ALGORİTMALAR VE PROGRAMLAMA II

Hafta 1

Giriş, Kapsama Kuralları ve Rasgele Sayı Üretimi

Derse Giriş

Ders Web Sitesi: http://hru-algpro.github.io/

Kaynaklar:

*“Doç. Dr. Caner ÖZCAN, KBÜ Yazılım Mühendisliği www.canerozcan.net ”

“Algoritma Geliştirme ve Programlamaya Giriş”, Doç. Dr. Fahri Vatansever.

“C how toProgram”, Paul J. Deitel, HarveyDeitel.

“A book on C”,All Kelley, İra Pohl

“A’danZ’yeC Klavuzu’’,KaanAslan.

* Bu ders içeriği ilgili kaynaktan türetilmiştir.

Derse Giriş

- Bol bol pratik yapın!
- Notlandırma
  - Vize Sınavı + Quiz 1 : %40
  - Final Sınavı + Quiz 2 : %60
- Not için değil, öğrenmek için çalışın. Not nasılsa kazanılır.

Dönem Boyunca Görülecek Konular

- Rekürsif Fonksiyonlar
- İşaretçiler (Değer Yoluyla Çağırma, Referans Yoluyla Çağırma, Dinamik Bellek Yönetimi)
- Struct, Enum ve Typedef Tanımlamaları
- Tek Bağlı Doğrusal Listeler
- Sıralama ve Arama Algoritması
- String ve Matematiksel Fonksiyonlar
- Sıralı ve Rasgele Erişimli Dosyalar

Nasıl İyi Yazılımcı Olunur?

- Pratik alışkanlıktır.
- Pratik rutin işlemlerdir.
- Pratik yaparak elde edilir.
- Pratik büyük özveri ve adanmışlık gerektirir.
- Pratik yaparak kazanılmış bazı yetenekler.
  - Atış yapmak
  - Araba sürmek
  - Yazı yazmak

Yazılmış Program Kodlarını Okuyun

- Bir roman yazarı olmak istiyorsanız en iyi yazılmış romanları okumadan roman yazmaya başlayabilir misiniz?
- Eğer bir film senaristi olmak istiyorsanız, en iyi film senaryolarını okumadan olabilir misiniz?
- Bir yazılım mühendisi olmak istiyorsanız program kodlarını okumadan nasıl yazılım mühendisi olabilirsiniz?
- Eğer yazılmış kodları okursanız beğendiğiniz teknikleri siz de kullanabilirsiniz. Yanlış olan yerleri görürseniz siz de o yanlışları yapmayabilirsiniz.

Yazılmış Program Kodlarını Okuyun

- Yazılım kodlarının pek çok özelliği vardır.

- Girintiler, açıklama satırları, isimlendirmeler, fonksiyon yapıları vs.

- Tecrübeli yazılım mühendislerinin yazdıkları kodları inceleyin. Kısa zamanda kendi yazdığınız kodlardan daha iyilerini yazmaya başlayabilirsiniz.

Kod Yazmaya Başlamadan Önce

- Kod yazmaya başlamadan önce dokümanları tamamlayın.
- Mutlaka tasarım yapılmalı.
- Şartnameler, tasarım belgeleri, kısıtlama ve varsayımlar, algoritmalar, akış diyagramları.
- Bugün öğrendiklerinle yarın için hazırlık yapmalısın.

Belirlenmiş Standartları Takip Et

- Belirlenmiş standartları takip ediniz ve kendiniz oluşturmayın.
  - Dosya adlandırma kuralları
  - İşlev ve modül adlandırma kuralı
  - Değişken adlandırma kuralları
  - Tarih, girinti, yorumlar
  - Okunabilirlik yönergeleri
  - Yapılması gerekenler ve yapılmaması gerekenlerin listesi

Kodları Gözden Geçir

- Kodlar gözden geçirilmek üzere yazılmalıdır.
  - Kötü kodlama ve standarda uymama
  - Performansı dikkate alınmamış
  - Tarih, girinti, açıklamalar uygun değil
  - Okunabilirlik zayıf
  - Açık dosyalar kapalı değil Tahsis edilen hafıza serbest değil
  - Çok fazla global değişken ve çok fazla kodlama
  - Zayıf hata yakalama.
  - Modülerlik yok ve tekrarlanan kod.

Program Geliştirme Ortamları

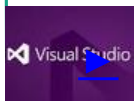
Code::Blocks

Program Geliştirme Ortamları

Download Dev-C++ from http://www.bloodshed.net/dev/devcpp.html and install it.https://en.wikiversity.org/wiki/Installing_and_using_Dev-C%2B%2B

Download the Code::Blocks 17.12 installer. If you know you don't have MinGW installed, download the package which has MinGW bundled.

Download eclipse Neon from http://www.eclipse.org/downloads/Choose Eclipse IDE for C/C++ Developers and install. You should download and install MinGW GCC http://www.mingw.org/



https://visualstudio.microsoft.com/tr/vs/features/cplusplus/

C Programlama ve Fonksiyonlar

- Fonksiyonlar büyük bilgi işlem görevlerini daha küçük görevlere böler.
- Büyük programlar yazmak için bir problem almak ve küçük, yönetilebilir parçalara ayrılmak çok önemlidir.
- İşlevler çağrıldığı yere değerleri döndürür.

tip fonksiyon_adı(parametre listesi)

{açıklama ifadeleri}

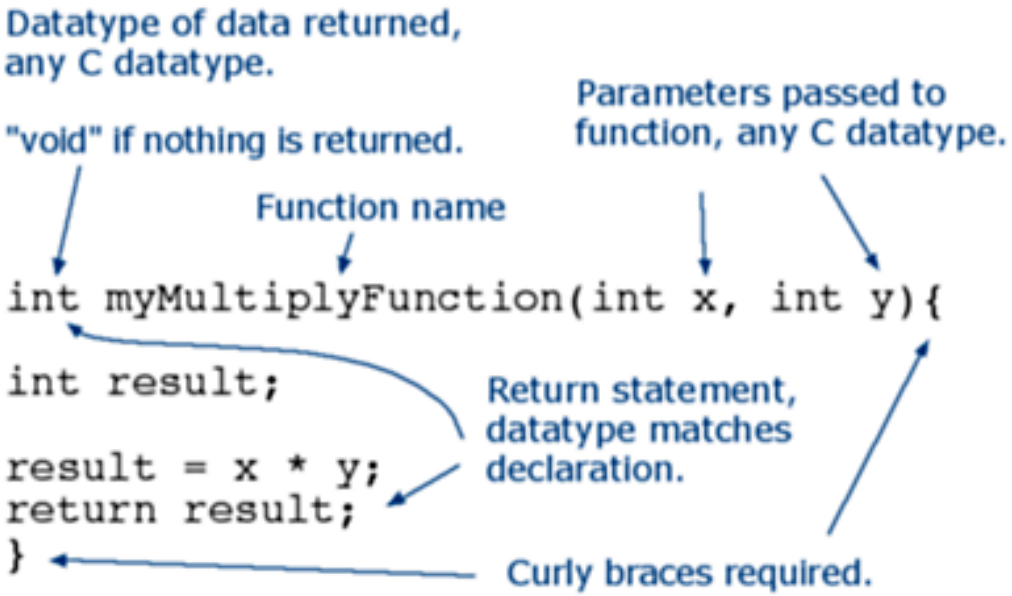
- Parametre listesi, virgülle ayrılmış bildirimler listesidir.

Fonksiyonlar


```
void nothing(void) { }      /* this function does nothing */

double twice(double x)
{
   return (2.0 * x);
}

int all_add(int a, int b)
{
   int   c;
   .....
   return (a + b + c);
}
```

Fonksiyonlar


```
void nothing(void) { }      /* this function does nothing */

double twice(double x)
{
   return (2.0 * x);
}

int all_add(int a, int b)
{
   int   c;
   .....
   return (a + b + c);
}
```



Fonksiyon Return İfadesi

- return; // return++a; // return(a*b)

Bir return ifadesiyle karşılaşıldığında, fonksiyonun yürütülmesi sonlandırılır ve kontrol tekrar çağrılma ortamına geçirilir. Return ifadesi bir ifade içeriyorsa, ifadelerin değeri de çağrılma ortamına iletilir.

Even though a function returns a value, a program does not need to use it.

```
while (.....) {
    getchar();          /* get a char, but do nothing with it */
    c = getchar();      /* c will be processed */
    .....
}
```

Fonksiyonlar


```
#include <stdio.h>

#define   N   7

long    power(int, int);
void    prn_heading(void);
void    prn_tbl_of_powers(int);

int main(void)
{
    prn_heading();
    prn_tbl_of_powers(N);
    return 0;
}
```

Fonksiyonlar


```
void prn_heading(void)
{
   printf("\n::::::  A TABLE OF POWERS  ::::::\n\n");
}

void prn_tbl_of_powers(int n)
{
   int   i, j;

   for (i = 1; i <= n; ++i) {
      for (j = 1; j <= n; ++j)
         if (j == 1)
            printf("%1d", power(i, j));
         else
            printf("%9ld", power(i, j));
      putchar('\n');
   }
}
```

Fonksiyonlar


```
long power(int m, int n)
{
    int     i;
    long    product = 1;

    for (i = 1; i <= n; ++i)
       product *= m;
    return product;
}
```

Here is the output of the program:

```
:::::  A TABLE OF POWERS  :::::

1         1         1          1          1          1          1
2         4         8         16         32         64        128
3         9        27         81        243        729       2187
.....
```

Nesne Faaliyet Alanı

- Faaliyet Alanı (Scope): Bir nesnenin tanınabildiği program aralığı
- Nesnenin faaliyet alanı, program içerisinde tanımlandığı yer ile ilgili

  1-Blok Faaliyet Alanı (BlockScope)

  Yalnızca bir blok içerisinde tanınma

  2-Fonksiyon Faaliyet Alanı (FunctionScope)

  Yalnızca bir fonksiyon içerisinde tanınma

  3-Dosya Faaliyet Alanı (File Scope)

  Tüm dosya içinde yani fonksiyonları tümünde tanınma
- Değişkenler faaliyet alanına göre 3 kısımda incelenir:

  Yerel değişkenler
  Global değişkenler

  Parametre değişkenleri
- **Not: İki değişkenin aynı faaliyet alanı grubuna ait olması (blok, fonksiyon veya dosya) faaliyet alanlarının tamamen aynı olduğu anlamına gelmez.**

Yerel Değişkenler

- Yerel değişkenler blok faaliyet alanı kuralına uyar. Sadece tanımlandıkları blok içerisinde geçerlidirler.

```
.....
{
        int a;
        {
                int b;
        }
}
....
```

b değişkeninin faaliyet alanı

a değişkeninin faaliyet alanı

Yerel Değişkenler

- Farklı faaliyet alanına sahip aynı isimli değişkenler tanımlanabilir.
- Derleyici bu değişkenleri farklı adreslerde tutar. İç bloklarda tanımlı
- aynı isimli değişkenler, dış blok içerisinde tanımlı

aynı isimli değişkenleri maskeler.

- Aynı blok içerisinde aynı isimli iki değişken tanımlanamaz.

```
int main()
{
    int a = 10;
    printf("a = %d\n", a);
    {
        int a = 20;
        printf("a = %d\n", a);
    }
    printf("a = %d\n", a);
    return 0;
}
```

Global Değişkenler

- Bütün blokların dışında tanımlanmış değişkenlerdir.
- Global değişkenler dosya faaliyet alanı kuralına uyar.
- Global değişkene ilk değer verilebilir.
- Aynı isimli global ve yerel değişkenin tanınabilir olduğu blokta ancak yerel değişkene erişilebilir.

```c
#include <stdio.h>
int a;

void fonk1(void)
{
    a = 20;
}

int main()
{
    a = 10;
    printf("a = %d\n", a);
    fonk1();
    printf("a = %d\n", a);

    return 0;
}
```

```c
#include <stdio.h>
int a=10;

void fonk1(void)
{
    a = 40;
    printf("a = %d\n", a);
}

int main()
{
    int a;
    a = 30;
    printf("a = %d\n", a);
    fonk1();
    printf("a = %d\n", a);

    return 0;
}
```

Parametre Değişkenleri

- Fonksiyon parametreleridirler. Fonksiyon faaliyet alanı kuralına
- uyarlar. Sadece parametresi oldukları fonksiyon içerisinde
- geçerlidirler.

```c
#include <stdio.h>
int a=10;

void fonk1(int a)
{
    a = 40;
    printf("a = %d\n", a);
}

int main()
{
    printf("a = %d\n", a);
    fonk1(a);
    printf("a = %d\n", a);

    return 0;
}
```

Nesne Ömrü

- Nesne Ömrü: Nesnelerin faaliyet gösterdiği zaman aralığını tanımlar
- Nesneler statik ve dinamik ömürlü olarak iki kısma ayrılır. Statik
- Ömürlü Nesne:
  - Program bitene kadar faaliyet gösterirler
  - Hafızada **data segment** bölgesinde tutulurlar.
  - 1-Global değişkenler 2-Stringler 3-Statik yerel değişkenler
- Dinamik Ömürlü Nesne:
  - Programın belli bölümünde belli zaman aralığında faaliyet gösterip yok olurlar
  - 1-Yerel değişkenler 2-Parametre değişkenleri 3-Dinamik bellek fonksiyonları ile oluşturulan nesneler
  - Yerel değişkenler ve parametre değişkenleri Stacksegment bölgesinde tutulur.

Nesne Ömrü

- Örn, yerel değişkenler tanımlandıkları blok çalıştığında tanımlanır blok bitince yok olurlar.
- Yani çalışma süresi bloğun çalışma süresi kadardır.
- Statik ömürlü değişkene ilk değer verilmese de değeri 0 olur.
- Dinamik ömürlü değişkenlere ilk değer atanmaz ise hafızada değişken için ayrılan bölgede o anda bulunan değer atanır.
- Örn, global değişken ilk değer atanmaz ise değeri 0, yerel değişkene ilk değer atanmaz ise değeri kestirilemez.

Yer ve Tür Belirleyiciler

- Örn, yerel değişkenler tanımlandıkları blok çalıştığında tanımlanır blok bitince yok olurlar.
- C'de nesnelerin ikincil özellikleri belirleyiciler ile tanımlanır.
- Belirleyiciler: 1-Yer belirleyici 2-Tür belirleyici
- 4 tane Yer Belirleyici vardır (Saklama Sınıfları)

  1-auto 2-register 3-static 4-extern
- 2 tane Tür belirleyici vardır

  1-const 2-volatile
- Genel değişken tanımlama biçimi:

  [yer belirleyici] [tür belirleyici] [tür] nesne;

  Yer belirleyici, tür belirleyici ve tür herhangi bir sıra ile olabilir.

  auto const int a = 10; *(tavsiye edilen)*
  const auto int a = 10;

  int const auto a=10;

Saklama Sınıfları

- C'deki her değişken ve fonksiyonun iki niteliği vardır. Tip ve depolama sınıfı.
- Dört depolama sınıfı auto, extern, registerve static olarak verilir.
- Nesne kendi bloğu içinde oluşturulur ve yok edilir.
  - **auto:**Fonksiyon içinde bildirilen değişkenler varsayılan olarak otomatiktir. Bu değişkenler fonksiyon kapsamında kullanılabilir.
    auto double x, y;

    Stack bölgesinde tutulurlar.
    Global değişkenler ve parametre değişkenleri auto özelliği alamaz.

Saklama Sınıfları

- **extern:** Bloklar ve fonksiyonlar arasında bilgi aktarma yöntemlerinden biri harici değişkenleri kullanmaktır.

  Bir işlevin dışında bir değişken bildirildiğinde, depolama birimi kendisine kalıcı olarak atanır ve depolama sınıfı

  extern olur.

  C derleyicisi başka bir modülde tanımlı fonksiyonu otomatik olarak extern kabul eder. Fonksiyonlar için extern kullanımı gereksizdir.

  extern ile tanımlanan değişkene başlangıç değeri verilmez

  ise derleyici tarafından hafızada yer ayrılmaz.

Örnekler: auto& extern


```
#include <stdio.h>
extern int a = 1, b = 2;
c = 3;
int f(void);

int main(void) {
        printf("%3d\n", f());
        printf("%3d%3d%3d\n", a, b, c);
        return 0;
}

int f (void)
{
        auto int b, c;
        a = b = c = 4;
        return(a + b + c);
}
```

Örnekler: auto & extern

- Extern'inbu kullanımı derleyiciye bu dosyada veya başka bir dosyada "başka yerde arama" yapmasını söylemek için kullanılır.

example1.c

```
#include <stdio.h>
İnt a = 1, b = 2; c = 3;
int f (void);
int main(void) {

    printf("%3d\n", f());
    printf("%3d%3d%3d\n", a, b, c);
    return 0;



}
```

file2.c

```
int f(void) {
      extern a;
      int b, c;
      b = c = a;
      return (a + b + c);
}
```

Saklama Sınıfları

- **register:** Saklama sınıfı registerderleyiciye ilişkilendirme değişkenlerinin yüksek hızlı bellek kayıtlarında saklanması gerektiğini söyler.

- Değişkenin bellekte değil de CPU'nun yazmaçlarında tutulacağını belirtir.
- Değişkenlerin yazmaçta tutulması programın hızlanmasını sağlar.

C kodu	Assembly
data3 = data1+data2	MOV reg, data1
	ADD reg, data2
	MOV data3, reg

- Belleğe erişim, yazmaçlara erişimden daha yavaştır. Çünkü belleklere erişim için belli bir makine zamanı gerekir.
- Yazmaçlar sınırlı sayıdadır.

Saklama Sınıfları

- **static:** Fonksiyonlar içinde tanımlanan yerel değişkenlerdir.

  Fonksiyon sonlandıktan sonra değişken değeri saklanır.

  Static belirleyiciye sahip değişkenler programın çalışması boyunca bellekten kaybolmazlar. auto'nunzıttıdır.

  Sadece tanımlandıkları fonksiyonda geçerlidirler.

  Statik bildirimlerin iki önemli ve farklı kullanımı vardır.

  Bunlardan biri, bloğa yeniden girildiğinde yerel bir değişkenin önceki değerini korumasına izin vermektir.

  İkinci ve daha önemli kullanımı dış tanımlamalarla bağlantılıdır. Değişkenin kapsamını kısıtlamak için

  kullanılır. Statik yerel ve global değişkenler **data segment** bölgesinde tutulur.

Örnek: register


```
#include <stdio.h>
#include <time.h>
int a =1; #define N
10000 int
main(void) {
      clock_t start, end; double cpu_time_used;
      register double i; start = clock();
      for(i=0;i<N;i=i+0.0001); end = clock();
      cpu_time_used = ((double) (end -start)) /
      CLOCKS_PER_SEC; printf("Running time is
      %f",cpu_time_used);



            return 0;
}
```

- Running time is 0,163 second with register variable.
- Running time is 0,419 second without register variable.

Example: static


```
void f(void)
{
    static int   cnt = 0;

    ++cnt;
    if (cnt % 2 == 0)
        .....                   /* do something */
    else
        .....                   /* do something different */
}
```

Example: static


```
void f(void)
{
   .....      /* v is not available here */
}

static int     v;     /* static external variable */

void g(void)
{
   .....      /* v can be used here */
}
```

Example: static


```
static int   g(void);          /* function prototype */

void f(int a)                  /* function definition */
{                              /* g() is available here, */
    .....                      /* but not in other files */
}

static int g(void)             /* function definition */
{
    .....
}
```

Örnek


```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void ); /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void ); /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main( void )
13 {
14    int x = 5; /* local variable to main */
15
16    printf("local x in outer scope of main is %d\n", x );
17
18    { /* start new scope */
19       int x = 7; /* local variable to new scope */
20
21       printf( "local x in inner scope of main is %d\n", x );
22    } /* end new scope */
23
```

Örnek


```
24      printf( "local x in outer scope of main is %d\n", x );
25
26      useLocal(); /* useLocal has automatic local x */
27      useStaticLocal(); /* useStaticLocal has static local x */
28      useGlobal(); /* useGlobal uses global x */
29      useLocal(); /* useLocal reinitializes automatic local x */
30      useStaticLocal(); /* static local x retains its prior value */
31      useGlobal(); /* global x also retains its value */
32
33      printf( "\nlocal x in main is %d\n", x );
34      return 0; /* indicates successful termination */
35  } /* end main */
36
37  /* useLocal reinitializes local variable x during each call */
38  void useLocal( void )
39  {
40      int x = 25; /* initialized each time useLocal is called */
41
42      printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
43      x++;
44      printf( "local x in useLocal is %d before exiting useLocal\n", x );
45  } /* end function useLocal */
46
```

Örnek


```
47  /* useStaticLocal initializes static local variable x only the first time
48     the function is called; value of x is saved between calls to this
49     function */
50  void useStaticLocal( void )
51  {
52     /* initialized only first time useStaticLocal is called */
53     static int x = 50;
54
55     printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
56     x++;
57     printf( "local static x is %d on exiting useStaticLocal\n", x );
58  } /* end function useStaticLocal */
59
60  /* function useGlobal modifies global variable x during each call */
61  void useGlobal( void )
62  {
63     printf( "\nglobal x is %d on entering useGlobal\n", x );
64     x *= 10;
65     printf( "global x is %d on exiting useGlobal\n", x );
66  } /* end function useGlobal */
```

Örnek


```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

Rastgele Sayı Üretimi

rand fonksiyonu

- <stdlib.h> kütüphanesi yüklenmelidir 0 ile RAND_MAX (en
- düşük 32767-maksdeğeri 16 bit tamsayı)

- RAND_MAX <stdlib.h> içinde tanımlanmış sembolik sabittir.

- 0 ve RAND_MAX arasındaki her sayı seçilmek için eşit olasılığa sahiptir.
- rand ile üretilecek rastgele sayı aralığı uygulama ihtiyacına göre değişir.

Rastgele Sayı Üretimi

- Yazı tura programı sadece yazı için 1 tura için 0'a ihtiyaç duyar.
- Zar kullanan bir program 1 ile 6 arasında rastgele sayı üretmelidir.
- **Ölçekleme:**
  - Randtarafından üretilen değerler daima 0 ile RAND_MAX arasındadır.

    0 ≤ rand( ) ≤ RAND_MAX
  - 0 ile 5 arasında sayı üretmek için randfonksiyonu ile % kalan operatorükullanılır. Buna ölçekleme denir.

    rand( ) % 6

Rastgele Sayı Üretimi

- 6 sayısı ölçekleme faktörüdür. Aralığı
- kaydırmak için sonuca 1 eklenir.
  - **randNumber= 1 + rand( ) % 6**

    1 ≤ randNumber≤ 6 aralığında sayılar üretir.
- Genel kural:
  - **n = a + rand( ) % b ;**

    a kaydırma değeridir.
    b ölçekleme faktörüdür

Rastgele Sayı Üretimi


```
1  /* Fig. 5.7: fig05_07.c
2     Shifted, scaled integers produced by 1 + rand() % 6 */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     int i; /* counter */
10
11    /* loop 20 times */
12    for ( i = 1; i <= 20; i++ ) {
13
14       /* pick random number from 1 to 6 and output it */
15       printf( "%10d", 1 + ( rand() % 6 ) );
16
17       /* if counter is divisible by 5, begin new line of output */
18       if ( i % 5 == 0 ) {
19          printf( "\n" );
20       } /* end if */
21    } /* end for */
22
23    return 0; /* indicates successful termination */
24 } /* end main */
```

Rastgele Sayı Üretimi

- Örnek Program: 6000 kere zar atma 1'den 6'ya kadar
- tamsayılar yaklaşık olarak 1000 defa seçilmelidir.

```
1  /* Fig. 5.8: fig05_08.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     int frequency1 = 0; /* rolled 1 counter */
10    int frequency2 = 0; /* rolled 2 counter */
11    int frequency3 = 0; /* rolled 3 counter */
12    int frequency4 = 0; /* rolled 4 counter */
13    int frequency5 = 0; /* rolled 5 counter */
14    int frequency6 = 0; /* rolled 6 counter */
15
16    int roll; /* roll counter, value 1 to 6000 */
17    int face; /* represents one roll of the die, value 1 to 6 */
18
```

Rastgele Sayı Üretimi


```
19      /* loop 6000 times and summarize results */
20      for ( roll = 1; roll <= 6000; roll++ ) {
21         face = 1 + rand() % 6; /* random number from 1 to 6 */
22
23         /* determine face value and increment appropriate counter */
24         switch ( face ) {
25
26            case 1: /* rolled 1 */
27               ++frequency1;
28               break;
29
30            case 2: /* rolled 2 */
31               ++frequency2;
32               break;
33
34            case 3: /* rolled 3 */
35               ++frequency3;
36               break;
37
38            case 4: /* rolled 4 */
39               ++frequency4;
40               break;
41
```

Rastgele Sayı Üretimi


```
42            case 5: /* rolled 5 */
43               ++frequency5;
44               break;
45
46            case 6: /* rolled 6 */
47               ++frequency6;
48               break; /* optional */
49         } /* end switch */
50      } /* end for */
51
52      /* display results in tabular format */
53      printf( "%s%13s\n", "Face", "Frequency" );
54      printf( "    1%13d\n", frequency1 );
55      printf( "    2%13d\n", frequency2 );
56      printf( "    3%13d\n", frequency3 );
57      printf( "    4%13d\n", frequency4 );
58      printf( "    5%13d\n", frequency5 );
59      printf( "    6%13d\n", frequency6 );
60      return 0; /* indicates successful termination */
61   } /* end main */
```

```
Face    Frequency
   1         1003
   2         1017
   3          983
   4          994
   5         1004
   6          999
```

Rastgele Sayı Üretimi

- **rand** fonksiyonu gerçekte sahte rastgelelik üretir.
- **rand**fonksiyonunu defalarca çağırmak rasgeleymiş gibi görünen bir sayı dizisi oluşturur.
- Ancak, bu sayı dizisi her program çalıştırıldığında kendi kendini tekrarlar.
- Her program çalışmasında farklı dizilimde gerçek rastgele sayılar üretebilmek için **srand** fonksiyonu kullanılır.
- **srand** argüman olarak bir tamsayı alır.
- **srand** randfonksiyonunun programın her çalışmasında farklı dizilimde sayılar üretilmesini sağlar.

Rastgele Sayı Üretimi


```
1  /* Fig. 5.9: fig05_09.c
2     Randomizing die-rolling program */
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     int i; /* counter */
10    unsigned seed; /* number used to seed random number generator */
11
12    printf( "Enter seed: " );
13    scanf( "%u", &seed ); /* note %u for unsigned */
14
15    srand( seed );  /* seed random number generator */
16
17    /* loop 10 times */
18    for ( i = 1; i <= 10; i++ ) {
19
20       /* pick a random number from 1 to 6 and output it */
21       printf( "%10d", 1 + ( rand() % 6 ) );
22
```

Rastgele Sayı Üretimi


```
23          /* if counter is divisible by 5, begin a new line of output */
24          if ( i % 5 == 0 ) {
25             printf( "\n" );
26          } /* end if */
27       } /* end for */
28
29       return 0; /* indicates successful termination */
30    } /* end main */
```

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

```
Enter seed: 867
        2       4       6       1       6
        1       1       3       6       2
```

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

Rastgele Sayı Üretimi

- Her seferinde seed değeri girmeden rastgele sayı üretmek için

```
srand( time (NULL));
```

- Sistem saati otomatik olarak seeddeğeri elde etmek için kullanılır.
- **time**fonksiyonu 1970 Ocak ayı gece yarısından itibaren kaç saniye geçtiğini döndürür.
- time fonksiyonu için **<time.h>** kütüphanesi kullanılır.

Şans Oyunu: Craps

Kurallar:

- İki zar atılır
- Zarların toplamı hesaplanır
- İlk atışta 7 veya 11 gelirse oyuncu kazanır
- İlk atışta 2, 3 veya 12 gelirse oyuncu kaybeder
- İlk atışta 4,5,6,8,9,10 gelirse bu oyuncunun puanı oluyor.
- Oyuncu 7 atmadan önce kendi puanını tutturmalıdır.

#CrapsGAME

KEEP
CALM
AND
CODE
ON

Gelecek Hafta

- C Programlarının Bellek Düzeni
- Rekürsif (Özyinelemeli) Fonksiyonlar

S o r u l a r ?

Dinlediğiniz için teşekkürler