CS1100 - Lecture 20

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

In the last class, we discussed about the function `isprime()` that checks if an integer is a prime number or not. A program to print the first `n` prime numbers which uses this function is given below.

```
#include <stdio.h>
int isprime(int);
int main()
{
  int counter=0, num, n, flag;
  printf("enter the number of primes required\n");
  scanf("%d", &n);
  printf("\nfirst %d primes are given below\n",n);
  num=2;
  while(counter < n)
  {
      flag=isprime(num);
      if(flag==1)
      {
         printf("%d \n", num);
         counter = counter + 1;
      }
      num=num+1;
  }
  return(0);
}

int isprime(int k)
/*returns 1 if k is a prime number and returns 0 otherwise.*/
{
      int i;

      if(k<=1)
        return(0);
      else
      {
         for(i=2; i<=k/2; i++)
         {
            if (k%i == 0) //we found a non-trivial divisor of n
            {
```

```
                return(0);
            }
        }
    }
    return 1;
}
```

In the previous lecture, we explained that the function `isprime()` works as described in the comment i.e., `isprime` returns 1 if the parameter is a prime number and returns 0 otherwise. Now, we give a brief description about the control and data flow of the `main()` function assuming that `isprime()` works correctly.

In the `main()` function, the variable `counter` keeps the count of the prime numbers printed. It is initialized to 0. Since 1 is neither prime nor composite, the value of the variable `num` whose primality is to be checked starts from 2. Inside the `while` loop, the function `isprime()` is called with parameter `num`. Now, the program control passes from this line of the `main()` function to the first line of the `isprime()` function.

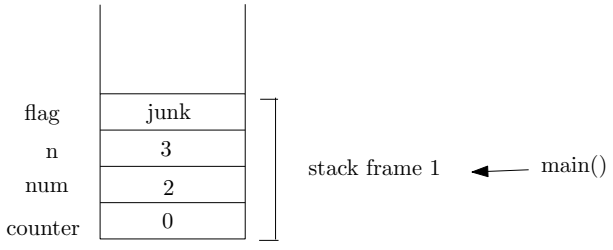
Each time the function `isprime()` is called, a new activation record is created for this function in the memory on top of that of the `main()` function and the program control passes to the first line of the `isprime()` function that is being called. This activation record has memory allocated for the parameters of the function and the local variables defined inside the function `isprime()`. Here, storage for variables `i` and `k` are created in the new activation record. When `isprime()` function is called, the value of the variable `num` from the `main()` function will be copied to the memory location of the variable `k` in the new activation record.

After executing the function `isprime()`, the control comes back to the `main()`. The return value of `isprime` is copied to the variable `flag` in `main()`. If the current value of `num` is prime, the variable `flag` will get assigned the value 1 or else it will get assigned the value 0. After this, the program execution continues from the next line in `main()`. If the variable `flag` becomes 1, the `num` value is printed and the `counter` value is incremented. After this, the variable `num` is incremented so that the new value of `num` (i.e., the old value of `num`+1) the next integer is checked for primality in the next iteration of the `while` loop. This repeats till `counter` becomes `n`, i.e., when `n` prime numbers have been printed.

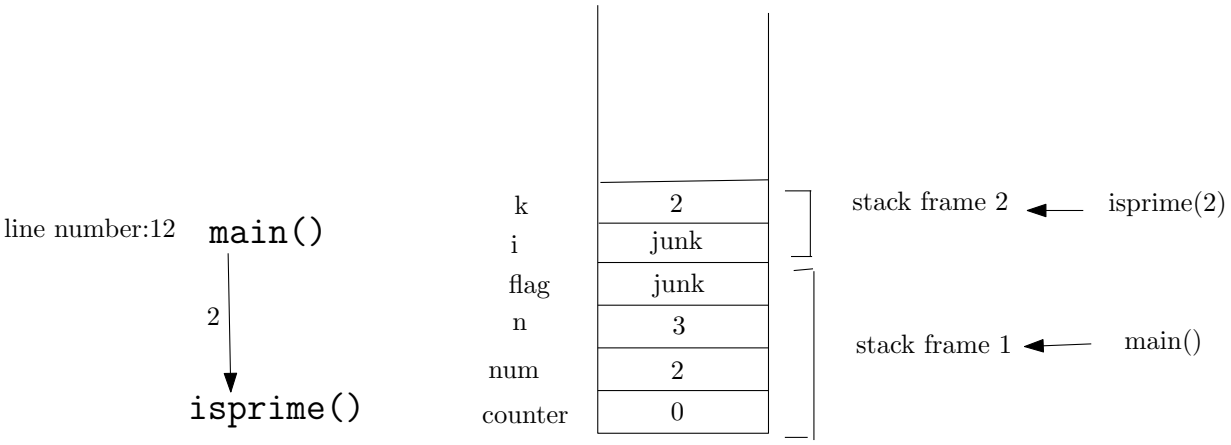
Detailed control and data flow of the program for the input n=3:

When the `main()` function of this program starts execution, an activation record is created in the memory for storing the variables `counter`, `num`, `n` and `flag`.

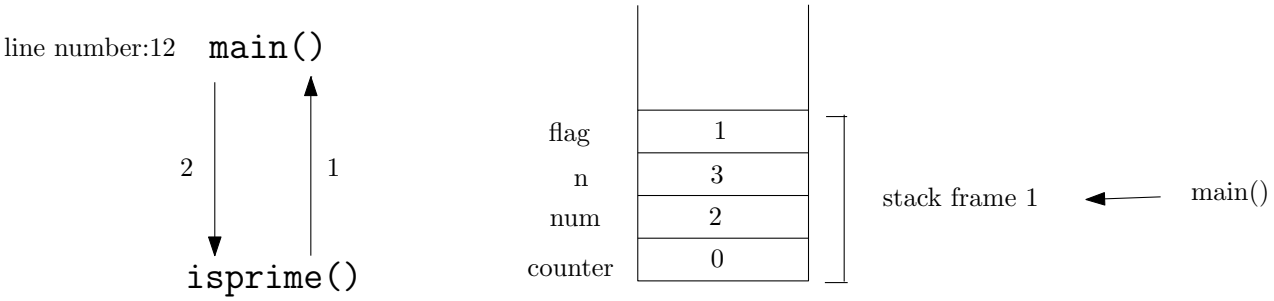
After reading the value of `n` and setting `num` as 2, these values are stored to their corresponding locations. The contents of the memory locations are as shown in the figure below.



For the input `n=3`,when the line `flag=isprime(num);` is executed for the first time, the value of the parameter `num` is 2 and after creating the new activation record for `isprime()`, the value of `num` is copied to the location of `k` in the new activation record. After this, the program control passes to the first line of this function. The contents of memory locations at this point of execution and the control flow during the function call is as follows.

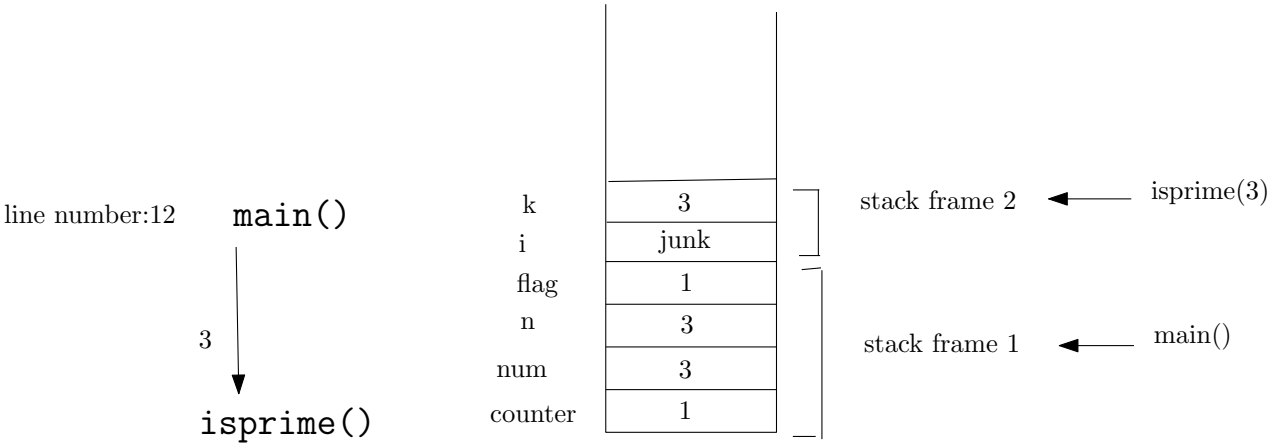


Since the value of the parameter `k` (which is 2) is prime, the function `isprime()` returns 1 and program control switches back to `main()`. The return value 1 is copied to the address location of the variable `flag` of the `main()` function and the activation record of `isprime()` is deleted from the memory. The contents of memory locations and the control flow during the return from the function is as follows.

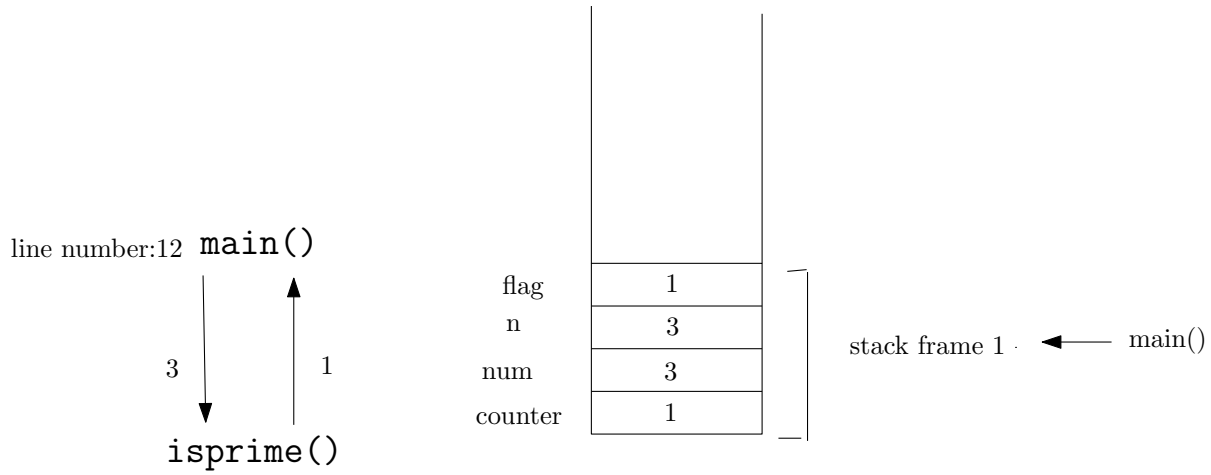


Now, the program counter proceeds to the next statement `if(flag==1)` in `main()`. Since the value of `flag` is 1, the value of `num` is printed, and `counter` is incremented to 1. The `num` value is now incremented to 3.

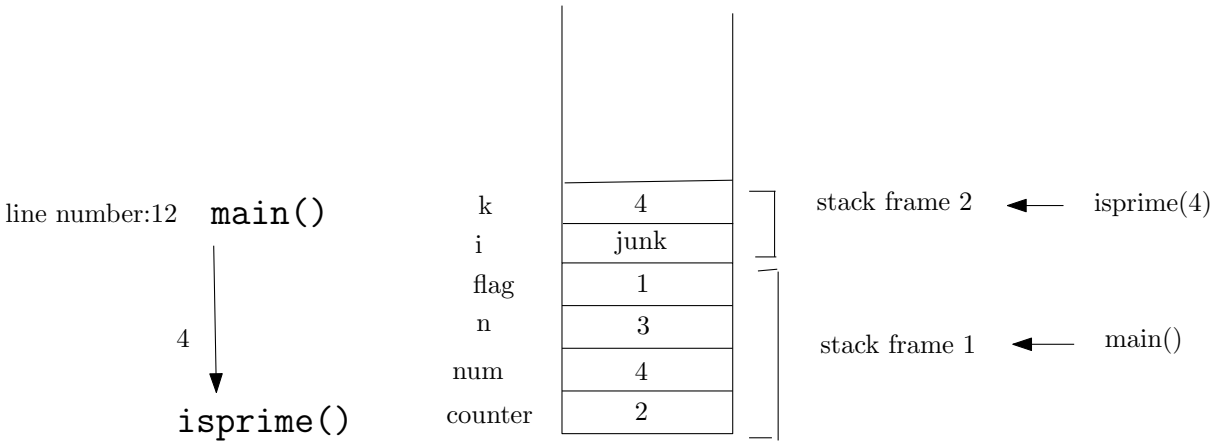
Since the `while` condition becomes *true*, the program control enters the loop again and `isprime(num)` is called. Now, the parameter value is 3 because the value of `num` is 3. After creating the new activation record for `isprime()`, the value of the parameter `num` is copied to the location of `k` in the new activation record. Now again, the program control passes to the first line of the `isprime()` function. The contents of memory locations at this point of execution and the control flow during the function call is as follows.



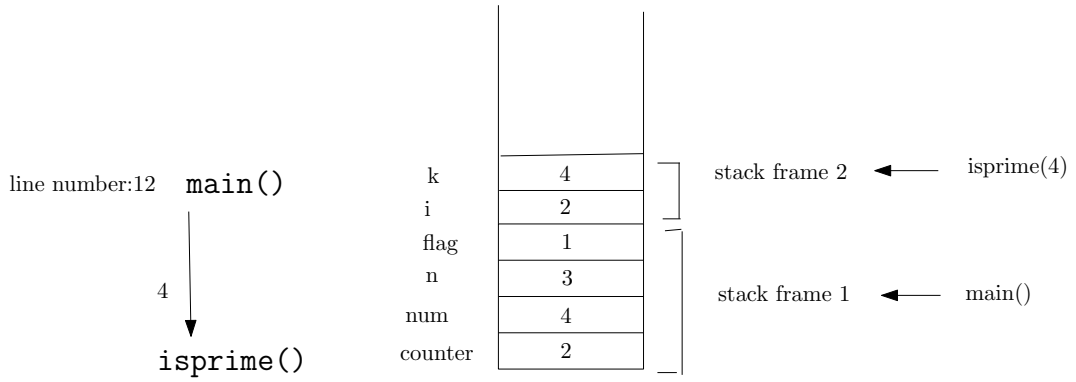
Since the value of the parameter `k` (which is 3) is prime, the function `isprime()` returns 1 and program control switches back to `main()`. The return value 1 is copied to the address location of the variable `flag` of the `main()` function and the activation record of `isprime()` is deleted from the memory. The contents of memory locations and the control flow during the return from the function is as follows.



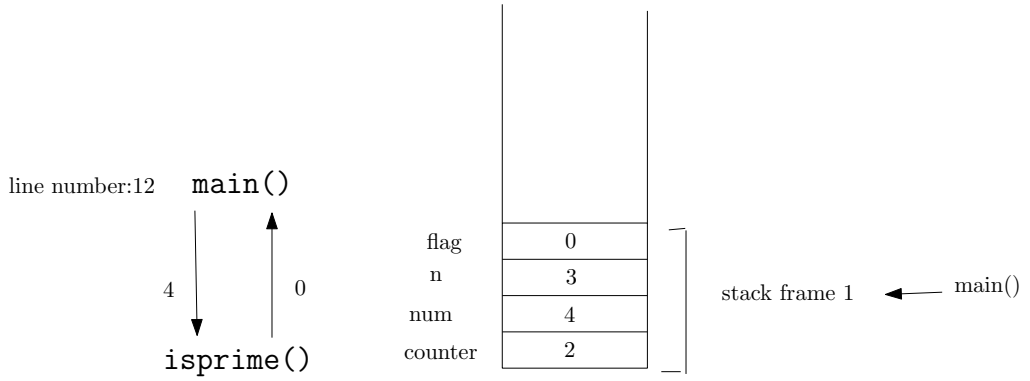
Since the value of `flag` is 1, the variables `counter` and `num` are incremented to 2 and 4 respectively. Since the `while` condition becomes *true*, the program control enters the loop again and `isprime(num)` is called. Now, the parameter value is 4 because the value of `num` is 4. After creating the new activation record for `isprime()`, the value of the parameter `num` is copied to the location of `k` in the new activation record. Now again, the program control passes to the first line of the `isprime()` function. The contents of memory locations at this point of execution and the control flow during the function call is as follows.



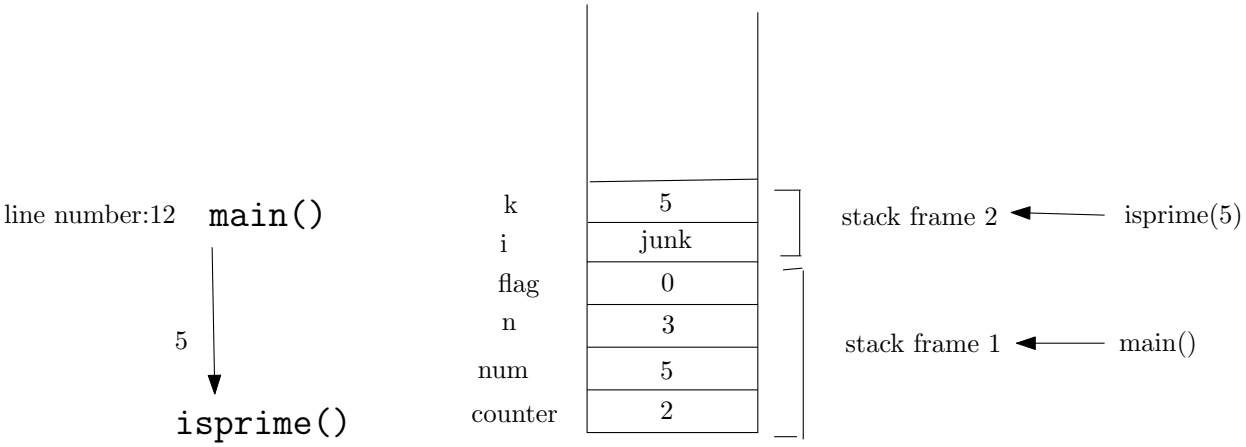
In this case, the program control enters the `for` loop inside the `isprime()` function. When the condition `if (k%i == 0)` is evaluated for `i=2`, the condition becomes *true* and in the next line, the function returns a value 0. Just before returning after the function, the memory state diagram is as follows.



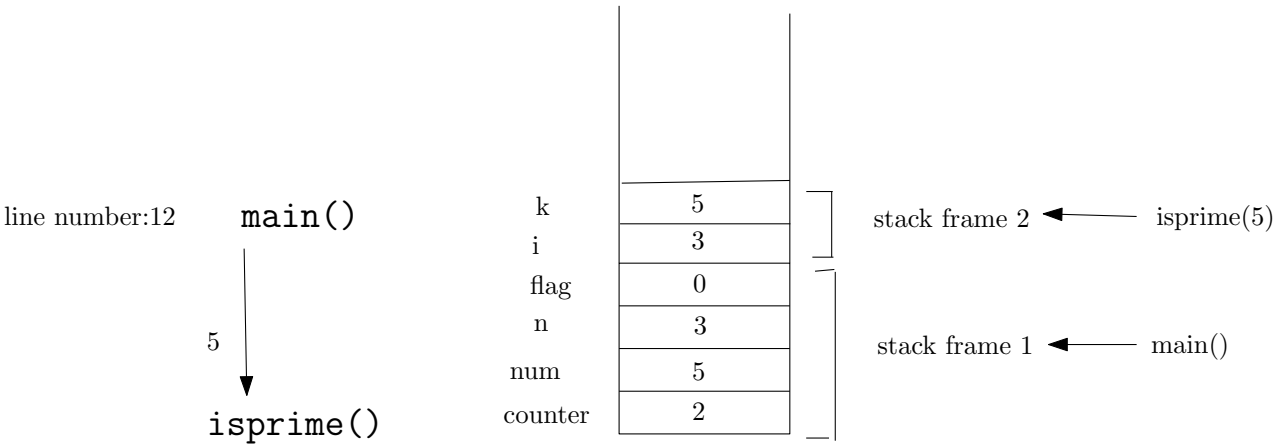
After executing the instruction `return(0);`, the function `isprime()` returns 0 and program control switches back to `main()`. The return value 0 is copied to the address location of the variable `flag` of the `main()` function and the activation record of `isprime()` is deleted from the memory. The contents of memory locations and the control flow during the return from the function is as follows.



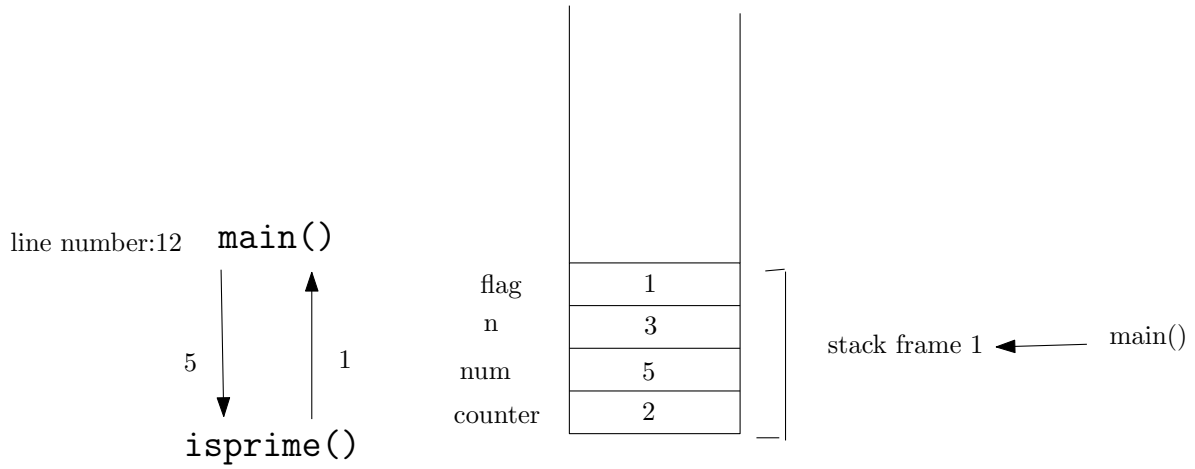
Now, `counter` and `num` are incremented to 2 and 5 respectively. Again the `while` condition is *true* and the `isprime(num)` is called with value of `num=5`. After executing the first line of `isprime()` function, the memory diagram and the control flow during the function call is as follows.



Since the value of `num` (which is 5) is prime, the `for` loop inside `isprime()` executes till the value of `i` becomes 3 and the loop is exited. At this point, the memory diagram and the control flow diagram is as follows.



After exiting the `for` loop, the function `isprime()` returns 1. The memory diagram after the program control switches back to `main()` will be as follows.



Now, the values of both `counter` and `num` are incremented to 3 and 6 respectively. But then the `while` condition becomes `false` and the program terminates.

The `main()` function in the above program can also be written in a different way as given below.

```
int main()
{
  int counter=0, num, n;
  printf("enter the number of primes required\n");
  scanf("%d", &n);
  printf("\nfirst %d primes are given below\n",n);
  num=2;
  while(counter < n)
  {
      if(isprime(num))
      {
         printf("%d \n", num);
         counter = counter + 1;
      }
      num=num+1;
  }
  return(0);
}
```

This example shows that the return value of a function call need not be always assigned to a variable, rather it can also be evaluated as an expression. In the `main()` function of the above program, the statement `if(isprime(num))` directly checks if the return value of `isprime()` is `1`. In that case, the value of `num` is printed. Otherwise, the next number is taken. A function call statement can also occur in a arithmetic expression. For example, `counter+isprime(num)` is a valid expression.

Parameter passing mechanism: Call by Value

When a function is called, there are two questions to be dealt with. One is how is data manipulated and the other is how the control flows within the program. Data manipulation is done by creating an activation record for the function called and storing all its variables in it. This is created on top of the activation record of the function which called the current function under execution. When a function is called, the program control passes to the first line of the function being called. The function is executed line by line until a `return` statement is seen. After returning from a function, the activation record of that function will be removed from the memory as mentioned before.

When a function A in a C program calls another function B, the program control switches from function A to the beginning of the function B and the values of actual parameters in the function call instruction in A are copied to the locations of the corresponding formal parameters in the new activation record created for B. When the program control returns from function B to function A, only the return value is given back to the function A. This type of parameter passing mechanism is known as **call by value** method. C language supports only this kind of mechanism. Other languages may support other kind of mechanisms such as call by reference which are slightly different from what we discussed.

Scope and life of a variable

Scope of a variable is the region of a program where a variable is visible to the system. Only those local variables in the current activation record is visible to the system. Life of a variable is the measure of how long a variable live i.e, time for which a variable is present in the memory. If the scope of a variable is only within a function, then it is a local variable. When we learn more advanced topics we will see how variables with larger scope can be defined.

Consider a function to find the factorial of a number.

```
long long int factorial(int k)
//returns k! for any input k such that 0 <= k < 20
{
      int i;
      long long int f=1;

      if(k<=1)
        return(1);
      else
      {
         for(i=1; i<=k; i++)
            f=f*i;

         return(f);
      }
}
```

The function `factorial` takes an integer parameter `k` as input whose factorial is to be found. Since the factorial of higher values of `k` will be large numbers that cannot be stored

as integers, the return type of the function is made `long long int`. For non-negative `k` values less than or equal to 1, the value `1` is returned and for `k` values greater than 1, factorial is computed and stored to a variable `f` of type `long long int` which is returned.

A program to find the factorial of a number which uses this function is given below.

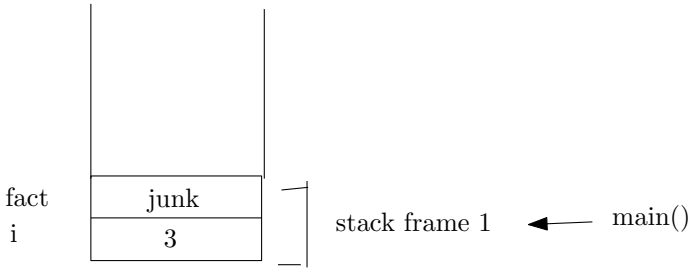
```
//for displaying factorials of numbers 0 to n
#include <stdio.h>
long long int factorial(int);
int main()
{
  int i;
  long long int fact;
  printf("enter i (<20)\n");
  scanf("%d", &i);
  if (i < 20)
  {
      fact=factorial(i);
      printf("%d! = %lld \n",i, fact);
  }
  else
     printf("n too large \n");

  return(0);
}

long long int factorial(int k)
//returns k! for any input k such that 0 <= k < 20
{
       int i;
       long long int f=1;

       if(k<=1)
         return(1);
       else
       {
          for(i=1; i<=k; i++)
             f=f*i;

          return(f);
       }
}
```

In the above program, there is a local variable `i` defined in the `main()` function and there is another local variable `i` defined in the `factorial()` function. The scope of these variables are limited to only the corresponding function and they have no meaning outside. The first variable `i` in the `main()` function is created in the first stack frame when the program starts executing. The local variable `i` in `factorial()` does not exist at this

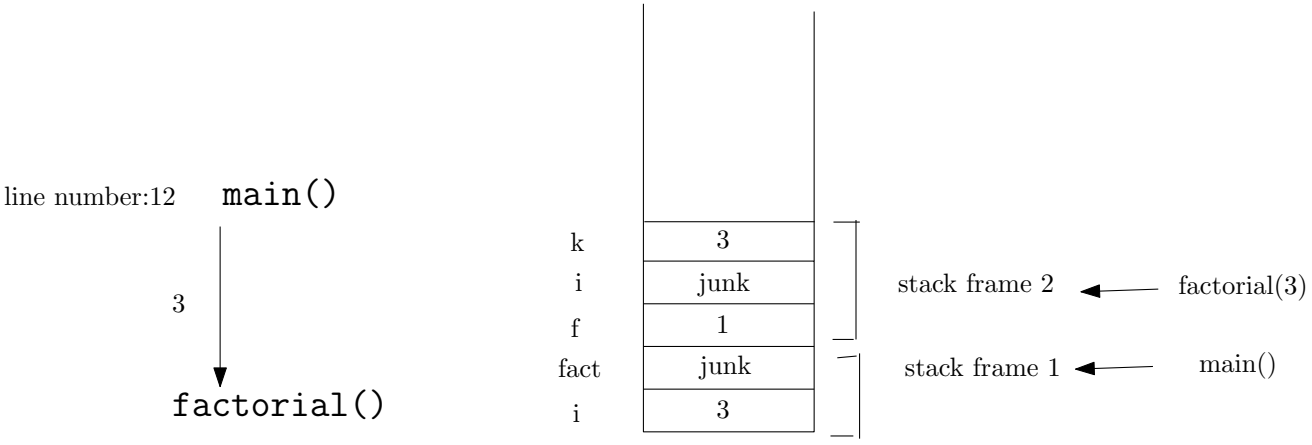
time. When the statement `fact=factorial(i);` is executed, a new activation record of `factorial()` is created and a memory location for the local variable `i` in `factorial()` is created in this new activation record. At this point of time, both the local variables with label `i` simultaneously exist in memory. But they are like two different variables independent of each other. This is because they are bound to two separate memory locations in two different activation records. While executing the function `factorial()`, the activation record on top of the memory is the activation record of `factorial()` and only variables in this activation record (stack frame) can be accessed or modified while executing the `factorial()` function. Similarly, when `main()` function is executing, the activation record on top of the memory will be the activation record of `main()` and only variables in this activation record (stack frame) can be accessed or modified while executing the `main()` function.

Detailed control and data flow of the program for the input i=3:

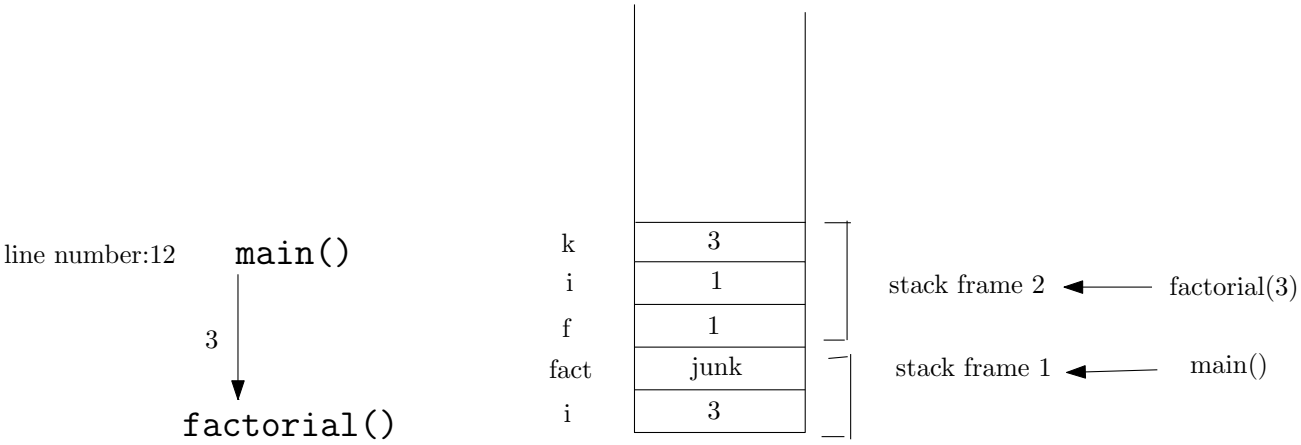
After reading `i=3` in the `main()` function, the memory diagram is as follows.



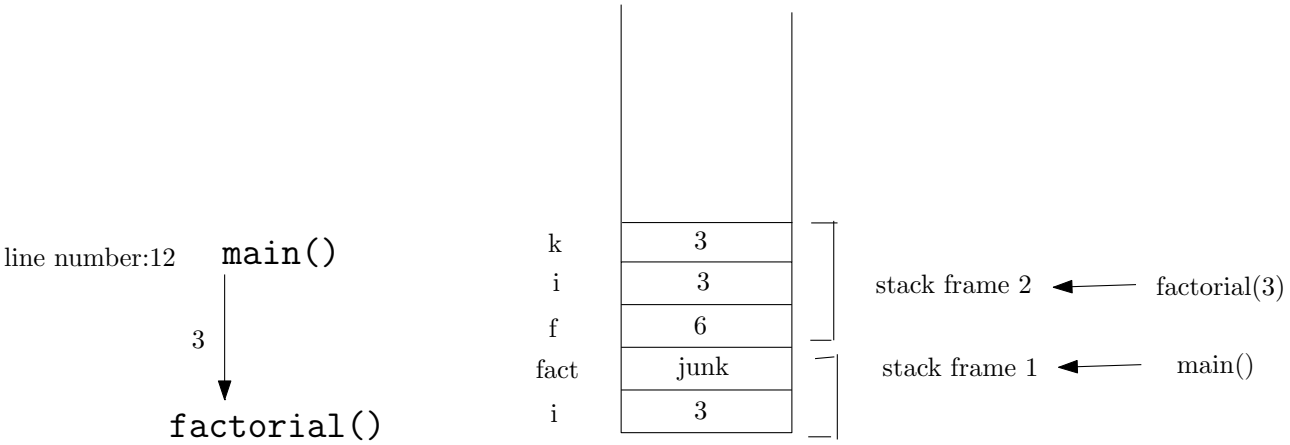
After calling the function `factorial()` with parameter 3, the program control gets transfered to the first line of the `factorial()` function. After executing the instruction `long long int f=1;`, the memory diagram and the control flow associated with the function call are as follows.



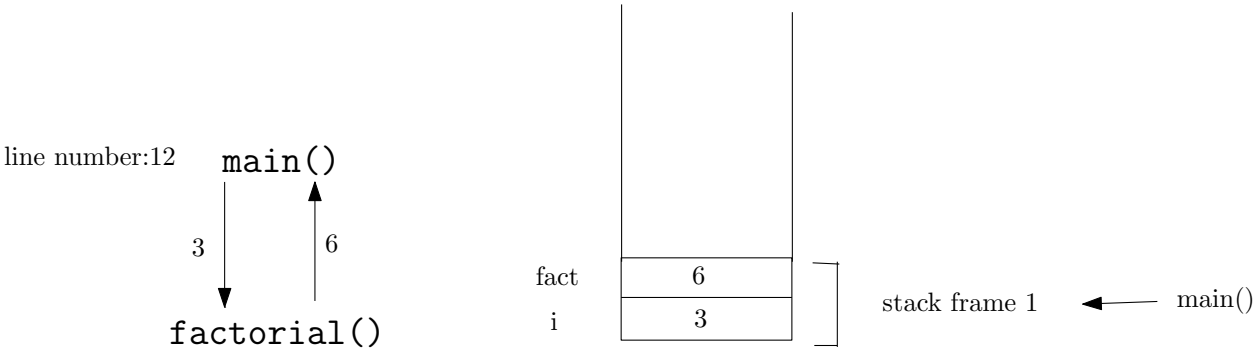
When the program control enters the `for` loop in the `factorial` function for the first time, the memory diagram and the control flow diagram are as follows.



After the program control comes out of the `for` loop, the memory diagram and the control flow diagram are as follows.



Now, the value of `f` is returned to the `main()` function and is copied to the location of the variable `fact`. The memory diagram and the control flow diagram are as follows.



Now, the program execution continues in `main()` function. The value of the variable `fact` (which is 6) is printed as the factorial of the integer `i`.