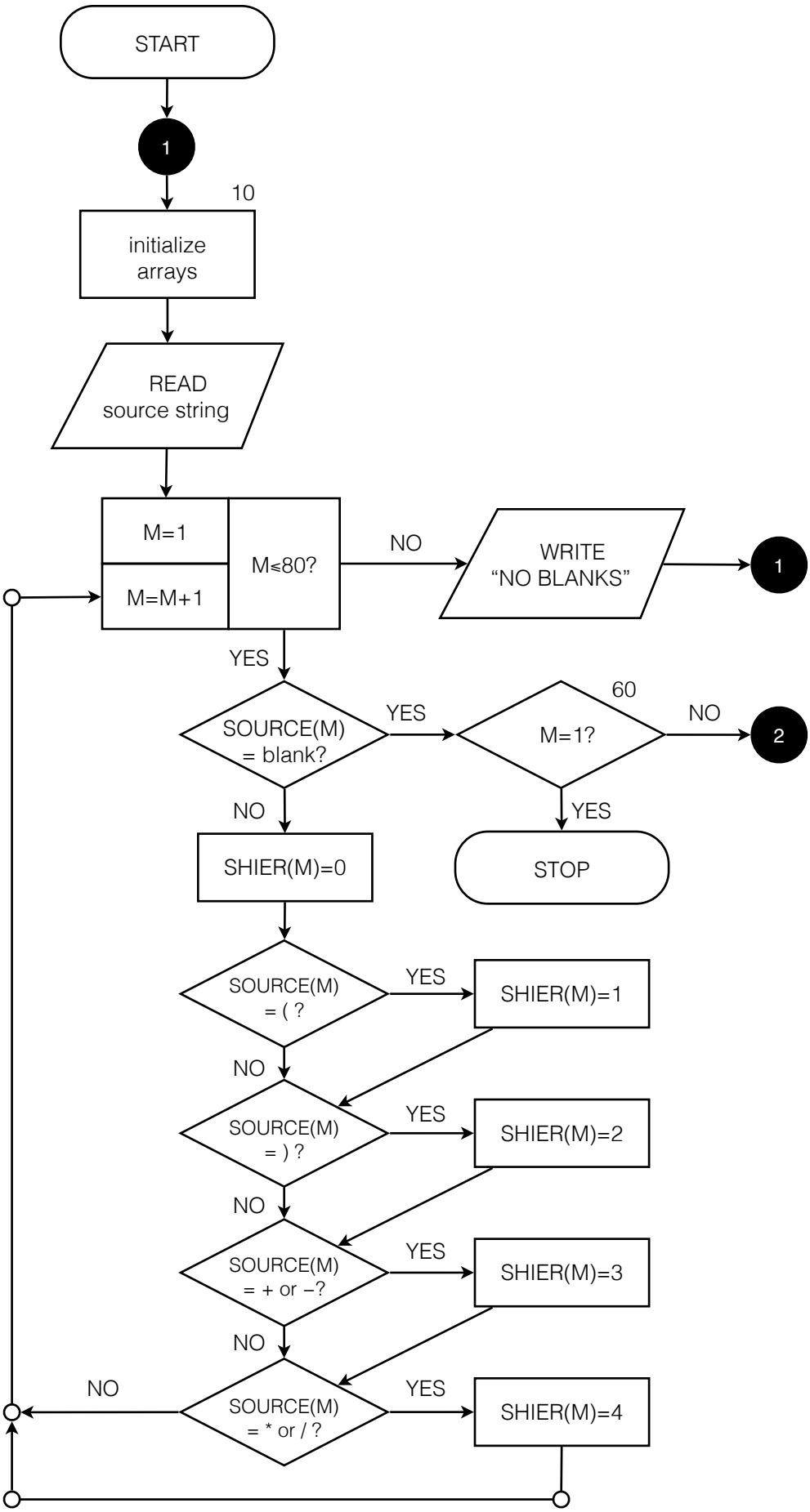
```
(        1
)        2
+ -      3
* /      4
```

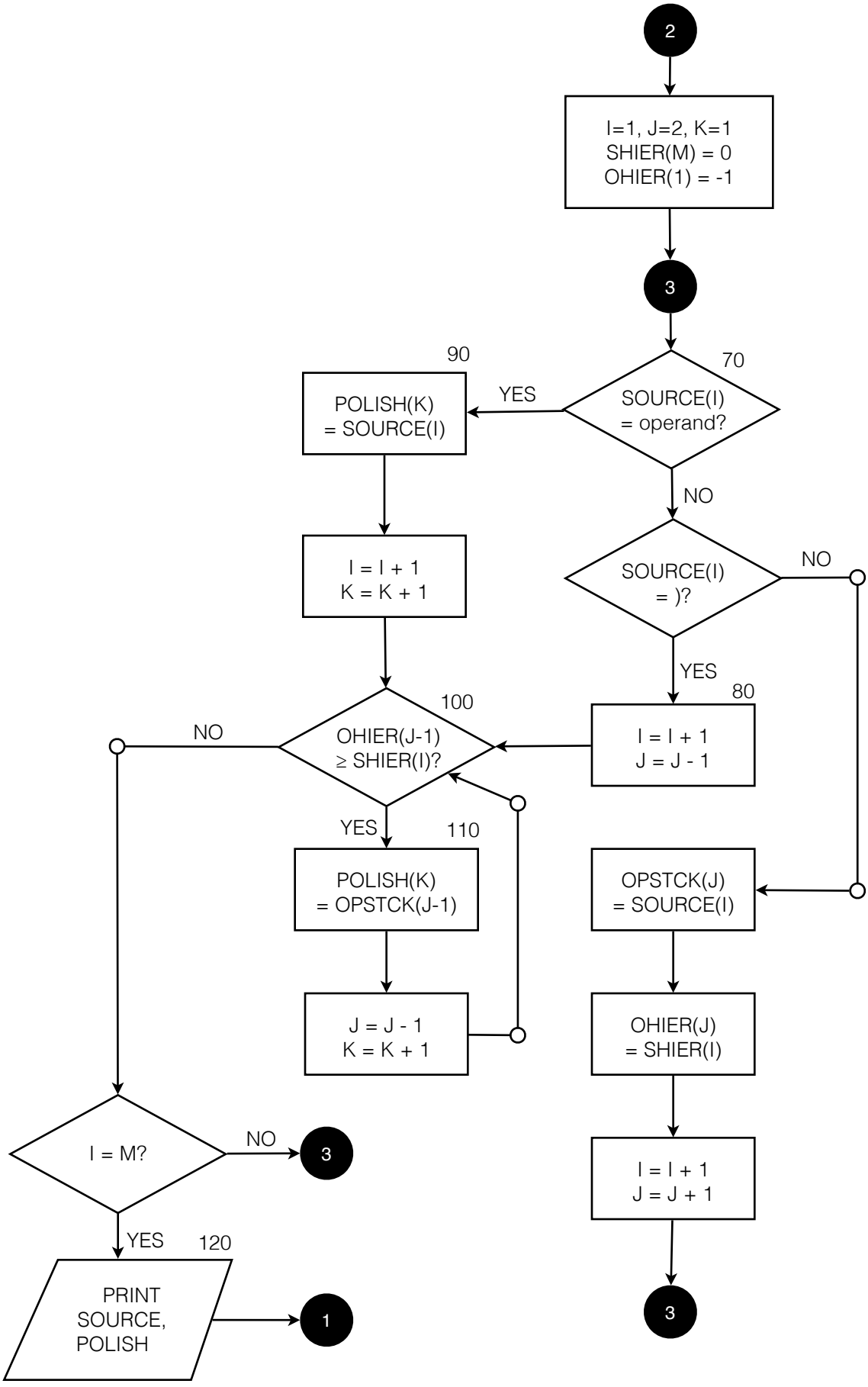
A ranking of zero identifies an operand.

The basic premise of the algorithm is as follows. Operands are always transferred to the output string (POLISH) as soon as they are encountered. Operators, except for right parenthesis, are always transferred to the OPSTCK (OPerator STaCK), to await transfer to the output string. When operators are transferred to OPSTCK, their ranking is assigned to the corresponding element of an array called OHIER (Operator HIERarchy).

After an operand has been transferred to the output string, a check is made to see if the last entry in OPSTCK has a higher ranking than the next operator in the input, or the same ranking - if so, the operator from OPSTCK is transferred to the output. Whenever a right parenthesis is found in the input, it is ignored and the matching left parenthesis, which will always be the last entry in the operator stack at this point, is also ignored.

The algorithm for the Fortran IV program is shown in flow-diagram form on the next two pages.


START
1
10
initialize arrays
READ source string
M=1
M=M+1
M≤80?
NO
WRITE "NO BLANKS"
1
YES
SOURCE(M) = blank?
YES
60
M=1?
NO
2
NO
YES
SHIER(M)=0
STOP
SOURCE(M) = (?
YES
SHIER(M)=1
NO
SOURCE(M) =) ?
YES
SHIER(M)=2
NO
SOURCE(M) = + or −?
YES
SHIER(M)=3
NO
NO
SOURCE(M) = * or / ?
YES
SHIER(M)=4


2
I=1, J=2, K=1
SHIER(M) = 0
OHIER(1) = -1
3
70
SOURCE(I)
= operand?
YES
90
POLISH(K)
= SOURCE(I)
NO
SOURCE(I)
=)?
NO
YES
80
I = I + 1
J = J - 1
I = I + 1
K = K + 1
100
OHIER(J-1)
≥ SHIER(I)?
NO
YES
110
POLISH(K)
= OPSTCK(J-1)
J = J - 1
K = K + 1
OPSTCK(J)
= SOURCE(I)
OHIER(J)
= SHIER(I)
I = I + 1
J = J + 1
3
I = M?
NO
3
YES
120
PRINT
SOURCE,
POLISH
1