

COS 126

Deciphering the Genetic Code

Step 0: preparation

- Read the assignment in the course packet to get an overview of the problem.
- Read the assignment checklist (on the course website) for some good hints and additional information & details about what to submit.
- Read K & R: 5.5 - 5.10, 7.5, and look over appendix B3 (string library)
- Read Sedgewick: section 3.6
- Copy all necessary files from the /u/cs126/files/gene directory. (See assignment checklist)

Step 1: writing the code() function

- Start with the code given to you in `gene.c`. You will need to add to this code in three places (where you see 'YOUR CODE HERE' written in the comments.)
- The input to the `code()` function is an array of characters. Each of the characters will be one from the set {a, c, g, t}. The output (or return value) of the function is an integer between 0 and 63. For example, if the input is `aaa`, return 0. If the input is `aac`, return 1. If the input is `aag`, return 2...and, if the input is `ttg`, return 62, and finally, if the input is `ttt`, return 63.
- To implement this function you should think of the three character string as an integer represented in base 4.
- To get started, you may want to write a small function, `code_helper()` that converts a character into an integer: a = 0, c = 1, g = 2, t = 3.
- To write the function, think about how you convert a number from octal (or any other number system) into decimal.
- To debug the function, you can use the technique described in the assignment checklist.

Step 2: writing the decode() function

- The input to the `decode()` function is an integer between 0 and 63. The function does not return any value; instead, it will print the 3 characters corresponding to the integer input. (see above)
- As in the `code()` function, you may want to write a 'helper' function to make the code simpler. Write `decode_helper()`, a function which converts an integer to its corresponding character (the opposite of `code_helper()`, above.)
- There is more than 1 way to write this function correctly. You may want to consider using an array to hold the three characters so that you don't have to print the characters in the order you compute them.
- To write this function, think about how you can convert a binary number to a decimal number using the repeated-division-by-2 method.
- The checklist contains information about debugging.

Step 3: understanding the main() part of the program

- `geneseq[]` is an array that will hold the sequence of {a,c,g,t} characters you read in from the *gene* input file.
- `protseq[]` is an array that will hold the sequence of capital letters (A-Y) you read in from the *protein* input file.

- `genecode[]` is the array that you will use to keep track of the 'matches' you have made. Understanding the purpose of this array is critical to your ability to write the code. An explanation of this array follows, but if you have any questions about it, be sure to contact a preceptor or ask a lab TA to explain it to you before you begin writing any code.

The `genecode[]` array

Each entry in the `genecode` array corresponds to one 3-character amino acid encoding. The same mapping used for the `code()` and `decode()` functions applies to this array. (In fact, the purpose of those `code()` and `decode()` functions you wrote was to allow you to use them with the `genecode` array.) So, `genecode[0]` corresponds to 'aaa', `genecode[1]` corresponds to 'aac'... and `genecode[63]` corresponds to 'ttt'. The `genecode` array holds characters: the capital letters corresponding to the 25 amino acids. Whenever you store a value in the `genecode` array, you are 'matching' a 3-character amino acid encoding to that value, a particular amino acid. For example, `genecode[0] = 'T'` specifies that 'aaa' is the encoding of amino acid 'T'. `genecode[15] = 'E'` specifies that 'att' is the encoding of amino acid 'E'. What you are trying to do in this assignment is assign values to the elements in the `genecode` array so that you can print out a table that shows the matches between the 3-character encodings and the amino acids. (See `gene.1.ans` for an example.)

- The `main()` function begins by opening the *protein* and *gene* files using the `fopen()` function. The names of the files are specified as '**command line arguments**'. These concepts will be taught in precept (but you don't *really* need to understand them to complete this assignment.)
- Next, the protein sequence is read from the *protein* file into the `protseq[]` array. The code is written so that all the values in the `protseq[]` array will be upper-case characters A-Y. After the last protein character is read, the value `'\0'` is inserted in the array. This **null** character denotes the end of the protein string.
- In a similar manner, the `geneseq[]` array is also initialized.
- At the end, code is provided to print the table of amino acid encodings. The `decode()` function is used to print out each 3-character encoding. All values of the `genecode[]` array are printed. You do not need to change this code.

Step 4: writing code to test a possible alignment (BIG STEP!)

- First, initialize the `genecode[]` array so that all of its elements contain '-'. We will use this character to denote a blank.
- You will need to create some more variables...name them carefully so that you won't get them confused with each other as your code becomes more complicated. Minimally, you should declare two integer variables to keep track of your current positions in the `geneseq` and `protseq` arrays.
- To test a possible alignment, repeat the following procedure until you reach the end of the protein sequence:
 1. Look-up the current 3-character amino acid encoding in the `genecode` array.
 2. If the amino acid stored there does not match the current amino acid (in the `protseq` array), exit the loop (you can use the **break**; statement to do this.)
 3. Otherwise, if the entry in `genecode` is a blank ('-'), store the current amino acid there.
 4. Update current positions in the `geneseq` and `protseq` arrays (+3 & +1, respectively.)
- The tricky part about the above procedure is looking-up the current 3-character amino acid encoding in the `genecode` array. To do this, you will need to pass the address of the current position in the `geneseq` array as the input to the `code()` function. Use the return value as the index for the `genecode` array. You may want to work on just this part of the code alone before you try to write the whole loop. Use the `prot.1` and `gene.1` input files to test your code.
- When you exit the loop outlined above, you need to determine whether you exited because a conflict occurred (step 2) or because you reached the end of the protein sequence. If you

reached the end of the protein sequence without a conflict, you may conclude that you have found a 'match.'

- Print a statement indicating whether you found a match or encountered a conflict. If you found a conflict, print the position in the protein sequence at which the conflict occurred.
- If you initialize the variables which keep track of the current position in the protseq and geneseq arrays both to zero, you should get the following result:

```
phoeni x. Princeton. EDU% a. out prot. 1 gene. 1
conflict!! at 4
aaa - aac - aag - aat Q
aca - acc - acg - act -
aga - agc I agg - agt -
ata - atc - atg - att M
caa - cac - cag - cat -
cca - ccc - ccg - cct -
cga - cgc - cgg - cgt -
cta - ctc - ctg - ctt -
gaa - gac - gag - gat -
gca - gcc - gcg - gct S
gga - ggc - ggg - ggt -
gta - gtc - gtg - gtt -
taa - tac - tag - tat -
tca - tcc - tcg - tct -
tga - tgc - tgg - tgt -
tta - ttc - ttg - ttt -
```

The example given in the assignment instructions (in your course packet) illustrates the procedure you are implementing. Try initializing the variable which indexes the current position in the geneseq array to 2, 3, and 10 to see how your code works with the other examples. For initial value = 10, you should get the following result:

```
phoeni x. Princeton. EDU% a. out prot. 1 gene. 1
Match:
aaa - aac - aag - aat R
aca - acc - acg - act -
aga - agc M agg - agt -
ata - atc - atg M att Q
caa - cac - cag - cat -
cca - ccc - ccg - cct -
cga - cgc - cgg - cgt -
cta S ctc - ctg - ctt -
gaa - gac - gag - gat -
gca I gcc - gcg - gct H
gga - ggc - ggg - ggt -
gta - gtc - gtg - gtt -
taa - tac - tag - tat -
tca - tcc - tcg - tct -
tga - tgc - tgg - tgt -
tta - ttc - ttg - ttt -
```

Step 5: writing code to find a match

- If you haven't done so already, make a back-up copy of your code so that if your code gets too complicated in this step, you can start over with code that works for steps 1-4.
- At the end of step 4, you changed the starting position of the geneseq array index by editing your code to initialize the index variable to a different value. You could find the match by repeatedly incrementing this variable by 1 and running your code to test for a match. With the simple prot.1 and gene.1 input files, you would need to run your program 11 times to discover that the match was located at position ten. Obviously, this approach is not the best solution,

because it's a lot of work for you. In this part of the assignment, you will modify your code so that your *program* will repeatedly check for a match, incrementing the starting position in the *geneseq* array by 1, until a match is found.

- So, you will need to create a loop in which you repeat all the code you wrote in step 4. Determine the conditions under which you will want to exit the loop carefully so that your program won't crash in the case in which there is no match between the two sequences.
- Don't forget to print out at which position the match occurred!
- To help you debug your program, here are the correct outputs for all of the sample data you have been given (results for prot.1/gene.1 given above):

```
phoeni x. Pri ncet on. EDU% a. out prot. 2 gene. 2
Match occurred at position: 1687
aaa K    aac N    aag K    aat N
aca T    acc T    acg T    act T
aga R    agc S    agg -    agt S
ata I    atc I    atg M    att I
caa Q    cac H    cag Q    cat H
cca P    ccc P    ccg P    cct P
cga R    cgc R    cgg R    cgt R
cta L    ctc L    ctg L    ctt L
gaa E    gac D    gag E    gat D
gca A    gcc A    gcg A    gct A
gga G    ggc G    ggg G    ggt G
gta V    gtc V    gtg V    gtt V
taa -    tac Y    tag -    tat Y
tca S    tcc S    tcg S    tct S
tga -    tgc C    tgg W    tgt C
tta L    ttc F    ttg L    ttt F
```

```
phoeni x. Pri ncet on. EDU% a. out prot. 3 gene. 3
Match occurred at position: 336
aaa K    aac N    aag K    aat N
aca T    acc T    acg T    act T
aga -    agc S    agg R    agt S
ata I    atc I    atg M    att I
caa Q    cac H    cag Q    cat H
cca P    ccc P    ccg P    cct P
cga R    cgc R    cgg R    cgt R
cta L    ctc L    ctg L    ctt L
gaa E    gac D    gag E    gat D
gca A    gcc A    gcg A    gct A
gga G    ggc G    ggg G    ggt G
gta V    gtc V    gtg V    gtt V
taa -    tac Y    tag -    tat Y
tca S    tcc S    tcg S    tct S
tga -    tgc C    tgg W    tgt C
tta L    ttc F    ttg L    ttt F
```

```
phoeni x. Pri ncet on. EDU% a. out prot. 4 gene. 4
Match occurred at position: 31864
aaa K    aac N    aag K    aat N
aca T    acc T    acg T    act T
aga R    agc S    agg -    agt S
ata I    atc I    atg M    att I
caa Q    cac H    cag Q    cat H
cca P    ccc P    ccg P    cct -
cga R    cgc R    cgg R    cgt R
cta L    ctc L    ctg L    ctt L
gaa E    gac D    gag E    gat D
gca A    gcc A    gcg A    gct A
```

gga G	ggc G	ggg G	ggt G
gta V	gtc V	gtg V	gtt V
taa -	tac Y	tag -	tat Y
tca S	tcc S	tcg S	tct S
tga -	tgc C	tgg W	tgt C
tta L	ttc F	ttg L	ttt F

Please contact Lisa Worthington <lworthin@cs.princeton.edu> if you find any typos/bugs in these instructions.