## Exercises

1. Computing the Ackermann Function

|  |
| --- |
| **int** Ackermann( **int** m, **int** n)  {  **if** (m==0) **return** n+1;  **else if** (m*>*0 && n==0) **return** Ackermann(m−1 ,1);  **else if** (m*>*0 && n*>*0)  **return** Ackermann( m−1, Ackermann(m, n−1));  } |

Write a C++ program to compute A(3,4) using above function.

|  |
| --- |
| #include<iostream>  #include<conio.h>  using namespace std;  int ackermann(int m, int n);  int main()  {  cout << "The value after execution of Ackermann(3,4) is: " << ackermann(3, 4) << endl;  \_getch();  return 0;  }  int ackermann(int m, int n)  {  if (m == 0)  {  return n + 1;  }  else if (m > 0 && n == 0)  {  return ackermann(m - 1, 1);  }  else if (m > 0 && n > 0)  {  return ackermann(m-1,ackermann(m, n - 1));  }  } |
|  |

1. Write a recursive function in C++ to print numbers from n to 0.

|  |
| --- |
| void printNumber(int n)  {  if (n < 0)  {  return;  }  else if (n >= 0)  {  cout << n << " ";  printNumber(n - 1);  }  } |
|  |

1. Write a function in C++ using Recursion to compute binomial coefficients C(n, k) using the recursive definition:

C(n,n) = 1

C(n,0) = 1

C(n,k) = C(n-1, k-1) + C(n-1,k) for (0<k<n) and n>1

|  |
| --- |
| int c(int n, int k)  {  if (n == k || k == 0)  {  return 1;  }  else if (k > 0 && n > k && n > 1)  {  return c(n - 1, k - 1) + c(n - 1, k);  }  } |
|  |

1. Write a recursive function in C++ to check if a number n is prime. (You have to check whether n is divisible by any number below n)

|  |
| --- |
| int primeCheck(int n, int i=2) //copied from internet  {  // Base cases  if (n <= 2)  return (n == 2) ? true : false;  if (n % i == 0)  return false;  if (i \* i > n)  return true;  // Check for next divisor  return primeCheck(n, i + 1);  } |
|  |

1. Write a function Multiply(int a[], int size) to (recursively) compute the result of the elements in an array.

*//Example Run*

**int** arr []={1 ,2 ,3 ,4} ; **int** result = Multiply ( arr ,4) ; cout*<<*result *<<*endl ; *//Should print 24*

|  |
| --- |
| int multiply(int a[], int Size)  {  if (Size == 0)  {  return a[Size];  }  else  {  return a[Size-1]\*multiply(a,Size-1);  }  } |
|  |

1. Give a recursive procedure that checks if a given list is a palindrome

|  |
| --- |
| int n = 0;  bool palindromeCheck(string s,int Size)  {  if (s[n] != s[Size-1])  {  return false;  }  else  {  n++;  palindromeCheck(s, Size - 2);  return true;  }  } |
|  |
|  |