Algoritmalar ve Programlama I – BLM 103

Hafta 14: Önişlemler ve Diğer Konular

Fenerbahçe Üniversitesi

14. Hafta İçeriği

- #include, #define Önişlemleri
- # ve ## Operatörleri
- Değişken Sayıda Argümanlar
- Uygulamayı exit ve atexit ile Sonlandırmak
- Uçucu (Valotile) tipi

Önişlemler

- C dilinde, program derlenmeden önce derleyiciye çeşitli talimatlar vermek mümkündür.

- Bazı talimatlar, mevcut c dosyasına derlenme esnasında başka dosyaların eklenmesini, bazı sabit sayıların tanımlanmasını içermektedir.

- Bu talimatlar # işareti ile başlamaktadır.

Önişlemler - #include

- İnclude talimatı derlenme işlemi esnasında kaynak koduna, eklenecek olan kütüphaneyi belirtmektedir.
- İki kullanım çeşiti vardır:

```
#include <dosyaadi>
#include "dosyaadi"
```

- Eğer include işlemi " ile yapıldıysa, derleyici derlenecek olan kod ile aynı klasörün içerisinde arayacaktır.

Önişlemler - #include

- " ile include kullanımı genellikle, başka bir yazılımcının kodunu kendi koduna ekleme amacıyla kullanılır.

- < > arasında kullanımda ise, derleyicinin ve sistem klasörlerinde ilgili kütüphane aranır.

Önişlemler - #define

- #define talimanı sembolik sabitler tanımlamak için kullanılır.
- #define talimat formatı:
    - #define isim değer
- Program derlenme aşamasında iken, programın içerisinde kullanılmış tüm tanımlamalar (yukarıdaki isim değeri gibi), tanımlandığı değer ile değiştirilir. Yani örnekte verilen isim tanımının geçtiği tüm yerler değer ile değiştirilir.

Önişlemler - #define

- Örnek
    - #define PI 3.14159
    - #define SAAT 24

Önişlemler - #define

- Define talimatı makrolar için de kullanılabilir.
- Örnek:
    - #define CEMBER_ALAN (x) ((PI) * (x) * (x))

Önişlemler - #define

- Define talimatı makrolar için de kullanılabilir.
- Örnek:
  - #define CEMBER_ALAN (x) ((PI) * (x) * (x))
  - alan = CIRCLE_AREA(4);

=

  - alan = ((3.14159) * (4) * (4));

Önişlemler - #define

- Makro kullanmanın avantajı
  - Fonksiyon çağırmanın maliyeti yoktur
  - Kod halen okunabilirliği azalmamıştır.

Koşullu Derleme

- Derleme işlemi esnasında bazı satırların bir koşula bağlı olarak derleme işlemine dahil edilip edilmeyeceğinin kontrolünün yapılmasını sağlayan bir yapı bulunmaktadır.

- Örneğin

```
#ifdef DEBUG
   printf( "Variable x = %d\n", x );
#endif
```

Tanımlaması yapıldığında, DEBUG isimli bir tanım define talimatı ile verilmemiş ise, printf derlemeye dahil edilmeyecektir.

ve ## operatörleri

- C dilinde önişlemlerde kullanılmak üzere # ve ## operatörleri mevcuttur.
- Alınan girdiyi, metine dönüştürmektedir.

- Örmek
    - #define HELLO(x) printf("Hello, " "#x " "\n");

- HELLO(test) çağrıldığında oluşan kod

    - printf("Hello, " "test" "\n");

ve ## operatörleri

- İki argümanı birleştirmektedir.:
    - #define TOKENCONCAT(x, y) x ## y

- Örnek
    - TOKENCONCAT(O, K) OK olarak birleştirilmiştir.

Giriş ve Çıkış Yönlendirme

- Varsayılan ayarlarda girişler klavyeden (stantard-in), çıkışlar (standard-out) ise monitör'e yapılmaktadır.
- Linux ve Windows gibi işletim sistemlerinde giriş ve çıkışı farklı yerlere yönlendirmek mümkündür.
- Uygulama derlendikten sonra komut satırında, bu işlemler gerçekleştirilebilmektedir.

Giriş ve Çıkış Yönlendirme

- Derlenmiş olan test.exe isimli bir uygulamanın girişleri komut satırından kayıtlı bir dosyadan verilebilir.

  Komut satırından

  - test.exe > cikti.txt

    komutu ile cikti.txt isimli dosyaya yazdırılabilir.

Giriş ve Çıkış Yönlendirme

- Derlenmiş olan test.exe isimli bir uygulamanın girişleri komut satırından kayıtlı bir dosyadan verilebilir.

  Komut satırından

  - test.exe < girdi.txt

    komutu ile girdi.txt isimli dosyadan alınabilir.
    Programdaki scanf satırı dosyadan okuma yapacaktır.

Giriş ve Çıkış Yönlendirme

- Bir programın çıktıları diğer bir programa giriş olarak verilebilir.
- Bunun için | işaret kullanılmaktadır.
- Örnek kullanım:
- test1.exe | test2.exe

Bu kullanımda, test1 exe'nin ürettiği çıkışlar, test2.exe'nin girişlerine beslenmiştir.

Giriş ve Çıkış Yönlendirme

- Bir programın çıktılarının dosyaya sürekli eklenmesi (Mevcut yazılmış metinlerin üzerine yazmaması için) için

- Örnek kullanım:
- test1.exe >> ciktilar.txt

ciktilar.txt dosyasına sürekli yeni veriler eklenmektedir. Append modu ile açılmaktadır.

Değişken Uzunluklu Argüman Listesi

- Fonksiyonlar tanımlanırken, kaç argümanının olacağı belli olmayan bir fonksiyon tanımlamak mümkündür.
- Örnek
  - Printf

- printf fonksiyonunun prototipi
  - int printf(const char *format, ...);

- … bir fonksiyon prototipinde kaç argümanın olacağının belli olmadığını ifade etmekedir.

Değişken Uzunluklu Argüman Listesi

```
 3  #include <stdio.h>
 4  #include <stdarg.h>
 5
 6  double average( int i, ... ); /* prototype */
 7
 8  int main( void )
 9  {
10     double w = 37.5;
11     double x = 22.5;
12     double y = 1.7;
13     double z = 10.2;
14
15     printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
16        "w = ", w, "x = ", x, "y = ", y, "z = ", z );
17     printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
18        "The average of w and x is ", average( 2, w, x ),
19        "The average of w, x, and y is ", average( 3, w, x, y ),
20        "The average of w, x, y, and z is ",
21        average( 4, w, x, y, z ) );
22     return 0; /* indicates successful termination */
23  } /* end main */
```

Değişken Uzunluklu Argüman Listesi

```
24
25  /* calculate average */
26  double average( int i, ... )
27  {
28     double total = 0; /* initialize total */
29     int j; /* counter for selecting arguments */
30     va_list ap; /* stores information needed by va_start and va_end */
31
32     va_start( ap, i ); /* initializes the va_list object */
33
34     /* process variable length argument list */
35     for ( j = 1; j <= i; j++ ) {
36        total += va_arg( ap, double );
37     } /* end for */
38
39     va_end( ap ); /* clean up variable-length argument list */
40     return total / i; /* calculate average */
41  } /* end function average */
```

Değişken Uzunluklu Argüman Listesi

```
w = 37.5
x = 22.5
y = 1.7
z = 10.2

The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

Programın Sonlandırılması

- Exit fonksiyonu programı kapanmaya zorlamaktadır.
- Fonksiyon, programda bir giriş hatasının tespitinde veya bir dosyanın açılamadığında genellikle kullanılır.
- atexit fonksiyonu programın exit ile kapandığı zamanki çağrılacak olan fonksiyonu belirtmektedir.

Programın Sonlandırılması

```
 3  #include <stdio.h>
 4  #include <stdlib.h>
 5
 6  void print( void ); /* prototype */
 7
 8  int main( void )
 9  {
10     int answer; /* user's menu choice */
11
12     atexit( print ); /* register function print */
13     printf( "Enter 1 to terminate program with function exit"
14        "\nEnter 2 to terminate program normally\n" );
15     scanf( "%d", &answer );
16
17     /* call exit if answer is 1 */
18     if ( answer == 1 ) {
19        printf( "\nTerminating program with function exit\n" );
20        exit( EXIT_SUCCESS );
21     } /* end if */
22
23     printf( "\nTerminating program by reaching the end of main\n" );
24     return 0; /* indicates successful termination */
25  } /* end main */
```

Programın Sonlandırılması

```
26
27  /* display message before termination */
28  void print( void )
29  {
30     printf( "Executing function print at program "
31        "termination\nProgram terminated\n" );
32  } /* end function print */
```

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
1

Terminating program with function exit
Executing function print at program termination
Program terminated
```

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
2

Terminating program by reaching the end of main
Executing function print at program termination
Program terminated
```

Volatile Kullanımı

- C derleyicileri, verilen kaynak kodu optimize ederler. Bazen kodun bazı bölümlerinin optimize edilmesi istenmeyen bir durumdur.

- Bunun için kodun optimize edilmesinin istenmeyen değişkenlerinin başına volatile ifadesi yazılarak optimizasyon yaptırılmaz.

Volatile Kullanımı

- Örneğin
  int x = 0;
  while(x==0);

İfadesinde, döngü hiçbir zaman çıkmaz. Derleyici bu ifadeye bakıp, x'i sürekli kontrol ettirmek yerine işlemciyi bekleme durumuna alabilir. Ancak bir gömülü sistemde ortak kullanımlı bir RAM kullanılıyor olabilir. Bu durumda X'in değeri başka bir işlemci tarafından adresine erişilerek değiştirilebilir. Buradaki optimizasyon nedeniyle x'e bakılmadığı için bekleme modundan çıkmayan bir uygulama ortaya çıkabilir.

goto Kullanımı

- Assembly dillerinde olduğu gibi etiketler kullanılarak, belirli bir etikete doğrudan atlamak için kullanılırlar.

- Xyz:

…

goto Xyz;

goto Kullanımı

```
 3  #include <stdio.h>
 4
 5  int main( void )
 6  {
 7     int count = 1; /* initialize count */
 8
 9     start: /* label */
10
11        if ( count > 10 ) {
12           goto end;
13        } /* end if */
14
15        printf( "%d  ", count );
16        count++;
17
18        goto start; /* goto start on line 9 */
19
20     end: /* label */
21        putchar( '\n' );
22
23     return 0; /* indicates successful termination */
24  } /* end main */
```