CS19003: Programming & Data Structure Lab, Section 15
Autumn 2020-21
Lab Test 3, March 4, 2021

PART - 2

Time: 10-40 am to 11-50 am

Instructions (Read carefully)

1. **Your C programs must first contain comments with your name, roll no., and Labtest no. (=3), as done in class.**
2. **Name your C file LT3_2_<your roll no>.c for (For example LT3_2_20ME30006.c)**
3. **Submit through the links (Intermediate and Final) for PART 2 in moodle. MAKE SURE TO VERIFY YOUR SUBMISSION after final submission.**
4. **(Very Important) There are two sets of problems. Problem set 1 is for 100% marks and covers the entire syllabus of what is covered in the lab. Problem set 2 is for 60% marks and covers only up to arrays and functions, plus searching/sorting. You are free to choose which set you will attempt. However, there cannot be any mixing between the sets, you can answer only one of the sets. Submit only one C file for one set.**
5. **All other instructions regarding lab test sent earlier in the slides to be followed strictly.**

PROBLEM SET 1 (FOR 100% (= 10) MARKS)

The decimal number that you normally work with is a base-10 system with 10 digits, 0 to 9. Consider a base-12 number system with the 12 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, and B. Decimal equivalent of A is 10 and that of B is 11. The decimal equivalent of a base-12 number can be found by multiplying each digit with appropriate powers of 12 depending on its position. For example, the decimal equivalent of the base-12 number 8A5B is $8\times12^3 + 10\times12^2 + 5\times12^1 + 11\times12^0 = 15335$. Similarly, the decimal equivalent of the base-12 number A76 is $10\times12^2 + 7\times12^1 + 6\times12^0 = 1530$. So the method is the same as that of computing the value of a base-10 number, just we use powers of 12 instead of powers of 10, and we have to take care of the two extra digits represented by the symbols A and B.

Adding two base-12 numbers is also similar. We will denote a base-12 number p as p_{12} (the subscript 12 is just a notation to indicate that the number is in base-12 form). The addition process is the same as for decimal number, and we add digits to get the sum digit and maybe a carry. Some examples of single digit additions are $1_{12} + 4_{12} = 5_{12}$ (no carry), $4_{12} + 5_{12} = 9_{12}$ (no carry), $7_{12} + 3_{12} = A_{12}$ (no carry), $9_{12} + 2_{12} = B_{12}$ (no carry), $5_{12} + 8_{12} = 11_{12}$ (carry generated), $8_{12} + 6_{12} = 12_{12}$ (carry generated), $A_{12} + B_{12} = 19_{12}$ (carry generated), $B_{12} + B_{12} = 1A_{12}$ (carry generated) etc. For multi-digit numbers the carry is added to the next digit sum the same way as for decimal numbers. So $24_{12} + 8A_{12} = B2_{12}$ ($A_{12} + 4_{12} = 12_{12}$, so the carry 1 is added to the next digits' sum to get $8_{12} + 2_{12} + 1_{12} = B_{12}$), $A3B_{12} + 8AB_{12} = 172A_{12}$ etc.

Write a C function

int FindSum(char *num, int k, ….)

that takes as parameters a null-terminated string **num** containing a base-12 number (i.e., if the number is say $24A5B_{12}$, then **num[0]** = '2', **num[1]** = '4', **num[2]** = 'A', **num[3]** = '5', **num[4]** = 'B', **num[5]** ='\0') and an integer **k** (**k** < 6). The function computes the sum (in decimal) of the **distinct 2nd largest** and **distinct 3rd largest** base-12 numbers formed by a continuous sequence of **k**-digits in the number **num,** if they exist. By distinct, we mean that if there are two or more same largest numbers, none of them counts as the distinct 2nd largest, and similarly, if there are two or more same 2nd largest numbers, none of them counts as the distinct 3rd largest. For example, if **num** = B4B4A31A31 and **k** = 2, then the distinct largest is B4, distinct 2nd largest is A3, and the distinct 3rd largest is 4B. Note that there are two numbers with the same value B4, but the 2nd largest has to be distinct, so we do not consider B4 as distinct 2nd largest also. Similarly, there are two numbers with the same value A3, but the 3rd largest has to be distinct, so we do not consider A3 as distinct 3rd largest also. Also note that

a. Distinct 2nd largest and 3rd largest numbers may not even exist. For example, if **num** = 99999 and **k** =3, there is not even a distinct 2nd largest number. Similarly, if **num** = ABBBB, and **k** = 3, distinct largest is BBB, distinct 2nd largest is ABB, and there is no distinct 3rd largest number.
b. The distinct largest, 2nd largest, and the 3rd largest numbers may have overlapping digits. For example, if **num** = BBA987, and **k** = 2, the distinct largest is BB, the distinct 2nd largest is BA (overlapping with the largest), and the distinct 3rd largest is A9 (overlapping with the 2nd largest).

The function should return (as return value) 1 if there exists distinct 2nd largest and 3rd largest such numbers, 0 otherwise. In case it returns 1, it should also return the distinct 2nd largest and 3rd largest numbers in base-12 form, and their sum in decimal (add appropriate parameters to the function for returning these).

The function should (i) find the distinct 2nd largest and 3rd largest numbers in base-12 form as above without converting anything to decimal (2 marks penalty if you do), (ii) add the two base-12 numbers directly, without converting any of the numbers to decimal (2 marks penalty if you convert to decimal and then add), and (iii) convert the sum to decimal and return it. You can use string functions and additional arrays if you want. You cannot use any parameter/variable of type pointer to pointer (ex. **char **p**) even if you know how to use it. Do not write any other functions.

Write a **main( )** function that does the following **in exactly this order**:

1. Read in the number of the digits **n** of a base-12 number (**n** < 25)
2. Read **n** characters of the base-12 number one by one from the keyboard in a loop (do not read together by %s). If any of the characters entered is not a valid base-12 digit, the digit should be entered again.
3. After reading all **n** digits, null-terminate the string.
4. Print the number read.
5. Read in an integer **k** (**k** < 6).

6. Call the function **FindSum( )** to find the sum of the distinct 2nd largest and 3rd largest base-12 number formed by a continuous sequence of **k**-digits in the number **num**.
7. Print the sum, as well as the distinct 2nd largest and 3rd largest base-12 numbers returned. In case there are no distinct 2nd largest and 3rd largest numbers, print an appropriate message.
8. You cannot use any variable of type pointer to pointer (ex. **char **p**) even if you know how to use it.

Example:

num = A55A95A95B90AB **k** = 3

Your program should output (exact messages can vary):

The 2nd and 3rd largest numbers in base-12 form are A95 and A55
The sum in decimal is 3058

Another example:

num = A999999 **k** = 3

Your program should output (exact messages can vary):

There are no distinct 2nd and 3rd largest numbers

PROBLEM SET 2 (FOR 60% (= 6) MARKS)

Write a C function

int FindLargest(char P[ ], int k)

that takes as parameter a null-terminated string **P** containing only the digits 0 to 9, and an integer **k**. The function returns the largest integer (in decimal) of **k** digits that can be formed from the digits in **P**. The digits in the largest integer need not be distinct, if a digit is repeated in **P**, it can also be repeated in the largest integer of **k** digits formed out of **P**. You can assume that **k** is less than the length of **P** (so **P** has more digits than **k**), and **k** is less than 6. You can use additional arrays, but you cannot sort the entire **P** array (or any copy of it made in any other array).

Write a C function

int IsSubReversed(char P[ ], char Q[ ])

that takes as parameter two null-terminated strings **P** and **Q** containing only digits from 0 to 9. The function returns 1 if any substring of **P** is the reverse of **Q**, 0 otherwise. A substring of a string is any sequence of contiguous characters in the string (including the string itself). For example, if **P** = "1234" and **Q** = "4321", then a substring of **P** (the string itself) is the reverse of **Q**. Similarly, if **P** = "1238631" and **Q** = "683", then a substring of **P** ("386") is the reverse of **Q**. However, if **P** = "1653" and **Q** = "43", then no substring of **P** is the reverse of **Q**.

Write a **main** function that does the following:

1. Reads in two strings from the keyboard using %s in two arrays **X** and **Y**. You can assume that the length of the strings is less than 100 and that all characters in the strings will be digits between 0 to 9.
2. Reads in an integer **k** (**k** < 6)..
3. Calls **FindLargest( )** to find and print the largest integer of size **k** that can be formed from the digits in **X**.
4. Calls **IsSubReversed( )** to find if any substring of **X** is the reverse of **Y**. You should print an appropriate message depending on the return value of the function.

Example:

X = "1325413", **Y** = "92435", **k** = 4

Then your program should print (exact messages may vary)

The largest integer with 3 digits in X is 5433
There is no substring in X that is reverse of Y

Another Example:

X = "131625", **Y** = "613", **k** = 4

Then your program should print (exact messages may vary)

The largest integer with 4 digits in X is 6532
There is a substring in X that is reverse of Y