

CENG 230

Introduction to C Programming

Week 8 – Functions

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli, and the lecture notes of the textbook by Hanly and Koffman.

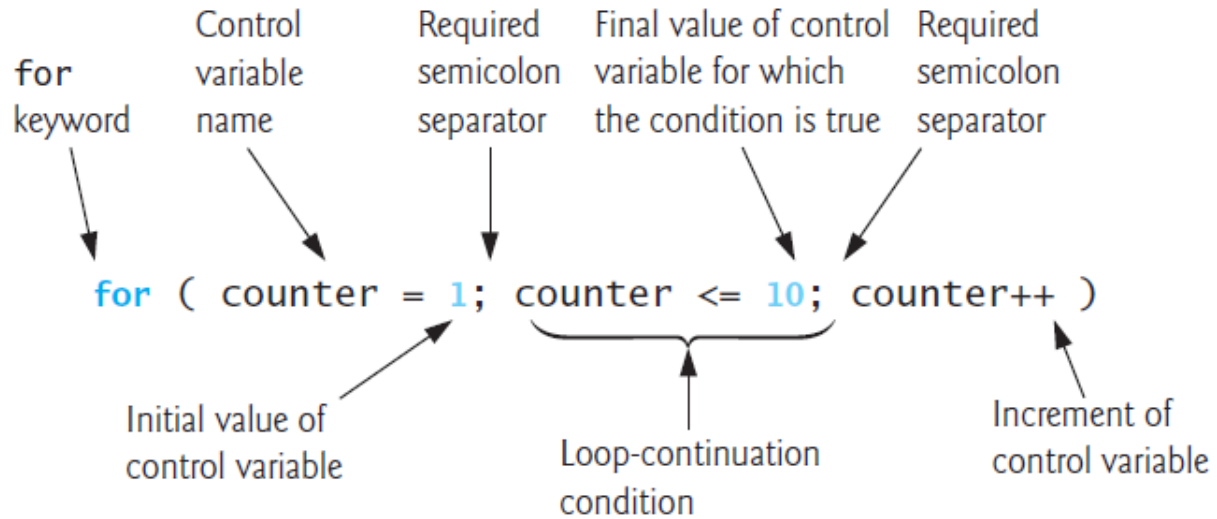
Finding fibonacci series

Previously on CEng 230!

```
#include <stdio.h>
int main()
{
    int count, n, t1=0, t2=1, display=0;
    printf("Enter number of terms: ");
    scanf("%d",&n);
    printf("Fibonacci Series: %d\n%d\n", t1, t2); /* Displaying first two terms */
    count=2; /* count=2 because first two terms are already displayed. */
    while (count<n)
    {
        display=t1+t2;
        t1=t2;
        t2=display;
        ++count;
        printf("%d \n",display);
    }

    system("pause");
    return 0;
}
```

Previously on CEng 230!



Repetitions

- **for** loop

Initialization;

```
for( expr1; expr2; expr3 )  
    statement
```

Initialization;

```
for( expr1; expr2; expr3 )  
{  
    statement;  
    statement;  
    statement;  
}
```

```
for( j = 0; j < N; j++)  
    printf("j: %d\n", j);
```

```
for(i=0, j=0;  
    i < 0 & j > N; i++, j--);
```

```
for(      ;      ; i++ )  
{  
    if( i > 0 ) return 0;  
}
```

Previously on CENG 230!

Previously on CEng 230!

```
4
5  /* function main begins program execution */
6  int main( void )
7  {
8      int counter; /* define counter */
9
10     /* initialization, repetition condition, and increment
11        are all included in the for statement header. */
12     for ( counter = 1; counter <= 10; counter++ ) {
13         printf( "%d\n", counter );
14     } /* end for */
15
16     return 0; /* indicate program ended successfully */
17 } /* end function main */
```

Fig. 4.2 | Counter-controlled repetition with the for statement. (Part 2 of 2.)

Previously on CEng 230!

1. Vary the control variable from 1 to 100 in increments of 1.

```
for ( i = 1; i <= 100; i++ )
```

2. Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).

```
for ( i = 100; i >= 1; i-- )
```

3. Vary the control variable from 7 to 77 in steps of 7.

```
for ( i = 7; i <= 77; i += 7 )
```

4. Vary the control variable from 20 to 2 in steps of -2.

```
for ( i = 20; i >= 2; i -= 2 )
```

5. Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.

```
for ( j = 2; j <= 17; j += 3 )
```

6. Vary the control variable over the following sequence of values: 44, 33, 22, 11, 0.

```
for ( j = 44; j >= 0; j -= 11 )
```

Previously on CEng 230!

```
1 Fig. 4.5: fig04_05.c
2 Summation with for */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main( void )
7 {
8     int sum = 0; /* initialize sum */
9     int number; /* number to be added to sum */
10
11     for ( number = 2; number <= 100; number += 2 ) {
12         sum += number; /* add number to sum */
13     } /* end for */
14
15     printf( "Sum is %d\n", sum ); /* output sum */
16     return 0; /* indicate program ended successfully */
17 } /* end function main */
```

Sum is 2550

Nested Loops

Previously on CENG 230!

- You can have loops within loops:

```
for(i=0; i<N; i++)  
{  
    for(j=0; j<N; j++)  
    {  
        ...  
    }  
}
```


Nested loops

```
1 #include <stdio.h>
2
3 /* function main begins program execution */
4 int main( void )
5 {
6     int x;
7     int y;
8     int i;
9     int j;
10
11     /* prompt user for input */
12     printf( "Enter two integers in the range 1-20: " );
13     scanf( "%d%d", &x, &y ); /* read values for x and y */
14
15     for ( i = 1; i <= y; i++ ) { /* count from 1 to y */
16
17         for ( j = 1; j <= x; j++ ) { /* count from 1 to x */
18             printf( "@" ); /* output @ */
19         } /* end inner for */
20
21         printf( "\n" ); /* begin new line */
22     } /* end outer for */
23
24     return 0; /* indicate program ended successfully */
25 }
```

Previously on CEng 230!

Previously on CEng 230!

4.36 What does the following program segment do?

```
1  for ( i = 1; i <= 5; i++ ) {  
2      for ( j = 1; j <= 3; j++ ) {  
3          for ( k = 1; k <= 4; k++ )  
4              printf( "*" );  
5          printf( "\n" );  
6      }  
7      printf( "\n" );  
8  }
```

infinite loops

(loops that do not finish executing)

```
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
    int counter = 1; /* initialization */

    while ( 1 ) { /* repetition condition */
        printf ( "%d\n", counter ); /* display counter */
        counter++; /* increment */
    } /* end while */
    system("pause");
    return 0; /* indicate program ended successfully */
} /* end function main */
```

Previously on CEng 230

Factors of a number

Previously on CEng 230!


```
#include <stdio.h>
int main()
{
    int n,i;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factors of %d are: ", n);
    for(i=1;i<=n;++i)
    {
        if(n%i==0)
            printf("%d ",i);
    }
    system("pause");
    return 0;
}
```

break;

Previously on CEng 230!

- Stop the loop/iteration and continue with the statement after the loop.
- Usable with while, for and do-while

```
while(...)  
{ ...  
    break;  
    ...  
}  
statement-X;
```



```
while( 1 )  
{  
    c = getchar();  
    if( c == EOF )  
        break;  
    putchar( c );  
}
```

continue;

Previously on CENG 230!

- Skips the remaining statements in the loop and continues with the “loop head”.
- Usable with while, for and do-while

```
while(...)  
{ ...  
    continue;  
    ...  
}
```

```
Sum = 0;  
for(i=0; i<N; i++)  
{  
    if( i%2 == 0 )  
        continue;  
    sum = sum + i;  
}
```

Previously on CEng 230!

41) What is the output?

```
for(i=0; i<=2; i++)  
  for(j=1; j<3; j++)  
    printf("%d%d", i, j);  
printf("%d%d", i, j);
```

a) 01021112212233

b) 0102111221222

c) 011121021222

d) 01112102122233

e) 01112102122222

42) What is the output?

```
int n=0, i=9, j=0;  
for(i=1, j=7; i<=j; i++, j--)  
  n++;  
printf("%d%d%d", i, j, n);
```

a) 170

b) 443

c) 444

d) 534

e) 900

Today

- Solve the previous assignment
- Modular programming with functions
 - Functions without arguments
 - Functions with arguments

Homework

- Write a program to read in numbers until the number **-1** is encountered. The sum, max and min of all numbers read until this point should be printed out.

Modular programming with functions

Modular programming

“Experience has shown that the best way to develop and maintain a large program is to construct it from **smaller pieces** or **modules, each** of which is more manageable than the original program.

This technique is called **divide and conquer.**”

Function **definition**

```
return_type function_name(parameter declarations)  
{  
    statement-1;  
    statement-2;  
    ...  
}
```

- if is *return_type* not void, “return” statement has to be used:

```
return expression;
```

Function **declaration**

- *return_type* *function_name*(**list-of-params**);
- The parameters have to have the same types as in the function definition although the names of the parameters may differ.
- Example:
 - int factorial(int N);
 - void prim(int m);
- If a function is used before it is defined, it has to be declared first.

Function **call**

`function_name(list of arguments)`

- Example:

- Function declaration:

- ```
int greatest(int A, int B, int C);
```

- Example function call:

- ```
printf(“%d\n”, greatest(10, 20, -10));
```

```
#include <stdio.h>

int square( int y ) /* y is a copy of argument to function */
{
    return y * y; /* returns square of y as an int */
} /* end function square */

/* function main begins program execution */
int main( )
{
    int x; /* counter */
    /* loop 10 times and calculate and output square of x each time */
    for ( x = 1; x <= 10; x++ ) {
        printf( "%d ", square( x ) ); /* function call */
    } /* end for */
    printf( "\n" );
    system("pause");
    return 0; /* indicates successful termination */
} /* end main */
```

```

1  /* Fig. 5.3: fig05_03.c
2     Creating and using a programmer-defined function */
3  #include <stdio.h>
4
5  int square( int y ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
9  {
10     int x; /* counter */
11
12     /* loop 10 times and calculate and output square of x each time */
13     for ( x = 1; x <= 10; x++ ) {
14         printf( "%d ", square( x ) ); /* function call */
15     } /* end for */
16
17     printf( "\n" );
18     return 0; /* indicates successful termination */
19 } /* end main */
20
21 /* square function definition returns square of parameter */
22 int square( int y ) /* y is a copy of argument to function */
23 {
24     return y * y; /* returns square of y as an int */
25 } /* end function square */

```

1 4 9 16 25 36 49 64 81 100

Fig. 5.3 | Using a programmer-defined function. (Part 2 of 2.)


```

1  /* Fig. 7.6: fig07_06.c
2     Cube a variable using call-by-value */
3  #include <stdio.h>
4
5  int cubeByValue( int n ); /* prototype */
6
7  int main( void )
8  {
9     int number = 5; /* initialize number */
10
11    printf( "The original value of number is %d", number );
12
13    /* pass number by value to cubeByValue */
14    number = cubeByValue( number );
15
16    printf( "\nThe new value of number is %d\n", number );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* calculate and return cube of integer argument */
21 int cubeByValue( int n )
22 {
23     return n * n * n; /* cube local variable n and return result */
24 } /* end function cubeByValue */

```

The original value of number is 5
The new value of number is 125

```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void ); /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void ); /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main( void )
13 {
14     int x = 5; /* local variable to main */
15
16     printf("local x in outer scope of main is %d\n", x );
17
18     { /* start new scope */
19         int x = 7; /* local variable to new scope */
20
21         printf( "local x in inner scope of main is %d\n", x );
22     } /* end new scope */
23
24     printf( "local x in outer scope of main is %d\n", x );
```

Fig. 5.12 | Scoping example. (Part 1 of 3.)

5.7 Find the error in each of the following program segments and explain how the error can be corrected (see also Exercise 5.46):

```
a) int g( void )
   {
       printf( "Inside function g\n" );
       int h( void )
       {
           printf( "Inside function h\n" );
       }
   }
```

```
b) int sum( int x, int y )
   {
       int result;
       result = x + y;
   }
```

```
c) int sum( int n )
   {
       if ( n == 0 ) {
           return 0;
       }
       else {
           n + sum( n - 1 );
       }
   }
```

5.7 Find the error in each of the following program segments and explain how the error can be corrected (see also Exercise 5.46):

```
d) void f( float a );
   {
     float a;
     printf( "%f", a );
   }

e) void product( void )
   {
     int a, b, c, result;
     printf( "Enter three integers: " )
     scanf( "%d%d%d", &a, &b, &c );
     result = a * b * c;
     printf( "Result is %d", result );
     return result;
   }
```

Sample 1

29) What is the output?

```
int f1 (int x)
{ int y=2;
  printf("%d%d",x,y);
  return x++;
  return ++y; }
int main (void)
{ int y=5, x=5;
  printf("%d%d\n", f1(y),y);
  return 0;}
```

- a) 25424 b) 5255 c) 5555 d) 5256 e) 5552

Sample 2

31) What is the output?

```
void f1 (void)
{ int y=5;
printf("%d",y); y++;
printf("%d",y);}
int main (void)
{ int y=3;
printf("%d",y);
f1();
printf("%d",y); return 0;}
```

a) 3563 b) 563563 c) 563566 d) 3566 e) 3567

Sample 3

32) What is the output?

```
void f1 (int x)
{ int y=2;
printf("%d%d", x,y);
x++; }
int main (void)
{ int y=5, x=5;
printf("%d%d", x,y);
f1(y);
printf("%d", x);
return 0;}
```

a) 55256 b) 255255 c) 55526 d) 255256 e) 55525