ECE 220 Computer Systems & Programming

Lecture 9 – Functions in C & Run-Time Stack

September 24, 2019

ECE ILLINOIS

ILLINOIS

C Functions

Provides abstraction

- hide low-level details
- give high-level structure to program, easier to understand overall program flow
- enable separable, independent development
- reuse code

Structure of a function

- zero or multiple arguments passed in
- single result returned (optional)
- return value is always a particular type

Making a Function Call in C

```
#include <stdio.h>
/* our Factorial function prototype goes here */
int Fact(int n);

/* main function */
int main() {
    int number;
    int answer;

    printf("Enter a number: ");
    scanf("%d", &number);

    answer = Fact(number); /* function call */
    /* number - argument transferred from main to Factorial */
       answer - return value from Factorial to main */

    printf("factorial of %d is %d\n", number, answer);

    return 0;
}
```

Function “Fact”:

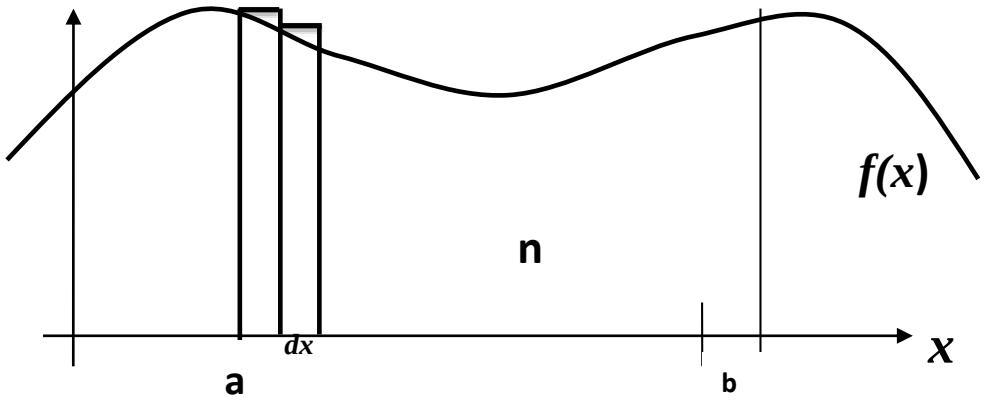
```
/* implementation of Factorial function goes here */
int Fact(int n) {
    int i, result=1; /* local variables in Factorial */

    for (i = 1; i <= n; i++)
        result = result * i;

    return result; /* return value */
}
```

Riemann integral

Adopted from Prof. Kindratenko's notes

Problem statement: write a program to compute integral of a function f(x) on an interval [a,b].

Algorithm: use integral definition as an area under a function *f*(x) on an interval [a,b]



$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} i\right) \frac{b-a}{n}$$

```c
#include <stdio.h>
float f(float x);
float Reimann(int n, float a, float b);
int main()
{
    printf("%f\n", Reimann(100, -1.0f, 1.0f));
     return 0;
}
/* f(x) = x*x+2x+3 */
float f(float x)
{
    return (x * x + 2 * x + 3);
}
/* compute integral of f(x) = x*x+2x+3 on [a,b] */
float Reimann(int n, float a, float b)
{
    float s = 0.0f;              /* computed integral value */
    int i;                       /* loop counter */
    float x, y;                  /* x and y=f(x) */
    float dx = (b - a) / n;      /* width of rectangles */

    for (i = 0; i < n; i++)
    {
        x = a + dx * i;
        y = f(x);
        s += y * dx;
    }
    return s;
}
```

Function that does not return value:

```c
#include <stdio.h>

void PrintBanner();      /* Function declaration */

int main()
{
    PrintBanner();       /* Function call        */
    printf("A simple C program.\n");
    PrintBanner();
}

void PrintBanner()       /* Function definition  */
{
    printf("============================\n");
}
```

*Note: Functions do not necessarily have to be in the same file
(see the github example)

print.h ---> declares the function prototype
main.c ---> call the “print” function
print.c ---> print function

rand(), srand() functions and RAND_MAX macros (stdlib.h)

```c
// C program to generate random numbers
#include <stdio.h>
#include <stdlib.h>

// Driver program
int main(void)
{
    // This program will create same sequence of
    // random numbers on every program run

    for(int i = 0; i<5; i++)
        printf(" %d ", rand());
//rand() returns a random integer value on [0, RAND_MAX]
    return 0;
}
```

Ref:
https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/

```c
// C program to generate random numbers
#include <stdio.h>
#include <stdlib.h>
#include<time.h>

// Driver program
int main(void)
{
    // This program will create different sequence of
    // random numbers on every program run

    // Use current time as seed for random generator
    srand(time(0));

    for(int i = 0; i<5; i++)
        printf(" %d ", rand());

    return 0;
}
```

How about the following “swap” function?

```c
#include <stdio.h>
void swap(int x, int y);
int main()
{
    int x = 1;
    int y = 2;

    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(x, y);

    //Did the swap function work the way you expect?
    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}

void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

Possible Solution

```c
#include <stdio.h>
void swap(int x, int y);
int z, k;

int main()
{
    int x = 1;
    int y = 2;

    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(x, y);

    //Did the swap function work the way you expect?
    printf("After swap: x = %d, y = %d\n", z, k);

    return 0;
}

void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
    z=x;
    k=y;
}
```

Possible Solution (advanced topics coming soon!)

```c
#include <stdio.h>

void swap(int *, int *);

int main()
{
    int x = 1;
    int y = 2;

    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(&x, &y);

    //Did the swap function work the way you expect?
    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}

void swap(int *x, int *y)
{
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;
}
```