

SC1003 Review Lecture Week 9





Review Lecture – Week 9

- **Week 9 – Learning Materials**
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
 - Coding in APAS
- Reviews on Functions and Pointers
- Examples



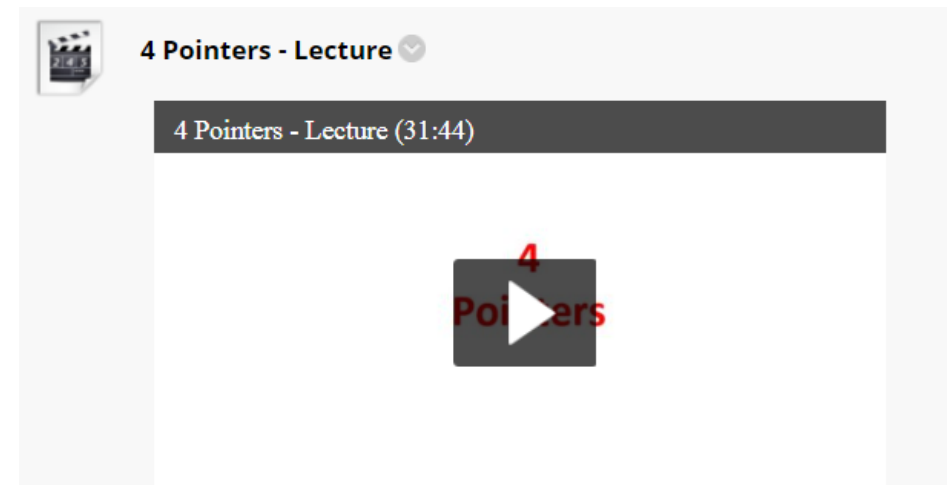
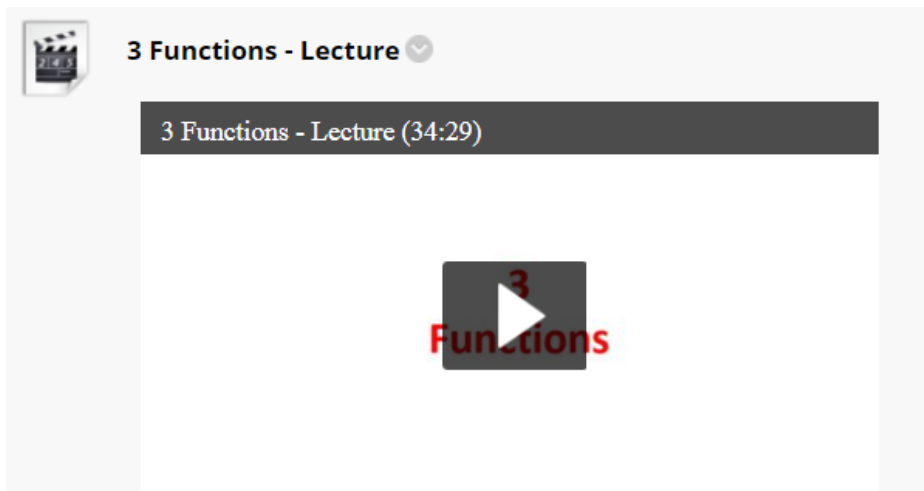
Learning Schedule (Week 8 – Week 13)

Week	Week 8 4 Oct	Week 9 11 Oct	Week 10 18 Oct	Week 11 25 Oct	Week 12 1 Nov	Week 13 8 Nov	Week 14 15 Nov
Topics	Basic C Programming and Control Flow	Functions and Pointers	Arrays	Character Strings	Structures		
Review Lecture	Date: 4 Oct 2021 (Monday) Time: 9:30am-10:30am Online: MS Teams (the link for the online lecture is given at the end of the table)	Date: 11 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 18 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 25 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 1 Nov 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)		Lab Test (MCQ Test & Coding Test) Dates: 15 Nov (Mon) and 16 Nov (Tue) Details will be announced when confirmed.
e-Learning Lectures	Learn: Course Introduction Learn: (1) Basic C Programming; (2) Control Flow	Learn: (1) Functions and (2) Pointers	Learn: (1) 1-D Arrays and (2) 2-D Arrays	Learn: Character Strings	Learn: Structures		
Lab-Tutorial	Learn: CodeBlocks IDE Do: Lab-Tutorial 1 (Qns are also available in APAS)	Do: Lab-Tutorial 2 (Qns are also available in APAS)	Do: Lab-Tutorial 3 (Qns are also available in APAS)	Do: Lab-Tutorial 4 (Qns are also available in APAS)	Do: Lab-Tutorial 5 (Qns are also available in APAS)		
Practice Questions	Learn: using APAS system Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)		
Assignment	Learn: (1) Assignment Submission and Grading process; (2) Review Request Form (Procedure)		Assignment paper – Available in APAS			Assignment due	



Lecture Video – Functions and Pointers

- **Watch Lecture Video – Functions and Pointers**
(NTULearn: C Programming > E-Learning Lectures > Week 9)



Week 9 - Lab

- **Lab 2 – Functions and Pointers**

(NTULearn: C Programming > Lab-Tutorials > Lab-Tutorial 2)

Lab 2 – Functions and Pointers

Lab session – One hour is scheduled for the lab session. There are 3 questions in this lab.

Note: You do not need to submit your code for this lab.

Lab Questions 1-3

You may use the program template in Figure 1 to test your functions in the following three questions. The program contains a **main()** which includes a switch statement so that the following functions can be tested by the user. Write the code for each function and use the suggested test cases to test your code for correctness.

```
#include <stdio.h>
/* function prototypes */
```

Suggested solutions:

Available in the same folder.
You may refer to the suggested code if you have any difficulty in attempting the lab questions.

Lab Questions in APAS:

1. numDigits
2. digitPos
3. square

Available at: APAS > Exercise (choose Topic): You may test your code with sample test cases in APAS.



Week 9 - Tutorial

- Tutorial 2 – Functions and Pointers**

(NTULearn: C Programming > Lab-Tutorials > Lab-Tutorial 2)

Tutorial 2 – Functions and Pointers

1. Assume the following declaration:

```
int number;  
int *p;
```

Assume also that the address of number is 7700 and the address of p is 3478. That is,

3478	<input type="text"/>	p
	.	
	.	
	.	
7700	<input type="text"/>	number

Tutorial Coding Questions:

1. calDistance

Suggested solutions:

Available at the end of each week in the same folder in NTULearn.

Available at: APAS > Exercise (choose Topic): You may test your code with sample test cases in APAS.



LAMS MCQ Questions

- **LAMS MCQ Questions – Functions and Pointers**
(NTULearn: C Programming > LAMS MCQ Questions > Functions)

LAMS LAMS MCQ Practice Questions - Functions

Watch the lecture video and read the lecture notes before attempting MCQ practice questions are for exercise only.

Functions

Q1

What will be the output of the program?

```
#include <stdio.h>
void fun(int p);
int main()
{
    int a=1;
    fun(a);
    printf("%d\n",a);
    return 0;
}
```

LAMS LAMS MCQ Practice Questions - Pointers

Watch the lecture video and read the lecture notes before attempting MCQ practice questions are for exercise only.

Pointers

Q1

What will be the output of the program?

```
#include <stdio.h>
int main()
{
    int a=5,b=10,c=15,s;
    int *px,*py,*pz,*ps;
    px=&a; py=&b; pz=&c;
    ps=&s;
    *ps=*px;
    if (*ps<*py) *ps=*py;
    if (*ps<*pz) *ps=*pz;
    printf("%d\n",s);
    return 0;
}
```

- Answers and explanations on each question are available in the same folder.



APAS - Coding Practice Questions

- Coding Practice Questions – Functions and Pointers**

(APAS: Quiz > Functions and Pointers)

- | | |
|--------------------|--------------------|
| 1. computePay | 11. countOddDigits |
| 2. computeSalary | 12. allOddDigits |
| 3. divide | 13. extEvenDigits |
| 4. power | 14. extOddDigits |
| 5. gcd | 15. reverseDigits |
| 6. perfectProd | |
| 7. digitValue | |
| 8. sumSqDigits | |
| 9. countEvenDigits | |
| 10. allEvenDigits | |

Suggested solution can be found at (need VPN is accessing from outside NTU):

<http://172.21.147.174/>NTUQA>





APAS – Automated Grading of Submission Code

- The answer code can be submitted to APAS and marked automatically. The grading is achieved through the checking of **input/output data** from **test cases** with the “**exact string matching**” technique.
- Please note that the **sample test cases** (used in **pretest**) are used for **testing only**. No scores will be assigned to the correctness of running sample test cases.
- We use “**hidden**” **test cases** for the grading of your program as this is to ensure that your program code will not be hard-coded to achieve the desired output results.
- Note that the final score is computed according to the correctness of the programs based on “**hidden**” test cases, **NOT sample test cases**.
- Generally, if your code is not hard-coded, and your code is able to pass the pretest, it should also be able to pass the hidden test cases.





Notes on Coding in APAS

1. Do not change the **given code** in the program template.
2. Each assignment question will provide you with the (i) problem specification, (ii) program template and (iii) sample input/output sessions. These are the problem requirements. You should follow exactly the problem requirements (especially the **program input and output data format**) to build your programming code for the question.
3. A simple mistake on letter case will cause your program to be marked as incorrect in APAS as we use “**exact string matching**” technique based on program **input/output** for checking the correctness of the program.
4. “**TIMEOUT**” error message in APAS – it occurs (1) when the program waits for user input and no input has occurred during runtime; (2) when the program runs in an infinite loop. Therefore, you need to check your program code when this error occurs.
5. Also note that there is **NO need to do user input error checking** in your program unless it is explicitly stated in the question requirement.





Review Lecture – Week 9

- Week 9 – Learning Materials
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
 - Coding in APAS
- **Reviews on Functions and Pointers**
- Examples



Python vs C – Defining a Function

Python	C
<p>A Python function definition has the form:</p> <pre>def function-name(formal-parameter-list): body</pre> <p>Example:</p> <pre>def disc(a, b, c): return b * b - 4 * a * c</pre> <ul style="list-style-type: none">• If there are no parameters, an empty list <code>()</code> is used.• The body is delimited by indentation and can be any number of lines.• A function does not have to return a value; if it does not, no return statement is required, though return without a value (or with <code>None</code>) can be used to end the function execution early.	<p>A C function definition has the form:</p> <pre>type function-name(formal-parameter-list) { body }</pre> <p>Example:</p> <pre>double disc(double a, double b, double c) { return b * b - 4 * a * c; }</pre> <ul style="list-style-type: none">• The return type of a C function must always be specified. If it does not return a value, void must be used.• If there are no parameters, an empty list <code>()</code> is used.• A type must be specified for each of the parameters.• The body is always delimited by curly braces <code>{ , }</code> even if the body is a single line.• Indentation is ignored by the compiler (though stylistically it is still highly desirable for the benefit of a human reader).• If the function does not return a value (its type is void), no return statement is required, though return without a value can be used to end the function execution early.





Python vs C – Calling a Function

Python	C
<p>A Python function is called by specifying its name followed by a list of actual parameters (that must be the same length as the list of formal parameters).</p> <p>Example:</p> <pre>d = disc(1, 2, 1) print(d)</pre>	<p>A C function is called by specifying its name followed by a list of actual parameters (that must be of the same length as the list of formal parameters).</p> <p>Example:</p> <pre>double d; d = disc(1, 2, 1); printf("%f",d);</pre>



Python vs C – An Example Function

Python	C
<pre>from math import * def printRootTable(n): for i in range(1,n): print("{:2d}{:7.3f}".format(i, sqrt(i))) def main(): printRootTable(10) main()</pre>	<pre>#include <stdio.h> #include <math.h> void printRootTable(int n); // function prototype int main() { printRootTable(10); return 0; } void printRootTable(int n) { int i; for (i=1; i<=n; i++) { printf("%2d %7.3f\n", i, sqrt(i)); } }</pre>



C – Pointers

What is a pointer?

A **pointer** is a variable which contains the **address** (4 bytes) in memory of another variable. We can have a pointer to any variable type.

Example:

```
int i, j;  
int *iptr = &i, *jptr = &j;    // The * indicates variables of type ( pointer to int )  
i = 42;                          // changes i to 42  
*iptr = 100;                    // changes i from 42 to 100  
j = *iptr;                      // Now j is 100 also
```

// Note carefully the distinction between the following two lines:

```
*jptr = *iptr;    // Equivalent to j = i. jptr and iptr still point to different places  
jptr = iptr;      // Makes jptr point to the same location as iptr. Both now point to i
```



Call by Value vs Call by Reference – Function Communication

Call by Value

Calling function:

```
int main() {  
    int num=5; ... num=add1(num); ...  
}
```

num 5

Called function:

```
int add1(int value) { ... }
```

value 5

[**num** -actual parameter; **value** - formal parameter]

- In this parameter passing method, values of **actual parameters** are **copied** to function's **formal parameters** and the two types of parameters are stored in different memory locations.
- So any changes made inside functions are not reflected in actual parameters of the caller.

Call by Reference

Calling function:

```
int main() {  
    int num=5; ... add2(&num); ...  
}
```

num 5

Called function:

```
void add2(int *ptr) { ... }
```

ptr 5

ptr contains the address of variable num in main()

[**num** - actual parameter; **ptr** - formal parameter]

- Both the **actual** and **formal parameters** refer to the same locations.
- So any changes made inside the function are actually reflected in **actual** parameters of the caller.

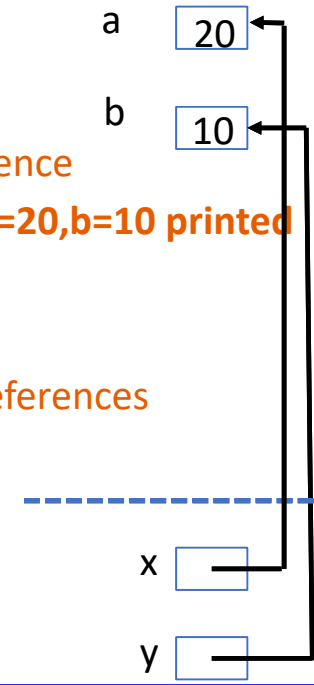


Call by Value vs Call by Reference - Differences

Call by Value	Call by Reference
<ol style="list-style-type: none">1. While calling a function, we pass <u>values</u> of variables to it. Such functions are known as Call by Value.2. In this method, the value of each variable in calling function is <u>copied</u> into the corresponding variables of the called function.3. With this method, the changes made to the variables in the called function have no effect on the values of actual variables in the calling function.4. In call by values we cannot alter the values of actual variables through function calls.5. Values of variables are passed by simple technique.	<ol style="list-style-type: none">1. While calling a function, instead of passing the values of variables, we pass <u>address</u> of variables (location of variables) to the function known as Call by Reference.2. In this method, the address of actual variables in the calling function are <u>copied</u> into the pointer variables of the called function.3. With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.4. In call by reference we can alter the values of variables through function calls.5. <u>Pointer variables</u> are necessary to define to store the address values of variables.



Call by Value vs Call by Reference - Example

Call by Value	Call by Reference
<pre>#include <stdio.h> void swap(int x, int y); int main() { int a = 10, b = 20; swap(a, b); // Call by Values printf("a=%d b=%d\n", a, b); // a=10,b=20 printed return 0; } // Swap functions that swaps two values void swap(int x, int y){ int t; t = x; x = y; y = t; printf("x=%d y=%d\n", x, y); }</pre> <p>Output: x=20 y=10 a=10 b=20</p> <div>[Thus, actual values of a and b remain unchanged even after exchanging the values of x and y.]</div>	<pre>#include <stdio.h> void swap(int*, int*); int main() { int a = 10, b = 20; swap(&a, &b); // Call by reference printf("a=%d b=%d\n", a, b); // a=20,b=10 printed return 0; } // Function to swap two variables by references void swap(int* x, int* y){ int t; t = *x; *x = *y; *y = t; printf("x=%d y=%d\n", *x, *y); }</pre>  <p>Output: x=20 y=10 a=20 b=10</p> <div>[Thus, actual values of a and b get changed after exchanging values of x and y.]</div>





Review Lecture – Week 9

- Week 9 – Learning Materials
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
 - Coding in APAS
- Reviews on Functions and Pointers
- **Examples**



Example 0 – LAMS MCQ Q10 on Pointers

```
#include <stdio.h>
int s(int *p);
int main()
{
    int a=0,i,*p,sum;
    p=&a;
    for (i=0; i<3; i++)
    {
        scanf("%d",p);    // no need to use & op
        sum=s(p);
        printf("%d ",sum);
    }
    return 0;
}
int s(int *p)
{
    int sum=10;
    sum=sum+*p;
    return (sum);
}
```

main():

sum

a

s():

p

Assume the input values **1,3,5** are entered during program execution, what will be the output of the program?

- A. 10 12 14
- B. **11 13 15**
- C. 12 14 16
- D. 13 15 17

Answer: B



Example 1 – computeSalary

Write a C program that determines the gross pay for each employee in a company.

- The company pays “straight-time” for the first 160 hours worked by each employee for four weeks and pays “time-and-a-half” for all hours worked in excess of 160 hours.
- You are given a list of employee Ids (an integer), the number of hours each employee worked for the four weeks, and the hourly rate of each employee.
- The program should input this information for each employee, then determine and display the employee’s gross pay.
- The sentinel value of -1 is used for the employee *id* to indicate the end of input.

Your program should include three functions, apart from the **main()** function, to handle the **input**, and the **computation** of the gross pay.

Program Input and Output:

Enter ID (-1 to end):

11

Enter number of hours:

155

Enter hourly pay rate:

8

computeSalary1(): ID 11 grossPay 1240.00

computeSalary2(): ID 11 grossPay 1240.00

Enter ID (-1 to end):

12

Enter number of hours:

165

Enter hourly pay rate:

8

computeSalary1(): ID 12 grossPay 1340.00

computeSalary2(): ID 12 grossPay 1340.00

Enter ID (-1 to end):

-1



Example 1 – Suggested Code

Python: main()

```
def main():
    id, noOfHours, payRate = readInput()
    while (id != -1):
        print("computeSalary(): ", end="");
        grossPay = computeSalary(noOfHours, payRate)
        print("ID {:d} grossPay {:.2f}".format(id, grossPay))
        id, noOfHours, payRate = readInput()

main()
```

C: main()

```
#include <stdio.h>
void readInput(int *id, int *noOfHours, int *payRate);
double computeSalary1(int noOfHours, int payRate);
void computeSalary2(int noOfHours, int payRate, double *grossPay);
int main()
{
    int id = -1, noOfHours, payRate;
    double grossPay;
    readInput(&id, &noOfHours, &payRate);
    while (id != -1) {
        printf("computeSalary1(): ");
        grossPay = computeSalary1(noOfHours, payRate);
        printf("ID %d grossPay %.2f \n", id, grossPay);
        printf("computeSalary2(): ");
        computeSalary2(noOfHours, payRate, &grossPay);
        printf("ID %d grossPay %.2f \n", id, grossPay);
        readInput(&id, &noOfHours, &payRate);
    }
    return 0;
}
```



Example 1 – Suggested Code

Python:

```
def readInput():
    id = int(input("Enter ID (-1 to end): \n"))
    if (id != -1):
        noofHours = int(input("Enter number of hours: \n"))
        payRate = int(input("Enter hourly pay rate: \n"))
    else:
        noofHours = 0
        payRate = 0
    return id, noofHours, payRate
def computeSalary(noOfHours, payRate):
    if (noOfHours > 160):
        grossPay = 160 * payRate
        noOfHours -= 160
        if (noOfHours > 0):
            grossPay += noOfHours * 1.5 * payRate
    else:
        grossPay = noOfHours * payRate
    return grossPay
```

C:

```
void readInput(int *id, int *noOfHours, int *payRate)
{
    printf("Enter ID (-1 to end): \n"); scanf("%d", id);
    if (*id != -1) {
        printf("Enter number of hours: \n");
        scanf("%d", noOfHours);
        printf("Enter hourly pay rate: \n");
        scanf("%d", payRate);
    }
}
```

Note – no need to use & op in scanf() here)

Aim - Using Call by Reference to pass more than one input values to the main() function.



Example 1 – Suggested Code

(i) Using Call by value

```
double computeSalary1(int noOfHours, int payRate)
{
    double grossPay;
    if (noOfHours > 160)
    {
        grossPay = 160 * payRate;
        noOfHours -= 160;
        if (noOfHours > 0)
            grossPay += noOfHours * 1.5 * payRate;
    }
    else
        grossPay = noOfHours * payRate;
    return grossPay;
}
```

(ii) Using Call by reference

```
void computeSalary2(int noOfHours, int payRate, double *grossPay)
{
    if (noOfHours > 160)
    {
        *grossPay = 160 * payRate;
        noOfHours -= 160;
        if (noOfHours > 0)
            *grossPay += noOfHours * 1.5 * payRate;
    }
    else
        *grossPay = noOfHours * payRate;
}
```





Example 2 – power

Write a function that computes the power p of a positive number num . The power may be any integer value. Write two iterative versions of the function. The function **power1()** returns the computed result, while **power2()** passes the result through the pointer parameter **result**. In this question, you should not use any functions from the standard math library.

The function prototypes are given below:

```
float power1(float num, int p);  
void power2(float num, int p, float *result);
```

Program input and output

Enter the number and power:

2 3

power1(): 8.00

power2(): 8.00

Enter the number and power:

2 -4

power1(): 0.06

power2(): 0.06



Example 2 – Suggested Code

Python:

```
def power(num, p):
    result = 1.0;
    if (p == 0):
        return 1
    elif (p < 0):
        while (p):
            result *= 1.0/num
            p += 1
    else:
        while (p):
            result *= num
            p -= 1
    return result
def main():
    print("Enter the number and power: ")
    num,p = map(int, input().split())
    result = power(num, p)
    print("power(): {:.2f}".format(result))
main()
```

C: main()

```
#include <stdio.h>
float power1(float num, int p);
void power2(float num, int p, float *result);
int main()
{
    int power; float number, result=-1;
    printf("Enter the number and power: \n");
    scanf("%f %d", &number, &power);
    printf("power1(): %.2f\n", power1(number, power));
    power2(number,power,&result);
    printf("power2(): %.2f\n", result);
    return 0;
}
```



Example 2 – Suggested Code

(i) Using Call by Value

```
float power1(float num, int p)
{
    float result=1;
    if (p == 0)
        return 1;
    else if (p < 0)
        while (p++)
            result *= 1.0/num;
    else
        while (p--)
            result *= num;
    return result;
}
```

(ii) Using Call by reference

```
void power2(float num, int p, float *result)
{
    *result = 1;
    if (p == 0)
        *result = 1;
    else if (p < 0)
        while (p++)
            *result *= 1.0/num;
    else
        while (p--)
            *result *= num;
}
```

main():

result

3

power2():

result





Example 3 – digitValue

Write a function that returns the value of the k^{th} digit ($k > 0$) from the **right** of a non-negative integer n . For example, if $n=1234\mathbf{5}67$ and $k=\mathbf{3}$, the function will return 5 and if $n=1234$ and $k=8$, the function will return **0** when the digit position k is not found in n . Write the function in two versions.

The prototypes of the function are given below:

```
int digitValue1(int num, int k);  
void digitValue2(int num, int k, int *result);
```

Program input and output

```
Enter a number: 1284567  
Enter the digit position: 3  
digitValue1(): 5
```

```
Enter a number: 1234  
Enter the digit position: 8  
digitValue2(): 0
```

Example 3 – Suggested Code

Python:

```
def digitValue(num, k):
    for i in range(k):
        r = num%10
        num //= 10
    return r

def main():
    num = int(input("Enter the number: "))
    digit = int(input("Enter the digit position: "))
    result = digitValue(num, digit)
    print("digitValue():", result)

main()
```

C: main()

```
#include <stdio.h>
int digitValue1(int num, int k);
void digitValue2(int num, int k, int *result);
int main()
{
    int num, digit, result;
    printf("Enter the number: \n");
    scanf("%d", &num);
    printf("Enter the digit position: \n");
    scanf("%d", &digit);
    printf("digitValue1(): %d\n", digitValue1(num, digit));
    digitValue2(num, digit, &result);
    printf("digitValue2(): %d\n", result);
    return 0;
}
```



Example 3 – Suggested Code

(i) Using Call by Value

```
int digitValue1(int num, int k)
{
    int i, r;

    for (i=0; i<k; i++){
        r = num%10;
        num /= 10;
    }
    return r;
}
```

e.g. num=1234567, k=3

In the loop:

- $i=0, r=1284567\%10 = 7$
 $num=1284567/10=128456$
- $i=1, r=128456\%10 = 6$
 $num=128456/10=12845$
- $i=2, r=12845\%10 = 5$
 $num=12845/10=1284$

• Exit loop

Return 5

(ii) Using Call by Reference

```
void digitValue2(int num, int k, int *result)
{
    int i, r;

    for (i=0; i<k; i++)
    {
        r = num%10;
        num /= 10;
    }
    *result = r;
}
```





Example 4 – countEvenDigits

Write a C function to count the number of even digits, i.e. digits with values 0,2,4,6,8 in a positive number (>0). For example, if number is 1234567, then 3 will be returned. Write the function in two versions. The function countEvenDigits1() returns the result, while the function countEvenDigits2() returns the result via the pointer parameter, count.

The function prototypes are given below:

```
int countEvenDigits1(int number);  
void countEvenDigits2(int number, int *count);
```

Program input and output

Enter a number:

1234567

countEvenDigits1(): 3

countEvenDigits2(): 3

Enter a number:

1357

countEvenDigits1(): 0

countEvenDigits2(): 0



Example 4 – Suggested Code

Python:

```
def countEvenDigits(num):
    count = 0;
    while (num != 0):
        if ((num%10)%2 == 0):
            count += 1;
        num //= 10
    return count

def main():
    num = int(input("Enter the number: "))
    result = countEvenDigits(num)
    print("countEvenDigits():", result)

main()
```

C: main()

```
#include <stdio.h>
int countEvenDigits1(int number);
void countEvenDigits2(int number, int *count);
int main()
{
    int number, result;
    printf("Enter a number: \n");
    scanf("%d", &number);
    printf("countEvenDigits1(): %d\n", countEvenDigits1(number));
    countEvenDigits2(number, &result);
    printf("countEvenDigits2(): %d\n", result);
    return 0;
}
```



Example 4 – Suggested Code

(i) Using Call by Value

```
int countEvenDigits1(int number)
{
    int count = 0;
    while (number != 0) {
        if ((number%10)%2 == 0) {
            count += 1;
        }
        number/=10;
    }
    return count;
}
```

(ii) Using Call by Reference

```
void countEvenDigits2(int number, int *count)
{
    *count = 0;
    while (number != 0)
    {
        if ((number%10)%2 == 0) {
            *count += 1;
        }
        number/=10;
    }
}
```

Example 5 – extEvenDigits

Write a function that extracts the even digits from a positive number, and combines the even digits sequentially into a new number. The new number is returned to the calling function. If the input number does not contain any even digits, then the function returns -1. For example, if the input number is 1234567, then 246 will be returned; and if the input number is 13, then -1 will be returned. You do not need to consider the input number such as 0, 10, 3033, etc. Write the function in two versions. The function extEvenDigits1() returns the result to the caller, while the function extEvenDigits2() returns the result through the pointer parameter, result.

The function prototypes are given as follows:

```
int extEvenDigits1(int num);  
void extEvenDigits2(int num, int *result);
```

Program input and output

Enter a number:

1234

extEvenDigits1(): 24

extEvenDigits2(): 24

Enter a number:

1357

extEvenDigits1(): -1

extEvenDigits2(): -1



Example 5 – Suggested Code

Python:

```
def extEvenDigits(num):
    power = 1;
    total = 0;
    while (num>0):
        digit = num % 10;
        num //= 10;
        if ((digit % 2) == 0):
            total += digit * power;
            power *= 10;
        if (power == 1):
            return -1
        else:
            return total
def main():
    num = int(input("Enter the number: "))
    result = extEvenDigits(num)
    print("extEvenDigits():", result)
main()
```

C: main()

```
#include <stdio.h>
#define INIT_VALUE 999
int extEvenDigits1(int num);
void extEvenDigits2(int num, int *result);
int main()
{
    int number, result = INIT_VALUE;
    printf("Enter a number: \n"); scanf("%d", &number);
    printf("extEvenDigits1(): %d\n", extEvenDigits1(number));
    extEvenDigits2(number, &result);
    printf("extEvenDigits2(): %d\n", result);
    return 0;
}
```



Example 5 – Suggested Code

(i) Using Call by Value

```
int extEvenDigits1(int num)
{
    int power = 1; int total = 0; int digit;
    while (num > 0) {
        digit = num % 10;
        num /= 10;
        if ((digit % 2) == 0) {
            total += digit * power;
            power *= 10;
        }
    }
    return (power == 1) ? -1 : total;
}
```

(ii) Using Call by Reference

```
void extEvenDigits2(int num, int *result)
{
    int power = 1; int total = 0; int digit;

    while (num > 0) {
        digit = num % 10;
        num /= 10;
        if ((digit % 2) == 0) {
            total += digit * power;
            power *= 10;
        }
    }
    if (power == 1)
        *result = -1;
    else
        *result = total;
}
```





Thank you !!!

