CPE101 Programming Languages I

Week 12
Functions

Assist. Prof. Dr. Caner ÖZCAN

Functions

- Functions
  - Modules in C
  - Programs combine user-defined functions with library functions
  - C standard library has a wide variety of functions

Benefits of Functions

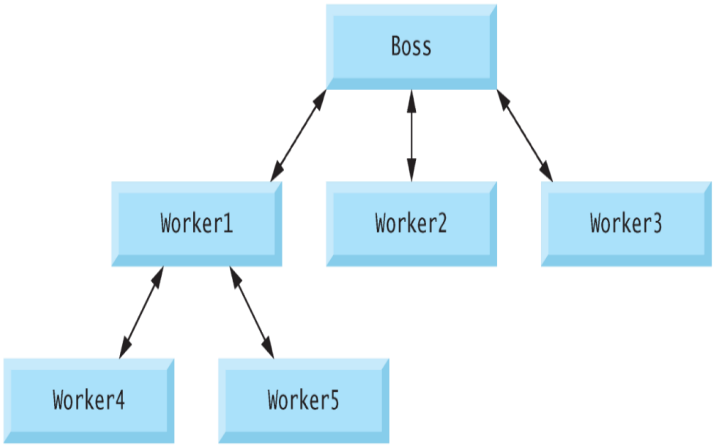
- Benefits of Functions
  - Divide and conquer
    - Construct a program from smaller pieces or components
    - These smaller pieces are called modules.
    - Functions allow you to modularize a program.
    - Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or modules, each of them is more manageable than the original program.
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoid code repetition

Functions

- The variables defined in a function are the local variables of this function.
  - Only known in the body of the function
- Parameters
  - Most functions have a list of parameters that provide the means for communicating information between functions
  - Also local variables of the function
- Function calls
  - Provide function name and arguments (data)
  - Function performs operations or manipulations
  - Function returns results

Functions

- Function call analogy:
- Boss asks worker to complete task
- Worker gets information, does task, returns result
- Information hiding: boss does not know details



Defining Functions

- Format of a function definition :

```
return_value_type  function_name  ( parameter_list )
{
  definitions_and_statements
}
```

- *function_name* is any valid identifier.
- *return_value_type* is the data type of the result returned to the caller
- *return_value_type* void indicates that a function does not return a value.
- Together, the *return_value_type*, *function_name* and *parameter_list* are referred to as the function header.

Defining Functions

- **parameter_list** is a comma-separated list that specifies the parameters received by the function when it's called.
- If a function does not receive any values, *parameter-list* is **void**.
- A type must be listed explicitly for each parameter

Defining Functions

- The ***definitions_and_statements*** within fancy parentheses form the function body.
- The function body is also referred to as a **block**.
- Variables can be declared in any block, and blocks can be nested.
- **A function cannot be defined inside another function.**

Defining Functions

- There are three ways to return control from a called function to the point at which a function was invoked.
- If the function does not return a result
  - Control is returned simply when the function-ending right fancy bracket is reached.
  - or by executing the statement return;
- If the function does return a result, the statement

  return *expression;*

  returns the value of *expression to the caller.*

Function Prototype

- Identity of a function.
- Prototype only needed if function definition comes after use in program.
- The function that has a prototype given below:
  - int maximum(int x, int y, int z);
  - Takes 3 integer parameters.
  - Returns integer value.

Function Prototype

- If a function call does not match the function prototype compilation error is produced.
- An error is also generated if the function prototype and the function definition disagree.
- Another important feature of function prototypes is the coercion of arguments, i.e., the forcing of arguments to the appropriate type.
- For example, the math library function sqrt can be called with an integer argument even though the function prototype in <math.h> specifies a double argument, and the function will still work correctly.
  - The statement ;
  - `printf( "%.3f\n", sqrt( 4 ) );`
  - correctly evaluates sqrt(4), and prints the value 2.000

Defining Functions

```c
/* finding maximum of three integers */
#include <stdio.h>

int max(int x, int y, int z); //function prototyp

int main()
{
    int a, b, c;
    printf("plesase enter three numbers");
    scanf("%d%d%d",&a,&b,&c);
    printf("the maximum number is %d",max(a,b,c));
    return 0;
}
int max(int x, int y, int z)
{
    int maximum;
    if(x>y)
    {
        if(x>z)
            maximum=x;
        else
            maximum=z;
    }
    else if(y>z)
        maximum=y;
    else
        maximum=z;

    return maximum;
}
```

!!!

Buradaki resim karışıktı daha iyi anlaşılması için aynı örneği yazdık

Header Files

- Each standard library has a corresponding header containing the function prototypes and definitions of various data types.
- `<stdlib.h> , <math.h>` , etc
- Load with `#include <file name>`
  - `#include <math.h>`
- Custom header files
  - Create file with functions.
  - Save as `filename.h`
  - Load in other files with `#include "filename.h"`
  - Reuse functions.

Header Files

- **math.h** → Mathematics library functions
- **ctype.h** → Functions for testing characters for certain properties, functions to convert into uppercase or lowercase etc.
- **stdio.h** →Standard input/output functions
- **stdlib.h** →Functions for converting numbers to text or text to number, memory management, random number generation and other utility functions.
- **string.h** → String processing functions
- **time.h** → Time and date functions

Mathematic Library Functions

- Mathematic Library Functions
  - Perform common mathematical calculations.
  - `#include <math.h>`
- Format for calling functions
  - `Function_name( arguments );`
- If multiple arguments, use comma-separated list
- All math functions return data type `double`
- Arguments may be constants, variables, or expressions

Mathematic Library Functions

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

Mathematic Library Functions

Function	Description	Example
`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`

Example: Square function

```c
#include <stdio.h>
float kareAl(float);

void main()
{
    int sayac;
    for(sayac = 1; sayac<=10; sayac++)
    {
        printf("Sayi:%d  Karesi:%d\n", sayac, kareAl(sayac));
    }

    printf("\n%.2f", kareAl(4.5));
}

float kareAl(float a)
{
    return a*a;
}
```

Example: Arithmetic functions

```c
#include <stdio.h>
int toplam(int, int);
int cikar(int, int);
int carp(int, int);
float bol(int, int);

void main()
{
    int secim,s1,s2;
    while(1)
    {
        printf("1-Topla\n2-Cikar\n3-Carp\n4-Bol\n5-Cikis\n");
        scanf("%d", &secim);
        printf("Sayilari gir:");
        scanf("%d %d", &s1, &s2);

        if(secim == 1)
            printf("Sonuc = %d", topla(s1,s2));
        else if(secim == 2)
            printf("Sonuc = %d", cikar(s1,s2));
        else if(secim == 3)
            printf("Sonuc = %d", carp(s1,s2));
        else if(secim == 4)
            printf("Sonuc = %.2f", bol(s1,s2));
        else if(secim == 5)
            exit(0);
        else printf("Yanlis giris");
    }
}
```

```c
int topla(int a, int b)
{
    return a+b;
}
int cikar(int a, int b)
{
    return a-b;
}
int carp(int a, int b)
{
    return a*b;
}
float bol(int a, int b)
{
    return (float)a/b;
}
```

Example: Exponent function

```c
#include <stdio.h>
double usAl(double, double);

void main()
{
    double a,b;
    printf("Taban ve us degeri gir:");
    scanf("%lf %lf", &a, &b);
    printf("%.2f", usAl(a,b));
}

double usAl(double x, double y)
{
    int sayac;
    double sonuc=1.0;
    for(sayac=0;sayac<y;sayac++)
    {
        sonuc *= x;
    }
    return sonuc;
}
```

References

- Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- J. G. Brookshear, “Computer Science: An Overview 10th Ed.”, Addison Wisley, 2009.
- Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- Paul J. Deitel, “C How to Program”, Harvey Deitel.
- Bayram AKGÜL, C Programlama Ders notları