Part A Write a MIPS subroutine \texttt{SumMags} that reads in a vector of integers and sums up the magnitude (absolute value) of each element, placing the sum of magnitudes in register $3$. Assume the length of the vector (# of integer elements) is given in register $2$ and is $> 0$, and the base address of the vector is in register $1$. Your code calls the subroutine \texttt{Abs}, which computes the absolute value of an integer $x$ given in register $4$; it returns $|x|$ in register $4$. Follow the steps outlined in the comments in the rightmost column below. \textbf{You may modify only registers $1$ through $4$.}

egin{tabular}{|c|c|c|}
\hline
\textbf{label} & \textbf{instruction} & \textbf{comment} \\
\hline
\texttt{SumMags:} & \texttt{addi} $3$, $0$ $0$ & \# initialize running sum ($3 = 0$) \\
\hline
\texttt{Loop:} & \texttt{lw} $4$, $0$(\texttt{1}) & \# load current vector element $x$ into $4$ \\
\hline
\texttt{B:} & [leave blank for part A] & \# code to be written in part B to \# preserve registers on stack \\
\hline
 & \texttt{jal Abs} & \# call Abs ($4 = |x|$) \\
\hline
\texttt{C:} & [leave blank for part A] & \# code to be written in part C to \# restore registers on stack \\
\hline
 & \texttt{add} $3$, $3$, $4$ & \# add $|x|$ to running sum \\
\hline
 & \texttt{addi} $1$, $1$, $4$ & \# increment vector pointer to next element \\
\hline
 & \texttt{addi} $2$, $2$, $-1$ & \# decrement number of elements by 1 \\
\hline
 & \texttt{bne} $2$, $0$, \texttt{Loop} & \# if number of elements $\neq 0$, loop back \\
\hline
 & \texttt{jr} $31$ & \# return to caller \\
\hline
\end{tabular}

Part B To ensure that \texttt{SumMags} can be properly called by another subroutine and that \texttt{SumMags} can call \texttt{Abs} without losing any of the intermediate values it computes, you must add code before and after the “\texttt{jal Abs}” instruction. Write MIPS code to preserve registers before the \texttt{jal} by pushing them on the stack. Assume \texttt{Abs} can modify \textit{any} registers, not just $4$.

\begin{tabular}{|c|c|c|}
\hline
\textbf{label} & \textbf{instruction} & \textbf{comment} \\
\hline
\texttt{B:} & \texttt{addi} $29$, $29$, $-4$ & \# push $31$ by adjusting SP \\
\hline
 & \texttt{sw} $31$, $0$(\texttt{29}) & \# and storing $31$ \\
\hline
 & \texttt{addi} $29$, $29$, $-4$ & \# push $1$ by adjusting SP \\
\hline
 & \texttt{sw} $1$, $0$(\texttt{29}) & \# and storing $1$ \\
\hline
 & \texttt{addi} $29$, $29$, $-4$ & \# push $2$ by adjusting SP \\
\hline
 & \texttt{sw} $2$, $0$(\texttt{29}) & \# and storing $2$ \\
\hline
 & \texttt{addi} $29$, $29$, $-4$ & \# push $3$ by adjusting SP \\
\hline
 & \texttt{sw} $3$, $0$(\texttt{29}) & \# and storing $3$ \\
\hline
 & \texttt{jal Abs} & \# call Abs ($4 = |x|$) \\
\hline
\end{tabular}

\textbf{Alternative answer}, more effectively using register indirect plus displacement addressing:
### B:

```mips
addi $29, $29, -16  # make room for 4 words on stack
sw   $31, 12($29)  # preserve $31
sw   $1, 8($29)  # preserve $1
sw   $2, 4($29)  # preserve $2
sw   $3, 0($29)  # and storing $3
jal Abs  # call Abs   ($4 = |x|)
```

### Part C

Write MIPS code to restore registers after the `jal` by popping them from the stack. Assume `Abs` can modify any registers, not just $4.

### Alternative answer:

```mips
jal Abs  # call Abs   ($4 = |x|)
C:  lw   $3, 0($29)  # pop $3 by loading it and
    addi $29, $29, 4  # adjusting SP
    lw   $2, 0($29)  # pop $2 by loading it and
    addi $29, $29, 4  # adjusting SP
    lw   $1, 0($29)  # pop $1 by loading it and
    addi $29, $29, 4  # adjusting SP
    lw   $31, 0($29)  # pop $31 by loading it and
    addi $29, $29, 4  # adjusting SP
addi $29, $29, 16  # adjust SP
```

### Alternative answer:

```mips
jal Abs  # call Abs   ($4 = |x|)
C:  lw   $3, 0($29)  # restore $3
    lw   $2, 4($29)  # restore $2
    lw   $1, 8($29)  # restore $1
    lw   $31, 12($29)  # restore $31
addi $29, $29, 16  # adjust SP
```