disbamer.sh

> bash disbamer.sh 43 | less -S

Was a read number supplied?

$# : Number of arguments passed to bash script

We looking for one input only!

```
if [[ $# -ne 1 ]]; then
    echo messages
    exit 2
fi
```

Is **samtools** loaded?

Try samtools, output to null, and redirect error to stdout, if we get output then show message

```
> command -v samtools >/dev/null 2>&1
  || { echo >&2 "- Is samtools
  loaded? "; exit 1; }
```

Extract Read from SAM file.

run samtools, use sed to get line requested, and put read in to var bamdata

```
> bamdata=`samtools view sam.lnk | sed
  "${1}q;d"`
```

Get read data fields.

Use awk to get fields from bamdata. Bash 'read' puts output into variable : seqD

```
> read seqD <<< $(echo "$bamdata"| awk
  '{print $10}')
# Repeat for flag region position
  sequence cigar …
```

Check sequence field.

Secondary alignments don't hold sequence

Sam data only has sequence in primary/supplementary mapped reads.

```
> if [ "$seqD"== "*"]; then
  echo "No sequence data : secondary?"
  exit 1
fi
```

Cont…

disbamer.sh

> bash disbamer.sh 43 | less -S

Using CIGAR calculate length of Reference sequence required

awk script

Use awk script to calculate. Use bash 'read' to assign result to variable, returns '*' if error.

```
> read seqlengthD <<< $(awk -v
    cigA="$cigarD"-f cigtoRefLen.awk)
```

Obtain reference matching sequence for read

awk script

Send bash variables for 'region, position and length' to the awk script; return sequence

```
> read seqrefD <<< $(awk -v
    regA="$regionD"-v posA="$positionD"-v
    lenA="$seqlengthD"-f getrefseq.awk
    ref.lnk)
```

Display read alongside Reference with indels marked

awk script

Send bash variables for 'cigar, read and reference sequences' to the awk script for display

```
> awk -v cigA="$cigarD"-v seqA="$seqD"-v
    refA="$seqrefD"-f viewread.awk
```

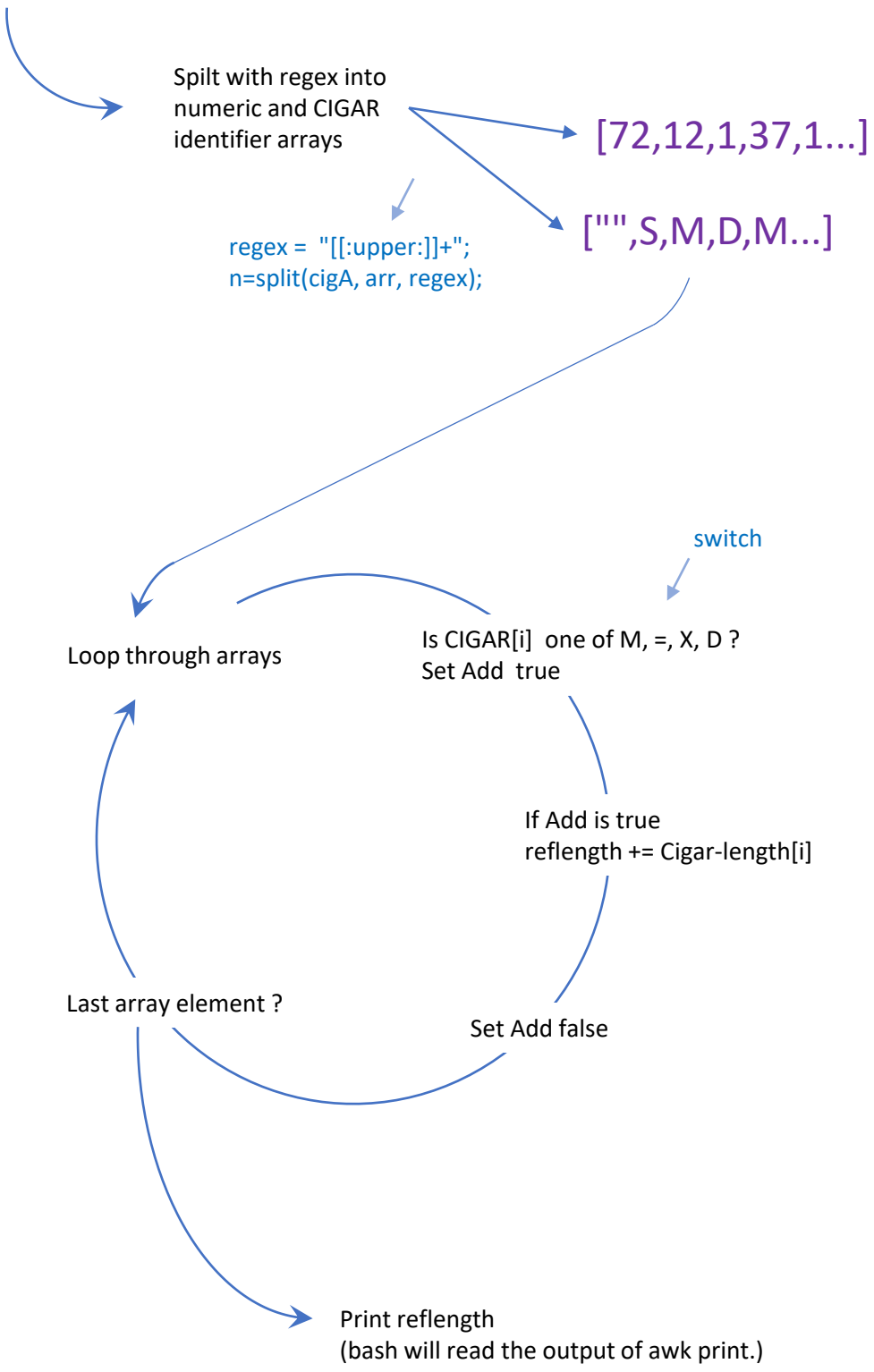
disbamer.sh
> bash disbamer.sh 43 | less -S

```
#!/bin/sh                    # - D - I - S - B - A - M - E - R -

# COMMENT: if no args display message to stderr ">&2"
if [[ $# -ne 1 ]]; then
   echo "------ Usage: bash disbamer.sh 2  (2 is the line of read number) ">&2
   echo "------ Display a read from a bam file along with associated reference sequence.    ">&2
   echo "------ disbamer will call 'samtools view' on a sam file defined by soft link to 'sam.lnk' and display a read.  " >&2
   echo  ------ Then it will look up reference \(via ref.lnk\) and display reference sequence for read from sam file.  >&2
   exit 2
fi

echo ">>> Running disbamer for read $1 ">&2
echo ------ Calling samtools view for file $(basename $(readlink sam.lnk)) to display read in line :  $1
echo bam path: $(readlink sam.lnk); echo ref path: $(readlink ref.lnk)

# COMMENT: check samtools loaded!
command -v samtools >/dev/null 2>&1 || { echo >&2 "------ Is samtools loaded? error running samtools, exiting."; exit 1; }

echo ---------R-E-A-D----B-A-M------------ "(samtools sam.lnk)"
bamdata=`samtools view sam.lnk | sed "${1}q;d"`
# echo "$bamdata"; # echo ------------------------------------
read readD <<< $(echo "$bamdata"| awk '{print $1}')
echo read "$readD"
read flagD <<< $(echo "$bamdata"| awk '{print $2}')
echo flag "$flagD"
read regionD <<< $(echo "$bamdata"| awk '{print $3}')
echo region "$regionD"
read positionD <<< $(echo "$bamdata"| awk '{print $4}')
echo position "$positionD"
read qualityD <<< $(echo "$bamdata"| awk '{print $5}')
echo quality "$qualityD"
read cigarD <<< $(echo "$bamdata"| awk '{print $6}')
echo cigar "$cigarD"
read seqD <<< $(echo "$bamdata"| awk '{print $10}')
echo sequence ${seqD:1:50} "...."  # substring
# echo "check_"$seqD"_check"
if [ "$seqD"== "*"]
then
  echo "No sequence data present, secondary alignments in bam do not show bases."
  echo "check data,  exiting.....";   echo ------------------------------------;   exit 1
fi

# need length of reference to extract - from cigar?
read seqlengthD <<< $(awk -v cigA="$cigarD"-f cigtoRefLen.awk)
echo ---------l-e-n-g-t-h---------- "(cigtoRefLen.awk)"
echo matching reference length required : "$seqlengthD"
if [ "$seqlengthD"== "*"]
then
  echo "CIGAR STRING error, 0 length sequence.";   echo "check data,  exiting.....";   echo ------------------------------------
  exit 1
fi

echo ---------r-e-f-e-r-e-n-c-e---------- "(getrefseq.awk ref.lnk)"
echo Please Wait! getting matching reference:  "----may take some time....."
read seqrefD <<< $(awk -v regA="$regionD"-v posA="$positionD"-v lenA="$seqlengthD"-f getrefseq.awk ref.lnk)
if [ "$seqrefD"== ""]
then
  echo "REFERENCE NOT FOUND. .";   echo "please check data and reference link,  exiting.....";   echo ---------------------------------
  exit 1
fi

echo
echo ---------g-e-n-o-m-i-c---v-i-e-w----- "(viewread.awk)"
awk -v cigA="$cigarD"-v seqA="$seqD"-v refA="$seqrefD"-f viewread.awk
```

cigtoRefLen.awk

Using CIGAR calculate length of Reference sequence required

72S12M1D37M1D10M2D2M3D14M1D4M1D23M2D1M1I24M692S



Note: All code is in a BEGIN block, and only operates on passed variable (via –v)

cigtoRefLen.awk

```
#! /usr/bin/awk -f

# Given a CIGAR string this script will calculate the length of the Reference sequence it aligns to. will count matches,
mismatches and deletes, from the cigar string. (Ignore I, N, P, S, H)
#  usage from a bash script : # assign CIGAR string to bash variable
#                       cigarD="72S12M1D37M1D10M2D2M3D14M1D4M1D23M2D1M1I24M692S"
                        # call the awk script passing the bash variable to a awk variable using -v
                        # read the output of the awk script into another bash variable (seqlengthRef) with bash "read var <<<"

#                       read seqlengthRef <<< $(awk -v cigA="$cigarD" -f CigtoRefLen.awk)

BEGIN {
# printf("\nyou passed me : %s\n",cigA)

regex =  "[[:upper:]]+";
n=split(cigA, arr, regex);
#            arr will be array of numbers from CIGAR [72,12,1,37,1...]
regex =  "[[:digit:]]+";
m=split(cigA, brr, regex);
#           brr will be array of letters form CIGAR ["",S,M,D,M...]  will have empty letter in first position, from split function.

rlen=0
addPos=0   # flag to add current cigar value to length

for ( i=1; i<n; i++ ) {
       # print arr[i] ":" brr[i+1]
      len = arr[i]

      switch( brr[i+1] ) {

      case "M" :     addPos = 1
                  break;
      case "=" :     addPos = 1
                  break;
      case "X" :     addPos = 1
                  break;
      case "D" :     addPos = 1
                  break;
      case "I" :       break;
      case "N" :       break;
      case "P" :       break;
      case "S" :       break;
      case "H" :      break;

      default:
            break;
      }

      if ( addPos == 1 ) {
            rlen = rlen + len
            addPos = 0
            # print " adding: " arr[i] ":" brr[i+1] "  Running Total " rlen
      }
} # for loop

#print "reference length required (not including initial softclip) : " rlen
if ( rlen == 0 ) {
            print "*"
} else {
 print rlen
}
}
```

getrefseq.awk

Obtain reference matching sequence for read

BEGIN
{ initialise }

- Set REGION (chromosome to search for)
- Set begin position
- Set end position

awk reads reference line by line

switch

Stage 1	Stage 2	Stage 3	Stage 4
Search for > Find Region	Count bases Find Start	Count length Find End	**Print Ref** **exit**

Region not found
END
{Error eval > stderr}

Print Output read by disbamer script

Reference File format:

```
>CM000663.2 chr 1, GRCh38
NNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNN
ttccAATACCTCTGAAAAAACTGatccaaa
:
>KI270706.1 contig chr 1, GRCh38
GAATTCAGCTGAGAAGAACAGGCAAG
TAATTTAAGATTTTCTCCCCCTACgtaatt
ACTTGCTGTTTAAGGAACTAATTAAAC
:
```

region

start

end

getrefseq.awk

```
#! /usr/bin/awk -f

# Given a CIGAR region position and length this script will extract the corresponding sequence from a Reference.
#   will count matches, mismatches and deletes, from the cigar string. (Ignore I, N, P, S, H)
#
#  usage from a bash script >   # assign region, position and length to string to bash variables (cigtoRefLen.awk will
calculate length required from a  cigar string)
#                   awk -v regA="$regionD" -v posA="$positionD" -v lenA="$seqlengthD" getrefseq.awk lnkref
                        # call the awk script passing the bash variables as above
                                                # Note needs the link to the reference!
                        # read the output of the awk script into another bash variable (seqlengthRef) with bash "read var <<<"
#                       read seqrefD <<< $(awk command above)
#                       example region and position: region = "CM000668.2" region = "chr6" position = 62656
# NOTE: will need the region in the SAM/BAM to match region in the reference!! (ie chromosome name, this should be the
case if you aligned to the reference!
#                       otherwise you will need to rename - possible addition to this utility is the name conversion? )
# Search output goes to /dev/stderr (it will initially display in "less" but will dissappear, once move cursor. If redirecting
disbamer to file search output will show on screen.

# find read reference alignment sequence

BEGIN {
# ::::Get Data Passed from disbamer::::
region = regA
position = posA
seqlenreq = lenA

to_pos = position + seqlenreq
aregion = 0   # found region?
aSeq = ""
aPos = 0
stage = 1
printf("search regions: ")  > "/dev/stderr"
regCnt = 1
}
```

Cont…

getrefseq.awk

awk reads reference line by line

```
{
switch (stage) {
      case "1" : # find region
            if ( index( $1, region ))
                  { stage = "2"
                    # print region, stage
                  }
                        if ( index( $1, ">" ))
                                    {
                     printf(".R%s",regCnt)   > "/dev/stderr"  # region, stage. output to std error as print will default back to
shell script.
                     # printf(".R%s ref region:%s  search region: %s \n ",regCnt,$1, region)   > "/dev/stderr"
                     regCnt = regCnt + 1
                                    }
            break
      case "2" : # find start
            # print aPos, position
            if ((aPos + length($0)) >= position) {
                  aSeq = substr($0,(position - aPos), (length($0) - (position - aPos) + 1))
                  stage = 3
                    # printf(" found start %s, aPos %s, aSeq: %s, len %s, \n ref: %s \n",position,aPos,aSeq,length(aSeq),$0) >
"/dev/stderr"
            }
            aPos = aPos + length($0)
            break
      case "3" : # find end
            # print "3", aPos
            if ((aPos + length($0)) >= (to_pos)) {
                    # print " last aSeq b4 final append " aSeq, length(aSeq), " to_pos " to_pos, "aPos", aPos # test
                  aSeq = aSeq substr($0,1,(to_pos - aPos + 1))
                  stage =  4
            } else {
                                    # print " refseq: " aSeq, length(aSeq)  # test
                  aSeq = aSeq $0    # append current seq data to aSeq.
                  aPos = aPos + length($0)
            }
            break
      case "4" : # OUPUT print the required reference string to stdout, calling bash script uses read to collect.
                        printf("\n%s\n",aSeq)
              # print " length " length(aSeq), aPos
             exit 1   # return to disbamer found reference sequence
            break
      default : print "what? region not found...."
                        break
}
}

END { # only get here if end of file reached and region not found.
switch (stage) {
case "1" :
            printf("\n XXXX Could not find region: %s in lnk.ref, link to your reference. \n",$1)   > "/dev/stderr"
            break;
case "2" :
      printf("\n XXXX Could not find start postion: %s in region %s for lnk.ref, link to your reference. \n",position, $1)   >
"/dev/stderr"
      break;
case "3" :
      printf("\n XXXX Could not find end position: %s in region %s for lnk.ref, link to your reference. \n",to_pos,$1)   >
"/dev/stderr"
      break;
case "4" : # OK
     #  printf("\n Found region and position in reference. \n",$1)   > "/dev/stderr"
            printf("\n") > "/dev/stderr"
      break;
default : break
}
}
```

viewread.awk

Display read alongside Reference with indels marked

Input:
- Cigar String,
- Read Sequence (seq) and
- Reference sequence

```
5M1I3M2D7M2I6M
TCCCCCTACTAATTACGCTGTT
TCCCCTACAGTAATTACGGTCTT
```

Spilt Cigar String via regex into numeric and CIGAR identifier arrays

step through CIGAR arrays [CIGAR] [length]

switch CIGAR

M: Add matching sequence to results

=: Add matching sequence to results

X: Add matching sequence to results (mismatches)

D: Add cigar element length of dashes "-" to results

I: Add quoted sequence to results (these are not in Reference)

N: Add cigar element length of N's "N" to results

P: Add cigar element length of P's "P" to results

S: If first Cigar then set start of result sequence to end of clip. Otherwise increment over clip.

H: do nothing hard clip sequences are not in sam file.

Build a copy of the Read Sequence, showing inserts and deletions. Also gather statistics.
- Matches
- Deletions
- Insertions
- Clip

For [M, =, X, D] Build position string.
For I (inserts) build string with quotes.

Read is now:

```
TCCCC'C'TAC--TAATTAC'TT'GCTGTT
```

deletion

insertions

For each char in formatted read

Format Reference sub sequence to align to our read.
Build an indicator string of deletions, mismatches and additions.

': add insert indicator '+' into reference and indicator for length of insert

-: reference remains as is, indicator string has dashes

Otherwise : reference remains as is, indicator string has 'x' if reference value does not match read.

Output:

```
READ      TCCCC'C'TAC--TAATTAC'TT'GCTGTT
Reference TCCCC'+'TACAGTAATTAC'++'GGTCTT
Indicator .....'+'...--.......'++'.x.x..
Position  .........1.........2.........3
          .........0.........0.........0
```

viewread.awk

Note: All code is in a BEGIN block, and only operates on passed variable (via –v)

```
#! /usr/bin/awk -f

# Given a read sequence, cigar string and reference sequences, print out the alignment.
#
#  usage from a bash script >   # assign read sequence, cigar string and reference sequences, to awk variables from a bash script, and call awk script:
#                          awk -v cigA="$cigarD" -v seqA="$seqD" -v refA="$seqrefD" -f viewread.awk
                                # call the awk script passing the bash variables as above
# Displays The read sequence with '-' for deletes 'N' quoted base(s) for insert(s); then the reference sequence on the next line, followed by the read position on subsequenct lines.
# Pipe display to less -S to allow scrolling horizontally..
# a line wrapping enhancment is under consideration....


BEGIN {

# :::from disbamer:::
cig = cigA
seq = seqA
ref = refA
# output summary of input data:
# print "Cigar: ", cig  # $6
# print "Seq read: ", substr(seqA,1,50), "...."   # $10
# print " Seq ref: ",  substr(refA,1,50), "...."

# initialise data
regex =  "[[:upper:]]+";
n=split(cig, arr, regex);
regex =  "[[:digit:]]+";
m=split(cig, brr, regex);
pos=1
read_seq= ""
softClip = ""
hardClip = ""
totalClip = 0
startsoftclip = 1
inserts = 0
deletes = 0
matches = 0
# posStr calculates the read position, will write read position vertically underneath the sequence.
posStr = ""
posStr10 = ""
posStr100 = ""
posStr1000 = ""
posStr10000 = ""
posStr100000 = ""
posStr1000000 = ""
rlen = 1
addPos = 0
addIns = 0
```

Cont…

viewread.awk

step through CIGAR arrays
[CIGAR] [length]

```
for ( i=1; i<n; i++ ) {
        # print arr[i] ":" brr[i+1]
        len = arr[i]

        switch( brr[i+1] ) {

        case "M" : read_seq = read_seq substr(seq,pos,len)
                        pos = pos + len
                        addPos = 1
                        matches = matches + len
                        break;
    case "=" : read_seq = read_seq substr(seq,pos,len)
            pos = pos + len
            addPos = 1
            break;
    case "X" : read_seq = read_seq substr(seq,pos,len)
            pos = pos + len
            addPos = 1
            break;
        case "D" : for(c=0;c<len;c++) read_seq = read_seq "-"
            addPos = 1
                        deletes = deletes + len
                        break;
    case "I" : read_seq = read_seq "'" substr(seq,pos,len) "'"
            pos = pos + len
                        addIns = 1
                        inserts = inserts + len
            break;
    case "N" : for(c=0;c<len;c++) read_seq = read_seq "N"
            pos = pos + len
            break;
    case "P" : for(c=0;c<len;c++) read_seq = read_seq "P"
            pos = pos + len
            break;
    case "S" : softClip = softClip "S" len " "
                        if ( pos == 1 ) {
                                startsoftclip = len
                                rlen = len
                        }
            pos = pos + len
                        totalClip = totalClip + len
            break;
    case "H" : hardClip = hardClip "H" len " "
            # pos does not alter for hard clip
                        totalClip = totalClip +         len
            break;

        default:
                break;

        }
```

Loop continues on next page…

Cont…

viewread.awk

For [M, =, X, D] Build position string.
For I (inserts) build string with quotes.

```
#  printf("len %s\n",len)
if ( addPos == 1 ) {
 for ( j=rlen; j<(rlen+len); j++ ) {
            if ( j % 10 != 0 ) {
                        posStr = posStr "."
                        posStr10 = posStr10 "."
                        posStr100 = posStr100 "."
                        posStr1000 = posStr1000 "."
                        posStr10000 = posStr10000 "."
                        posStr100000 = posStr100000 "."
            posStr1000000 = posStr1000000 "."
            }
      if ( j % 10 == 0 ) {
#                       k = int(j/10)
                        k = sprintf("%06i",j)  # format K with leading zeroes
#                       print k
                        posStr = posStr substr(k,length(k),1)
            posStr10 = posStr10 substr(k,length(k)-1,1)
            posStr100 = posStr100 substr(k,length(k)-2,1)
            posStr1000 = posStr1000 substr(k,length(k)-3,1)
            posStr10000 = posStr10000 substr(k,length(k)-4,1)
            posStr100000 = posStr100000 substr(k,length(k)-5,1)
            posStr1000000 = posStr1000000 substr(k,length(k)-6,1)
            }
 }
 rlen = rlen + len
 addPos = 0
}

# mark inserts
if ( addIns == 1 ) {
 posStr = posStr """; ; posStr10 = posStr10 """;  posStr100 = posStr100 """; posStr1000 = posStr1000 """; posStr10000 =
posStr10000 """;  posStr100000 = posStr100000 """;  posStr1000000 = posStr1000000 """;
 for ( j=rlen; j<(rlen+len); j++ ) {
{
            posStr = posStr "+"
            posStr10 = posStr10 "+"
            posStr100 = posStr100 "+"
            posStr1000 = posStr1000 "+"
            posStr10000 = posStr10000 "+"
            posStr100000 = posStr100000 "+"
            posStr1000000 = posStr1000000 "+"
      }
 }
 posStr = posStr """; posStr10 = posStr10 """;  posStr100 = posStr100 """; posStr1000 = posStr1000 """; posStr10000 =
posStr10000 """;  posStr100000 = posStr100000 """;  posStr1000000 = posStr1000000 """;
 addIns = 0
}

# printf("%s \n",read_seq);    # see output grow for each cigar
# printf("%s \n",posStr);

}  # for each cigar
```

Cont…

viewread.awk

Extract from main diagram

For each char in formatted read

Format Reference sub sequence to align to our read.

Build an indicator string of deletions, mismatches and additions.

': add insert indicator '+' into reference and indicator for length of insert

-: reference remains as is, indicator string has dashes

Otherwise : reference remains as is, indicator string has 'x' if reference value does not match read.

```
# align ref to our read
k=1; j=1
refAd = ""; refMis = ""
mismatchCount = 0
 while ( j <= (length(read_seq)) ) {
            switch ( substr(read_seq,j,1) ) {
            case "'" :      refAd = refAd "'"
                  refMis = refMis "'"
                              j = j + 1
                              while ( substr(read_seq,j,1) != "'" ) {
                              refAd = refAd "+"
                                    refMis = refMis "+"
                                          j = j + 1
                              }
                              refAd = refAd "'"
                  refMis = refMis "'"
                  j = j + 1
                              break
            case "-" :        refAd = refAd (substr(refA,k,1))
                              refMis = refMis "-"
                              k = k + 1;  j = j + 1
                              break
            default : refAd = refAd (substr(refA,k,1))
                              if ( substr(read_seq,j,1) == toupper(substr(refA,k,1)) ) {
                                    refMis = refMis "."
                              } else {
                                    refMis = refMis "x"
                                    mismatchCount =mismatchCount + 1
                              }
                              j = j + 1;  k = k + 1
                        break
            }
}

# output:

print "Legend:  line 1: aligned seq. line 2: reference seq. line 3: insert +/delete -/mismatch x/: indicators. lines 4- : position in aligned sequence."
print " "
printf("1 %s \n",read_seq);
printf("2 %s \n",refAd);
printf("3 %s \n",refMis);

# printf(" %s \n",refA);

n=split(posStr1000000,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr1000000); }
n=split(posStr100000,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr100000); }
n=split(posStr10000,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr10000); }
n=split(posStr1000,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr1000); }
n=split(posStr100,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr100); }
n=split(posStr10,carr,"[1-9]")
if ( n > 1 ) { printf("  %s \n",posStr10); }
printf("  %s \n",posStr);

print "Info:"
printf(" = and X, just print seq base, P prints \"P\", N prints \"N\", D \"-\", I \"'+'\" (inserts are enclosed in '), clipping not printed. \n")
if ( hardClip == "") { hardClip = "-" }
printf("Inserts: %s, Deletes: %s, Matches: %s. (using M-I-D cigars only)\n",inserts,deletes,matches)
printf("Soft clipping: %s,  Hard clipping: %s, total clip: %s.  \nRead Length (less clipping): %s  Reference Length: %s    Miss matches : %s.          >>> thankyou. \n",softClip,hardClip,totalClip,(length(seq)-totalClip),length(ref),mismatchCount)
print " "
}
```

Note: All code is in a BEGIN block, and only operates on passed variable (via –v)