Programlamaya Giriş

Fonksiyonlar

Hüseyin Ahmetoğlu

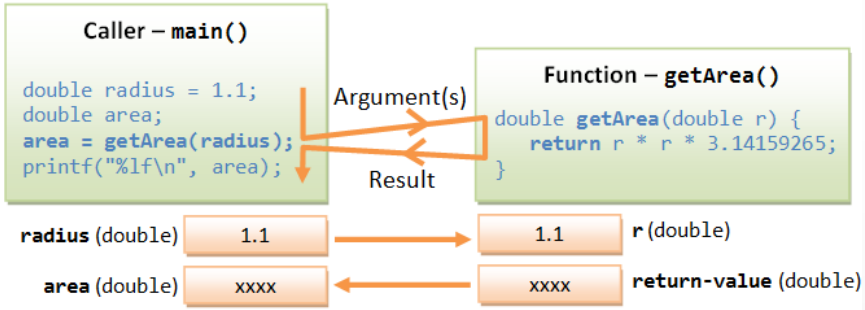
Neden Fonksiyonlar?

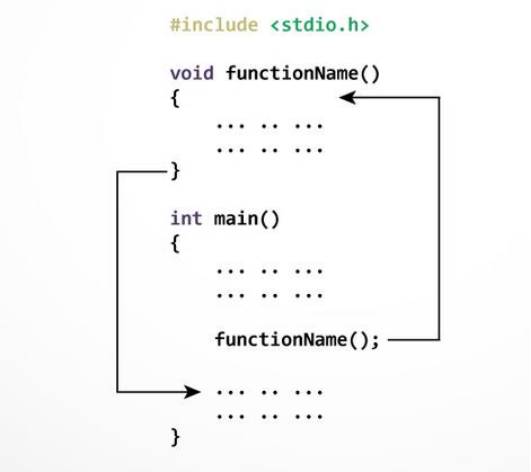
- Bazen kodların belirli bir kısmı birçok kez kullanılmalıdır. Kodları birçok kez yeniden yazmak yerine, onları bir "alt yordama" koymak ve bu "alt yordamı" birçok kez« olarak adlandırmak" - bakım ve anlayış kolaylığı açısından daha iyidir. (Fonksiyon, Alt Yordam, Metot, Yöntem)

Fonksiyon kullanmanın faydaları şunlardır:

- Böl ve fethet: Programı basit, küçük parçalar veya bileşenlerden oluşturun. Programı bağımsız görevler halinde modüler hale getirin.
- Kodları tekrarlamaktan kaçının: Kopyalayıp yapıştırmak kolaydır, ancak tüm kopyaları korumak ve senkronize etmek zordur.
- Yazılımın Yeniden Kullanımı: Diğer programlardaki fonksiyonları kitaplık kodları halinde paketleyerek yeniden kullanabilirsiniz.

Fonksiyonları Kullanma





Fonksiyonları Kullanma

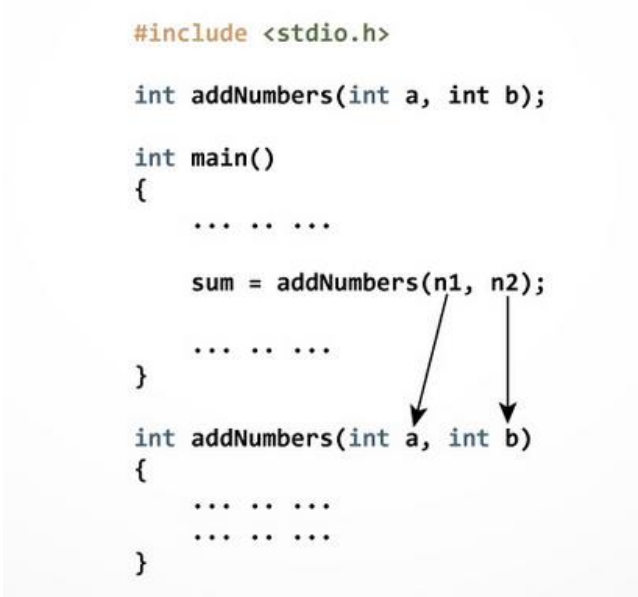
- C'de, bir fonksiyon prototipi bildirmeniz (fonksiyon kullanılmadan önce) ve programlanmış işlemleri içeren bir gövdeyle bir fonksiyon tanımı sağlamanız gerekir.

```c
/* Test Function (TestFunction.c) */
#include <stdio.h>
const int PI = 3.14159265;

// Function Prototype (Function Declaration)
double getArea(double radius);

int main() {
   double radius1 = 1.1, area1, area2;
   // call function getArea()
   area1 = getArea(radius1);
   printf("area 1 is %.2lf\n", area1);
   // call function getArea()
   area2 = getArea(2.2);
   printf("area 2 is %.2lf\n", area2);
   // call function getArea()
   printf("area 3 is %.2lf\n", getArea(3.3));
}

// Function Definition
// Return the area of a circle given its radius
double getArea(double radius) {
   return radius * radius * PI;
}
```

```
area 1 is 3.63
area 2 is 14.52
area 3 is 32.67
```

Fonksiyon Tanımlama

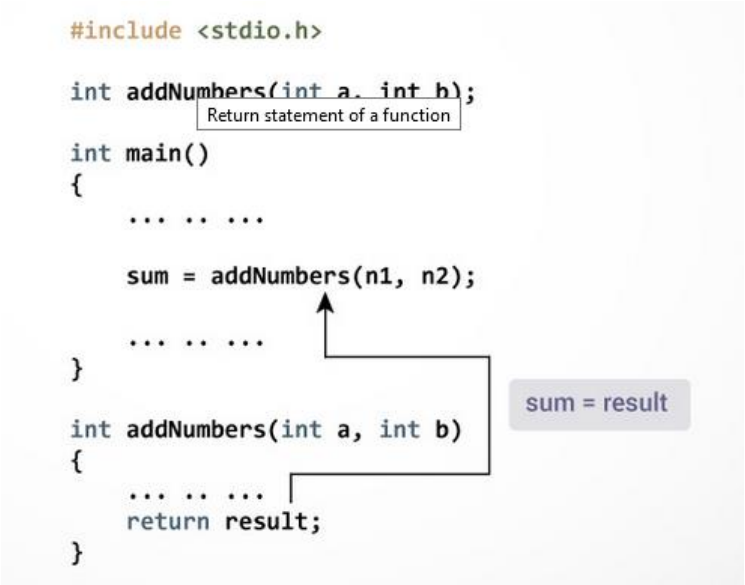
- ParametreListesi, virgülle ayrılmış parametre türü ve parametre adı içerir, yani, param-1-type param-1-name, param-2-type param-2-name,
- ReturnValueType, int veya double gibi dönüş değerinin türünü belirtir. Fonksiyonun hiçbir değer döndürmediğini belirtmek için void adlı özel bir dönüş türü kullanılabilir. C'de, bir Fonksiyonun tek bir değer döndürmesine veya hiç değer döndürmemesine (void) izin verilir. Birden çok değer döndüremez. [C bir dizi döndürmenize izin vermez!]




```
returnValueType functionName ( parameterList ) {
   functionBody ;
}
```

"return"

- Fonksiyon gövdesinin içinde, bir değer (Fonksiyon başlığında bildirilen returnValueType'ın) döndürmek ve çağırana geri göndermek için bir return ifadesi kullanabilirsiniz.
- Bir işlevin adı, bir veya daha fazla kelimeden oluşan bir fiil veya fiil cümlesi (eylem) olacaktır. İlk kelime küçük harfle yazılırken, geri kalanı büyük harfle yazılır. Örneğin, getArea (), setRadius (), moveDown (), isPrime (), vb.




```
return expression;   // Evaluated to a value of returnValueType declared in function's signature
return;              // For function with return type of void
```

Fonksiyon Prototipi

- C'de, bir Fonksiyon çağrılmadan önce bildirilmelidir. Fonksiyon tanımını kullanılmadan önce deklare edilelerek veya sözde bir Fonksiyon prototipi bildirerek elde edilebilir.
- Bir Fonksiyon prototipi, derleyiciye Fonksiyon arayüzünü, yani dönüş türünü, Fonksiyon adını ve parametre türü listesini (parametrelerin sayısı ve türü) söyler. Fonksiyon artık kaynak kodum herhangi bir yerinde tanımlanabilir.

```
// Function prototype - placed before the function is used.
double getArea(double);  // without the parameter name
int max(int, int);
```

Fonksiyon Prototipi

- Fonksiyon prototipine isteğe bağlı olarak parametre adlarını dahil edebilirsiniz. İsimler derleyici tarafından göz ardı edilecek, ancak dokümantasyon görevi görecektir.
- Fonksiyon prototipleri genellikle birlikte gruplanır ve başlık dosyasına yerleştirilir. Başlık dosyası birçok programa dahil edilebilir.

```
// Function Prototype
double getArea(double radius);  // parameter names are ignored, but serve as documentation
int max(int number1, int number2);
```

Fonksiyon Prototipi

```c
/* Testing max function (TestMaxFunction.c) */
#include <stdio.h>

int maximum(int, int); // Function prototype (declaration)

int main() {
   printf("%d\n", maximum(5, 8));  // Call maximum() with literals

   int a = 6, b = 9, c;
   c = maximum(a, b);              // Call maximum() with variables
   printf("%d\n", c);

   printf("%d\n", maximum(c, 99)); // Call maximum()
}

// Function definition
// A function that returns the maximum of two given int
int maximum(int num1, int num2) {
   return (num1 > num2) ? num1 : num2;
}
```

"void"

- Fonksiyonun çağırıldığı yere bir değer döndürmenize gerek kalmadan belirli eylemleri (ör. Yazdırma) gerçekleştirmek için bir Fonksiyona ihtiyacınız olduğunu varsayalım, dönüş değeri türünü geçersiz olarak bildirebilirsiniz.
- Fonksiyonun gövdesinde, bir 'return' kullanabilirsiniz; bu durum isteğe bağlıdır. Eğer kullanılmazsa fonksiyon işlevini yerine getirdiğinde çağrıldığı yere yine dönüş yapacaktır.

Fonksiyonun Yerel Değişkenleri ve Parametrelerinin Kapsamı

- Bir Fonksiyonun içinde bildirilen Fonksiyon parametreleri dahil tüm değişkenler yalnızca Fonksiyonun tarafından kullanılabilir.
- Fonksiyonun çağrıldığında oluşturulur ve Fonksiyonun döndükten sonra serbest bırakılır (yok edilir).
- Bunlar Fonksiyonun için yerel oldukları ve Fonksiyonun dışında kullanılamadıkları için yerel değişkenler olarak adlandırılırlar.
- Otomatik değişkenler olarak da adlandırılırlar, çünkü otomatik olarak oluşturulurlar ve yok edilirler - bunları tahsis etmek ve ayırmak için programcının açık bir eylemi gerekmez.

Boolean Fonksiyonlar

```c
/*
 * Test Boolean function (BooleanfunctionTest.c).
 */
#include <stdio.h>

// Function Prototype
int isOdd(int);

int main() {
   printf("%d\n", isOdd(5));  // 1 (true)
   printf("%d\n", isOdd(6));  // 0 (false)
   printf("%d\n", isOdd(-5)); // 0 (false)
}

int isOdd(int number) {
   if (number % 2 == 1) {
      return 1;
   } else {
      return 0;
   }
}
```

```c
bool isOdd(int number) {
   if (number % 2 == 0) {
      return false;
   } else {
      return true;
   }
}
```

```c
int isEven(int number) {
   return (number % 2 == 0);
}

int isOdd(int number) {
  return !(number % 2 == 0);  // OR return !isEven(number);
}

int main() {
   int number = -9;
   if (isEven(number)) {      // Don't write (isEven(number) != 0)
      printf("Even\n");
   }
   if (isOdd(number)) {      // Don't write (isOdd(number) != 0)
      printf("Odd\n");
   }
}
```

Fonksiyonlar ve Diziler

- Diziler fonksiyonlara parametre olarak verilebilir ancak diziyle birlikte dizinin boyutunun da fonksiyona sunulması parametre olarak geçilmesi gerekmektedir.

```c
/* Function to compute the sum of an array (SumArray.c) */
#include <stdio.h>

// Function prototype
int sum(int array[], int size);    // Need to pass the array size too
void print(int array[], int size);

// Test Driver
int main() {
   int a1[] = {8, 4, 5, 3, 2};
   print(a1, 5);   // {8,4,5,3,2}
   printf("sum is %d\n", sum(a1, 5));  // sum is 22
}

// Function definition
// Return the sum of the given array
int sum(int array[], int size) {
   int sum = 0;
   int i;
   for (i = 0; i < size; ++i) {
      sum += array[i];
   }
   return sum;
}

// Print the contents of the given array
void print(int array[], int size) {
   int i;
   printf("{");
   for (i = 0; i < size; ++i) {
      printf("%d", array[i]);
      if (i < size - 1) {
         printf(",");
      }
   }
   printf("}\n");
}
```

Değere göre ve Referansa göre Parametre Geçişleri

- Değere göre geçişte, bağımsız değişkenin bir "kopyası" oluşturulur ve Fonksiyona aktarılır. Çağrılan Fonksiyon "klon" üzerinde çalışır ve orijinal kopyayı değiştiremez. C'de temel türler (int ve double gibi) değere göre aktarılır.

```c
/* Fundamental types are passed by value into Function (TestPassByValue.c) */
#include <stdio.h>

// Function prototypes
int inc(int number);

// Test Driver
int main() {
   int n = 8;
   printf("Before calling function, n is %d\n", n); // 8
   int result = inc(n);
   printf("After calling function, n is %d\n", n);  // 8
   printf("result is %d\n", result);                // 9
}

// Function definitions
// Return number+1
int inc(int number) {
   ++number;  // Modify parameter, no effect to caller
   return number;
}
```

Değere göre ve Referansa göre Parametre Geçişleri

- Referansla geçişte, arayan değişkeninin bir referansı Fonksiyona aktarılır. Başka bir deyişle, çağrılan işlev aynı veriler üzerinde çalışır. Çağrılan işlev parametreyi değiştirirse, aynı çağırıcı kopyası da değiştirilir.

  C'de diziler referans olarak aktarılır. Yani, çağrılan Fonksiyon içindeki çağıranın dizisinin içeriğini değiştirebilirsiniz - dizileri Fonksiyona geçirmenin yan etkisi olabilir.

  C, Fonksiyonların bir dizi döndürmesine izin vermez. Bu nedenle, bir dizinin içeriğini değiştiren bir Fonksiyon yazmak istiyorsanız (örneğin, bir dizinin öğelerini sıralamak), Fonksiyon içinde ve dışında aynı kopya üzerinde çalışmak için referansa göre geçişe güvenmeniz gerekir. Değere göre geçişte, çağrılan Fonksiyonun bir klon kopya üzerinde çalıştığını ve orijinal kopyayı değiştirmenin bir yolu olmadığı unutulmamlıdır.

- Dizi fonksiyona referans olarak aktarılır. Yani çağrılan fonksiyon, çağıranla aynı dizinin kopyası üzerinde çalışır. Bu nedenle, fonksiyon içindeki dizi değişiklikleri fonksiyonun dışına yansıtılır.

- Dizi, klonlanmış bir kopya kullanılarak değere göre değil, referansa göre aktarılacak şekilde tasarlanmıştır.

- Bunun nedeni, devasa diziyi değere göre geçirmenin verimsiz olmasıdır - büyük dizinin klonlanması gerekir.

```c
/* Function to increment each element of an array (IncrementArray.c) */
#include <stdio.h>

// Function prototypes
void inc(int array[], int size);
void print(int array[], int size);

// Test Driver
int main() {
   int a1[] = {8, 4, 5, 3, 2};

   // Before increment
   print(a1, 5);   // {8,4,5,3,2}
   // Do increment
   inc(a1, 5);     // Array is passed by reference (having side effect)
   // After increment
   print(a1, 5);   // {9,5,6,4,3}
}

// Function definitions

// Increment each element of the given array
void inc(int array[], int size) {  // array[] is not const
   int i;
   for (i = 0; i < size; ++i) {
      array[i]++;  // side-effect
   }
}

// Print the contents of the given array
void print(int array[], int size) {
   int i;
   printf("{");
   for (i = 0; i < size; ++i) {
      printf("%d", array[i]);
      if (i < size - 1) {
         printf(",");
      }
   }
   printf("}\n");
}
```

‘const’ Fonksiyon Parametreleri

- Referans bazında orijinal verileri bozma riskleri. Fonksiyon içindeki dizileri değiştirme niyetiniz yoksa, Fonksiyon parametresinde ‘const anahtar sözcüğünü kullanabilirsiniz. Sabit Fonksiyon bağımsız değişkeni, Fonksiyon içinde değiştirilemez.
- Parametreleri yanlışlıkla değiştirmenizi engellediğinden ve birçok programlama hatasına karşı sizi koruduğundan, referansları iletmek için mümkün olduğunda const kullanılması önemlidir.

```c
/* Search an array for the given key using Linear Search (LinearSearch.c) */
#include <stdio.h>

int linearSearch(const int a[], int size, int key);

int main() {
   const int SIZE = 8;
   int a1[] = {8, 4, 5, 3, 2, 9, 4, 1};

   printf("%d\n", linearSearch(a1, SIZE, 8));  // 0
   printf("%d\n", linearSearch(a1, SIZE, 4));  // 1
   printf("%d\n", linearSearch(a1, SIZE, 99)); // 8 (not found)
}

// Search the array for the given key
// If found, return array index [0, size-1]; otherwise, return size
int linearSearch(const int a[], int size, int key) {
   int i;
   for (i = 0; i < size; ++i) {
      if (a[i] == key) return i;
   }
   return size;
}
```

Matematiksel Fonksiyonlar

- C, <math.h> kitaplığında yaygın olarak kullanılan birçok Matematiksel fonksiyon sağlar.

sin(x), cos(x), tan(x), asin(x), acos(x), atan(x):
Take argument-type and return-type of float, double, long double.

atan2(y, x):
Return arc-tan of y/x. Better than atan(x) for handling 90 degree.

sinh(x), cosh(x), tanh(x):
hyper-trigonometric functions.

pow(x, y), sqrt(x):
power and square root.

ceil(x), floor(x):
returns the ceiling and floor integer of floating point number.

fabs(x), fmod(x, y):
floating-point absolute and modulus.

exp(x), log(x), log10(x):
exponent and logarithm functions.

Rastgele Sayılar Oluşturma

- Stdlib.h üstbilgisi, 0 ile RAND_MAX (dahil) arasında sözde rasgele bir tam sayı üreten bir rand () fonksiyonu sağlar. RAND_MAX, stdlib.h'de tanımlanan bir sabittir (tipik olarak maksimum 16- / 32-bit işaretli tamsayı değeri, örneğin 32767). *Rand () %n* aracılığıyla [0, n) arasında rasgele bir sayı üretebilirsiniz.

  rand (), farklı çağrılarda aynı sözde rasgele sayı dizisini üretir. Stblib.h ayrıca rasgele sayı üretecini başlatmak için bir srand () işlevi sağlar. Tipik olarak, 1 Ocak 1970'den bu yana geçen saniye sayısını döndüren time(0) fonksiyonu (<time.h>) aracılığıyla elde edilen geçerli zamanla başlar.

```c
/* Test Random Number Generation (TestRand.c) */
#include <stdio.h>
#include <stdlib.h>  // for rand(), srand()
#include <time.h>    // for time()

int main() {
   // rand() generate a random number in [0, RAND_MAX]
   printf("RAND_MAX is %d\n", RAND_MAX);   // 32767

   // Generate 10 pseudo-random numbers between 0 and 99
   //   without seeding the generator.
   // You will get the same sequence, every time you run this program
   int i;
   for (i = 0; i < 10; ++i) {
      printf("%d ", rand() % 100);   // need <stdlib.h> header
   }
   printf("\n");

   // Seed the random number generator with current time
   srand(time(0));   // need <cstdlib> and <ctime> header
   // Generate 10 pseudo-random numbers
   // You will get different sequence on different run,
   //   because the current time is different
   for (i = 0; i < 10; ++i) {
      printf("%d ", rand() % 100);   // need <stdlib.h> header
   }
   printf("\n");
}
```

KAYNAKLAR

- Goel, A., & Mittal, A. (2016). Computer Fundamentals and Programming in C (RMK). Pearson Education India. Retrieved from https://www.oreilly.com/library/view/computer-fundamentals-and/9789332579200
- yet another insignificant Programming Notes. (2021, April 08). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming/index.html#Cpp