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,

A black rectangle with white background

AI-generated content may be incorrect.

For each case below, determine the value of

(a) number (b) &number (c) p (d) &p (e) \*p

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | (a) number | (b) &number | (c) p | (d) &p | (e) \*p |
| * 1. p = 100; number = 8 | 8 | 7700 | 100  [treat as variable] | 3478  [POV of variable] | Content @ memory location [100]  🡺 undefined since not initialised yet |
| * 1. number = p   **[p doesn’t change]** | 100  [value of p = number] | 7700 | 100 | 3478 | Content @ memory location [100] |
| * 1. p = &number | 100 | 7700 | 7700 | 3478 | 100  Content @ [7700] |
| * 1. \*p = 10 | 10 | 7700 | 7700 | 3478 | 10 |
| * 1. number = &p   just assigning value to a variable | 3478 | 7700 | 7700 | 3478 | 3478 |
| * 1. p = &p | 3478 | 7700 | 3478 | 3478 | 3478 |

All of the results are cumulative.

int \*(\*var): pointer to a pointer [arrays]

data



var

1. **(digitValue)** Write a function that returns the value of the *k*th digit (k>0) from the right of a nonnegative integer *num*. For example, if num is 1234567 and k is 3, the function will return 5 and if num is 1234 and k is 8, the function will return 0. Write the function in two versions. The function **digitValue1()** returns the result, while **digitValue2()** passes the result through pointer parameter result. The prototypes of the function are given below:

int digitValue1(int num, int k);

void digitValue2(int num, int k, int \*result);

Some sample input and output sessions are given below:

1. Test Case 1:

Enter the number:

234567

Enter k position:

*3*

digitValue1(): 5 digitValue2(): 5

1. Test Case 2:

Enter the number: 234567

Enter k position:

1

digitValue1(): 7 digitValue2(): 7

1. Test Case 3:

Enter the number: 123

Enter k position:

8

digitValue1(): 0 digitValue2(): 0

|  |
| --- |
| #include <stdio.h>  int digitValue1(int num, int k);  void digitValue2(int num, int k, int \*result);  int main()  {  int number, digit, result2 = -1;  // obtain user input  printf("Enter number: ");  scanf("%d", &number);  printf("Enter digit: ");  scanf("%d", &digit);  // passing into functions  int result1 = digitValue1(number, digit);  digitValue2(number, digit, &result2);  printf("digitValue1() = %d, digitValue2() = %d\n", result1, result2);  return 0;  }  int digitValue1(int num, int k)  {  for (int i = 1; i <= k - 1; i++)  num /= 10;  // when k > no. of digits in num --> num automatically becomes 0  return (num % 10);  }  void digitValue2(int num, int k, int \*result)  {  for (int i = 1; i < k; i++)  num /= 10;  \*result = num % 10;  } |

1. **(extOddDigits)** Write a function that extracts the odd digits from a positive number, and combines the odd digits sequentially into a new number. The new number is returned to the calling function. If the input number does not contain any odd digits, then the function returns 1. For example, if the input number is 1234567, then 1357 will be returned; and if the input number is 24, then –1 will returned. Write the function in two versions. The function extOddDigits1() returns the result to the caller, while the function extOddDigits2() returns the result through the pointer parameter, result. The function prototypes are given as follows:

**int extOddDigits1(int num);**

**void extOddDigits2(int num, int \*result);**

Some sample input and output sessions are given below:

1. Test Case 1:

Enter a number:

1234 extOddDigits1(): 13

extOddDigits2(): 13

1. Test Case 2:

Enter a number:

2468

extOddDigits1(): -1 extOddDigits2(): -1

1. Test Case 3:

Enter a number:

1357 extOddDigits1(): 1357 extOddDigits2(): 1357

1. Test Case 4:

Enter a number:

5 extOddDigits1(): 5 extOddDigits2(): 5

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #define INIT\_VALUE 999  int ext\_odd\_digits\_1(int num);  void ext\_odd\_digits\_2(int num, int \*result);  int main()  {  int number = 0, result1 = 0, result2 = 0;  // obtaining user input  printf("Enter number: ");  scanf("%d", &number);  // passing into functions  result1 = ext\_odd\_digits\_1(number);  ext\_odd\_digits\_2(number, &result2);  printf("extOddDigits1() = %d, extOddDigits2() = %d\n", result1, result2);  return 0;  }  // extract odd numbers --> arrange them in sequence  int ext\_odd\_digits\_1(int num)  {  int result = 0, last\_dig = 0, count = 0;  while (num > 0)  {  last\_dig = num % 10; // extracting last digit  if (last\_dig % 2 == 1) // if last digit is odd  {  result += last\_dig \* pow(10, count); // append to number  count++;  }  num /= 10; // divide by 10 to remove last digit  }    if (result == 0)  return -1;  else  return result;  // return (result == 0) ? -1 : result;  }  void ext\_odd\_digits\_2(int num, int \*result)  {  int last\_dig = 0, count = 0;  while (num > 0)  {  last\_dig = num % 10;  if (last\_dig % 2 == 1)  {  \*result += last\_dig \* pow(10, count);  count++;  }  num /= 10;  }  if (\*result == 0)  \*result = -1;  } |

1. **(calDistance)** Write a C program that accepts four decimal values representing the coordinates of two points, i.e. (x1, y1) and (x2, y2), on a plane, and calculates and displays the distance between the points:

A black and white math equation

AI-generated content may be incorrect.

Your program should be implemented using functions. Provide two versions of the function for calculating the distance: (a) one uses call by value only for passing parameters; and (b) the other uses call by reference to pass the result to the calling function.

The function prototypes are given below:

void inputXY(double \*x1, double \*y1, double \*x2, double \*y2); void outputResult(double dist);

double calDistance1(double x1, double y1, double x2, double y2); void calDistance2(double x1, double y1, double x2, double y2, double \*dist);

**\*\* call by reference: allows multiple values to be returned from 1 function, unlike call by value**

Some sample input and output sessions are given below:

1. Test Case 1:

Input x1 y1 x2 y2: 1 1 5 5

calDistance1(): 5.66 calDistance2(): 5.66

1. Test Case 2:

Input x1 y1 x2 y2: -1 -1 5 5

calDistance1(): 8.49 calDistance2(): 8.49

#include <stdio.h>

#include <math.h>

void inputXY(double \*x1, double \*y1, double \*x2, double \*y2);

void outputResult(double dist);

double calDistance1(double x1, double y1, double x2, double y2);

void calDistance2(double x1, double y1, double x2, double y2, double \*dist);

int main()

{

double x1, y1, x2, y2, distance = -1;

inputXY(&x1, &y1, &x2, &y2); // call by reference

distance = calDistance1(x1, y1, x2, y2); // call by value

printf("calDistance1(): ");

outputResult(distance);

calDistance2(x1, y1, x2, y2, &distance); // call by reference

printf("calDistance2(): ");

outputResult(distance); // call by value

return 0;

}

// scanf: floating / double = %lf placeholder

void inputXY(double \*x1, double \*y1, double \*x2, double \*y2)

{

printf("Enter X, Y coordinates (x1 y1 x2 y2): ");

scanf("%lf %lf %lf %lf", x1, y1, x2, y2);

}

void outputResult(double dist)

{

printf("%.2lf\n", dist);

}

double calDistance1(double x1, double y1, double x2, double y2)

{

double x\_diff = 0, y\_diff = 0;

x\_diff = pow(x2 - x1, 2);

y\_diff = pow(y2 - y1, 2);

return pow(x\_diff + y\_diff, 0.5);

}

void calDistance2(double x1, double y1, double x2, double y2, double \*dist)

{

double x\_diff = 0, y\_diff = 0;

x\_diff = pow(x2 - x1, 2);

y\_diff = pow(y2 - y1, 2);

\*dist = pow(x\_diff + y\_diff, 0.5);

}