COMP133: INTRODUCTION TO COMPUTER AND PROGRAMMING

Top-Down Design with Functions

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Functions

- A function is a group of statements that together perform a task.
- Every C program has at least one function, which is `main()`, and all the most trivial programs can define additional functions

Functions

- Two types of functions:
  - C library functions (`sqrt(x), abs(x),…`)
  - User defined functions (Your own functions)

- Math library contains mathematical functions.
- `#include<math.h>`

Mathematical Functions

Function	Standard Header File	Example	Argument(s)	Result
abs(x)	<stdio.h>	x=-5 abs(x)=5	int	int
ceil(x)	<math.h>	x=45.23 ceil(x)=46	double	double
cos(x)	<math.h>	x=0.0 cos(x)=1.0	double (radians)	double
exp(x)	<math.h>	x=1.0 exp(x)=2.71828	double	double

Mathematical Functions

Function	Standard Header File	Example	Argument(s)	Result
fabs(x)	<math.h>	x=-8.432 fab(x)=8.432	double	double
floor(x)	<math.h>	x=45.23 floor(x)=45	double	double
log(x)	<math.h>	x=2.71828 log(x)=1.0	double	double
log10(x)	<math.h>	x=100.0 log10(x)=2.0	double	double

Mathematical Functions

Function	Standard Header File	Example	Argument(s)	Result
pow(x,y)	<math.h>	x=0.16 y=0.5 pow(x,y)=0.4	double double	double
sin(x)	<math.h>	x=1.5708 sin(x)=1.0	double (radians)	double
sqrt(x)	<math.h>	x=2.25 sqrt(x)=1.5	double	double
tan(x)	<math.h>	x=0.0 tan(x)=0.0	double (radians)	double

Structure of Functions

1) Useful for programmers to divide their programs into separate modules (instead of one big program). This makes it easy to debug the code and handling error.

2) Reusability:
   - Once a function is defined, it can be used over and over again.
   - You can invoke the same function many times in your program.
   - Use same function in several different (and separate) programs.

Function Structure

```
return-type function-name(list-of-parameters)
{

    function-statements

}
```

Function Structure

1. Function with no arguments and no return value.
2. Function with no arguments but return value
3. Function with arguments and no return value
4. Function with argument and a return value

Writing Functions

- How to write a function:
  1. Function prototype
  2. Function Definition
  3. Function Call

Function Prototype

- Tells the compiler about a function's name, return type, and parameters.

- `return_type function_name ( parameter list )`
- `int sum (int ,int );// with parameters and return value`
- `void printNum (int);// with parameters and no return value`
- `float area (); // no parameters and with return value`
- `double circumference (double);// with parameters and return value`
- `voidprintChar (char); // with parameters and no return value`
- `voidprintSquare();//no arguments and no return value`

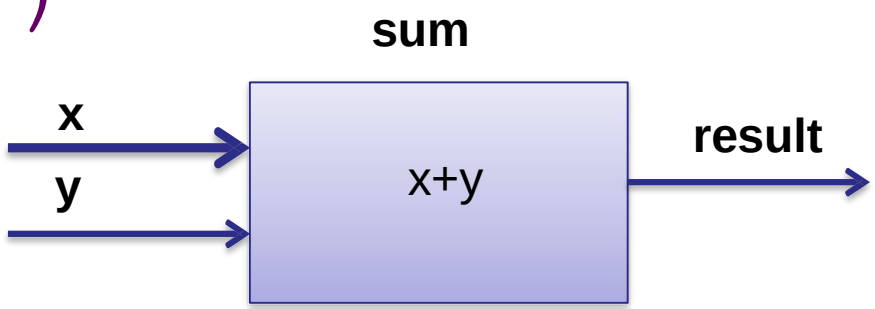
Function Definition

- Provides the actual body of the function.

```
return_type function_name ( parameter list )
{
    body of the function
}
```

Function Definition

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

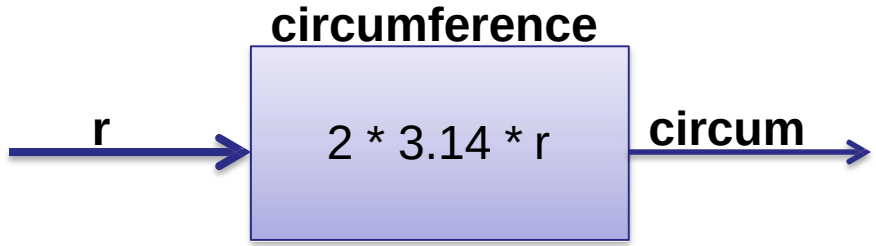


Function Definition

```
void  printNum ( int x )
{
    printf("%d", x);
}
```

Variable declarations and data types

```
double circumference (double r)
{
 double circum;
 circum= 2 * 3.14 * r;
 return circum;
}
```



Function Call

- To use a function, you will have to call that function to perform the defined task.

```
int mySum = sum (x,y);
double circum = circumference (r);
printNum(x);
```

Return Type

- **Return Type**: A function may return a value.
- The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

- A function that does not return a value will return void.

Example

- Write a function sum that takes two integers and returns the sum

```
int sum( int x, int y)
{
   return x + y;
}
```

- Write a function sum that takes two integers and prints the sum

```
void sum( int x, int y)
{
   printf("%d", x + y);
}
```


From the material of Mr. Abdallah Karakra, 2016, Birzeit University.

Function Name

- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters

- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body

- **Function Body:** The function body contains a collection of statements that define what the function does.

Example

- Write a program that contains a function sum that takes two integers and returns their sum.

Example

```
#include<stdio.h>

int sum( int x, int y );

int main(){
        int num1, num2, result;

        printf("Please enter 2 numbers\n");
        scanf("%d%d", &num1, &num2);
        result = sum(num1, num2);
        printf("The sum is %d\n", result);

        return 0;
}

int sum(int x, int y){
        return x+y;
}
```

Example

- Write a C program to compute the area of a circle with radius r.

```c
#include <stdio.h>
#include <math.h>
#define PI 3.141593
// function prototype
double computeArea (double);
int main()
{
   double r, area; // Declare variables.
   // Enter the radius.
   printf("Enter the radius of the circle: \n");
   scanf("%lf",&r);
   area= computeArea(r);//call function
    // Print the value of the area..
   printf("The area of a circle with radius %5.3f is %5.3f. \n",r,area);
   // Exit program.
   return 0;
}
 // Function Definition
double computeArea (double r)
{
    double area;
    // Compute the area of the circle.
    area = PI*pow (r,2);
    return area;
}
```

Example

- Write a C program to compute the circumference of a circle with radius r.

```c
#include <stdio.h>
#define PI 3.141593
//  function prototype
double computeCircumference (double,double);
int main()
{
   double r, circum; //  Declare variables.
    // Enter the radius.
   printf("Enter the radius of the circle: \n");
   scanf("%lf",&r);
   circum= computeCircumference(r,PI); //call function
    // Print the value of the circumference
   printf("The circumference of a circle with radius %5.3f is %5.3f. \n",r,circum);
    // Exit program.
   return 0;
}
 // Function Definition
double computeCircumference (double r,double pi)
{
    double circum;
     // Compute the circumference of the circle.
    circum = 2*pi*r;
    return circum;
}
```

Example

- Write a function prototype and the implementation that computes average, a function that returns the average of its two type double input parameters.

```
//function prototype
double average (double, double );

//the function
double average (double n1, double n2 )
{
  return ((n1 + n2) / 2.0);
}
```

Example

- Rewrite the following mathematical expression using **C math functions**
  - **$x=b^2+c^2-2bc$**

```
double x, b, c;
x= pow(b,2)+pow(c,2)-2*b*c;
```

Example

1. **Write a complete c program to do the following.**

- $Y = x^3 + x^2 + x$

- **Your program should include two functions, cubic to return x to the power of three and square to return x to the power of two.**

```c
#include <stdio.h>
int cubic   (int);
int square  (int);
int main()
{
    int x,y;
    printf("Please enter the value of x: ");
    scanf ("%d",&x);
    y= cubic(x)+ square(x)+x;
    printf("y = %d ",y);
    return 0;
}
int cubic   (int x)
{
  return (x * x * x);
}
int square    (int x)
{
  return (x *x );
}
```

Example

- **Write a complete c program with a function that takes a number and prints it.**

```c
#include <stdio.h>
void printNumber (int);
int main()
{
   int number;
   printf("please enter a number");
   scanf("%d",&number);
   printNumber (number);
   return 0;
}
void printNumber (int x)
{
   printf("%d",x);
}
```

Example

- **Write a complete c program with a function that reads a number and then prints it.**

```c
#include <stdio.h>
void printNumber ();
int main()
{

   printNumber ();
   return 0;
}
void printNumber()
{
   int number;
   printf("please enter a number");
   scanf("%d",&number);
   printf("%d",number);
}
```

Example

- What will be the output if you execute the following C code?

```c
#include <stdio.h>
int f(int , int , int );
int main ()
{
    int q;
    q = f(3, 3, 4);
    printf ("q is %d ", q);
}
int f(int q, int b, int c)
{
      int p;
      p = q * b + 2 * c;
      return (p);
}
```

```
Main function


q
```

```
f function
q=3 , b=3 , c=4
p=??
```

```
Output (screen):
```

Example

- What will be the output if you execute the following C code?

```c
#include <stdio.h>
int f(int , int , int );
int main ()
{
    int q;
    q = f(3, 3, 4);
    printf ("q is %d ", q);
}
int f(int q, int b, int c)
{
      int p;
      p = q * b + 2 * c;
      return (p);
}
```

```
Main function


q
```

```
f function
q=3 , b=3 , c=4
p=??
```

```
Output (screen):

 q is 17
```

Scope of Variables

- The variables that are defined in a function, are only accessible in that function. They are called **Local Variables**.
- When we send parameters to a function, those are defined as local variables of the new function.
- What happens is that when send parameters to a new function, we send a new copy of the values to the new function with another name.

Scope of Variables

```
#include <stdio.h>

void sum(int x, int y);
int main(){
        int x, y;

        printf("enter two values");
        scanf("%d%d", &x, &y);
        sum(x, y);
        printf("function main x=%d, the value of y=%d", x, y);

        return 0;
}
void sum(int x, int y){
        printf("the sum is %d\n", x+y);
        x = -100;
        y = 888;
        printf("function sum x=%d, the value of y=%d", x, y);
}
```

Output