CEng 230 Introduction to C Programming

Seyyit Alper SERT

Department of Computer Engineering
2017-2018 Fall

Web Pages
Official Course Page: ceng230.ceng.metu.edu.tr
Learning Management System (LMS): odtuclass.metu.edu.tr
Contact: alper.sert@ceng.metu.edu.tr

Modular programming (Functions)

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.***

```c
/* Fig. 5.3: fig05_03.c
   Creating and using a programmer-defined function */
#include <stdio.h>

int square( int y ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   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" );
   return 0; /* indicates successful termination */
} /* end main */

/* square function definition returns square of parameter */
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 */
```

```
1  4  9  16  25  36  49  64  81  100
```

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

```c
/* Fig. 7.6: fig07_06.c
   Cube a variable using call-by-value */
#include <stdio.h>

int cubeByValue( int n ); /* prototype */

int main( void )
{
   int number = 5; /* initialize number */

   printf( "The original value of number is %d", number );

   /* pass number by value to cubeByValue */
   number = cubeByValue( number );

   printf( "\nThe new value of number is %d\n", number );
   return 0; /* indicates successful termination */
} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
   return n * n * n; /* cube local variable n and return result */
} /* end function cubeByValue */
```

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

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`exp( x )`	exponential function e^x	`exp( 1.0 )` is `2.718282` `exp( 2.0 )` is `7.389056`
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`

`log10( x )`	logarithm of x (base 10)	`log10( 1.0 )` is `0.0`
		`log10( 10.0 )` is `1.0`
		`log10( 100.0 )` is `2.0`

`fabs( x )`	absolute value of x	`fabs( 13.5 )` is `13.5`
		`fabs( 0.0 )` is `0.0`
		`fabs( -13.5 )` is `13.5`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is `128.0` `pow( 9, .5 )` is `3.0`
`fmod( x, y )`	remainder of x/y as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is `1.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Fig. 5.2 | Commonly used math library functions.

 # include <math.h >

```c
/* Fig. 5.7: fig05_07.c
   Shifted, scaled integers produced by 1 + rand() % 6 */
#include <stdio.h>
#include <stdlib.h>

/* function main begins program execution */
int main( void )
{
   int i; /* counter */

   /* loop 20 times */
   for ( i = 1; i <= 20; i++ ) {

      /* pick random number from 1 to 6 and output it */
      printf( "%10d", 1 + ( rand() % 6 ) );

      /* if counter is divisible by 5, begin new line of output */
      if ( i % 5 == 0 ) {
         printf( "\n" );
      } /* end if */
   } /* end for */

   return 0; /* indicates successful termination */
} /* end main */
```

```
         6          6          5          5          6
         5          1          1          5          3
         6          6          2          4          2
         6          2          3          4          1
```

Fig. 5.7 | Shifted, scaled random integers produced by `1 + rand() % 6`. (Part 2 of 2.)

SCOPE RULES

The scope of an identifier is the portion of the program in which the identifier can be referenced. For example, when we define a local variable in a block, it can be referenced only following its definition in that block or in blocks nested within that block. The four identifier scopes are function scope, file scope, block scope, and function-prototype scope.

```c
/* global variable declaration */
int g;

int main ()
{
  /* local variable declaration */
  int a, b;
  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;
  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
  system("pause");
  return 0;
}
```

```c
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main ()
{
  /* local variable declaration */
  int g = 10;

  printf ("value of g = %d\n",  g);

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

```c
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main ()
{
  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;
  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  int k=sum (250,450);
  int j=sum_2(1200, 2700);
  system("pause");
  return 0;
}

/* function to add two integers */
int sum(int a, int b)
{
    printf ("value of a in sum() = %d\n",  a);
    printf ("value of b in sum() = %d\n",  b);
    return a + b;
}


int sum_2(int m, int n)
{
    printf ("value of m in sum() = %d\n",  m);
    printf ("value of n in sum() = %d\n",  n);
    printf ("value of a in sum_2() = %d\n",  a);
    return m + n;
}
```

```c
/* Fig. 5.12: fig05_12.c
   A scoping example */
#include <stdio.h>

void useLocal( void ); /* function prototype */
void useStaticLocal( void ); /* function prototype */
void useGlobal( void ); /* function prototype */

int x = 1; /* global variable */

/* function main begins program execution */
int main( void )
{
   int x = 5; /* local variable to main */

   printf("local x in outer scope of main is %d\n", x );

   { /* start new scope */
      int x = 7; /* local variable to new scope */

      printf( "local x in inner scope of main is %d\n", x );
   } /* end new scope */

   printf( "local x in outer scope of main is %d\n", x );
```

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

Recursion

```c
/* Fig. 5.14: fig05_14.c
   Recursive factorial function */
#include <stdio.h>

long factorial( long number ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   int i; /* counter */
   /* loop 11 times; during each iteration, calculate
      factorial( i ) and display result */
   for ( i = 0; i <= 10; i++ ) {
      printf( "%2d! = %ld\n", i, factorial( i ) );
   } /* end for */
   system("pause");
   return 0; /* indicates successful termination */
} /* end main */

/* recursive definition of function factorial */
long factorial( long number )
{
   /* base case */
   if ( number <= 1 ) {
      return 1;
   } /* end if */
   else { /* recursive step */
      return ( number * factorial( number - 1 ) );
   } /* end else */
} /* end function factorial */
```

```c
/* Fig. 5.15: fig05_15.c
   Recursive fibonacci function */
#include <stdio.h>

long fibonacci( long n ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   long result; /* fibonacci value */
   long number; /* number input by user */

   /* obtain integer from user */
   printf( "Enter an integer: " );
   scanf( "%ld", &number );

   /* calculate fibonacci value for number input by user */
   result = fibonacci( number );

   /* display result */
   printf( "Fibonacci( %ld ) = %ld\n", number, result );
   return 0; /* indicates successful termination */
} /* end main */

/* Recursive definition of function fibonacci */
long fibonacci( long n )
{
   /* base case */
   if ( n == 0 || n == 1 ) {
      return n;
   } /* end if */
   else { /* recursive step */
      return fibonacci( n - 1 ) + fibonacci( n - 2 );
   } /* end else */
} /* end function fibonacci */
```

Fig. 5.15 | Recursively generating Fibonacci numbers. (Part 1 of 2.)

5.7 Find the error in each of the following program segments and explain how the error can l 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 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

Sample 4

20)
```
void edi_budu(int a)
{ if (!a) return;
  else {printf("%d",a);
  edi_budu(a-1);} }
```

The above function, when called as edi_budu(3.14) will

a) print 3210
b) print 321
c) cause an infinite recursion.
d) cause a compile-time error: "void function cannot return"
e) cause a compile-time error: "argument a is int, but called with some float"

Sample 5

21) What will the following program print?

```
#include<stdio.h>
int i;
void f( ) {
        for (i=0;i<6 && i++,i<10;i++)
        printf("%d ",i); }
int main( ) {
        f( );
        return 0; }
```

a) 0 2 4 6 7 8 9 **b)** 0 2 4 5 6 7 8 9 **c)** 1 2 4 6 7 8 9
d) 1 3 5 6 7 8 9 **e)** 1 3 5 7 9

Sample 6

22)

```
int super_f(int x)
{
 int i,single=0,double=0;
 for (i=0;i<x;i++)
  if (i % 2) single = i;
  else double = i;
  printf("%d ",single+double);
 printf("\n"); }
```

The above function, when called as super_f(5) will

a) print 0 1 3 5 7 **b)** print 7 **c)** print 1 2 3 4

d) print 1 3 5 7 **e)** None of these

Sample 7

23) What is the output of the following program segment?

```
#include <stdio.h>
addTwoInteger(int a, int b){
        int x=10, y=11;
       return (x+y);  }
main( ){
       int x=5, y=6;
       printf("%d", addTwoInteger(x, y)); }
```

a) 21
b) 11
c) 32
d) error: x and y redeclared
e) error: wrong function declaration

Sample 8

25)What will the following program print?

```
#include<stdio.h>
int ex(int a)  {
                if (a<0) return -1;
                else if (a=0) return 0;
                else return 1;  }
   int main( ) {
                 printf("%d %d %d",ex(-10), ex(0), ex(10)); }
```

a) -1 0 1 **b)** -1 0 0 **c)** -1 1 1 **d)** -1 1 0 **e)** None of these

Sample 9

26) What will the following program print?

```
#include<stdio.h>
int k = 1;
int add(int x) { return (x+k++); }
int mult(int k) { return(k*=2); }
int main( ) {
            int t = 2;
            add(k);
            printf("%d %d ",t,k);
            k = mult(t);
            printf("%d %d ",k,t); }
```

a)1 2 1 2 **b)**1 2 2 1 **c)** 2 4 2 4 **d)** 2 2 4 4 **e)** 2 2 4 2

Sample 11

29) What is the output of the code below?

```
#include <stdio.h>
char c='g';
char f(char g) {
char c = 'f ';
printf("%c",c);
return c; }
void h(char x) {
  char ch = 'h';
  printf("%c%c",c,x); }
void k(char c) {
  char ch = 'k';
  printf("%c",c); }
int main() {
  char c = 'm';
  f(c);
  printf("%c",c);
  h(f(c));
  printf("%c",c);
  k(c);
  printf("%c",c);
  printf("\n"); return 0;  }
```

a) fmgfmmm **b)** fmgfmgm **c)** fmgmmmm

d) fmfgfmmm **e)** fmfgfmgm