**getchar() ve putchar() fonksiyonları**

Klavyeden girilen bir karakteri okuyarak bellekteki bir değişkene atamak için **getchar()** fonksiyonu, ekrana tek bir karakter yazdırmak için ise, **putchar()** fonksiyonu kullanılır. **stdio.h** başlık dosyasını kullanan bu iki fonksiyonun genel yapısı prototipi aşağıda gösterilmektedir:

int getchar(void);

int putchar(int id);

getchar() fonksiyonunu kullanarak klavyeden karakter girmemiz gerekirken, karakter girmek yerine ENTER tuşuna basarsak, getchar() fonksiyonu \n karakterini geri verir.

getchar() fonksiyonu klavyeden girdiğimiz karakteri, int bir değere çevrilen unsigned char bir değer olarak okur. Programlar, klavyeden okunan değeri genellikle char bir değişkene atar. Bu durumda int değerin en solda kalan bit'i devre dışı kalır.

Ancak, getchar() fonksiyonu int bir değer geri verir, çünkü klavyeden karakter okunurken bir hata meydana geldiğinde getchar() fonksiyonunun geri verdiği **EOF** makrosu genellikle -1 olan negatif bir int değerdir.

EOF makrosu **stdio.h** başlık dosyası içinde tanımlıdır. Sonuç olarak, klavyeden yapılan okuma işleminde bir hata meydana geldiğinde getchar() fonksiyonunun EOF değerini geri vermesi için int bir değer geri vermesi gerekir.

Şimdi, aşağıdaki örneği inceleyerek getchar() fonksiyonunun yaptığı işlemi incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

char cd1, cd2;

printf("İki karakter girip ENTER tuşuna basınız: ");

cd1 = getchar();

cd2 = getchar();

printf("Girdiğiniz karakterler: %c %c", cd1, cd2);

return 0;

}

Yukarıdaki örnekte, program çalıştığında bir imleç yanıp sönmeye başlar. Bu durumda, program klavyeden 2 karakter girilmesini bekler. Bir karakter girildiğinde, karakter yanıp sönen imleç yerine ekrana yazılır ve imleç bu defa ekrana yazılan karakterin hemen sağında yanıp sönmeye devam eder. İkinci karakter girildiğinde, karakter yine ekrana yazılır ve imleç bir sağa kayar. ENTER tuşuna bastığımızda, girdiğimiz karakterler cd1 ve cd2 değişkenlerine atanır. Bu durumda ENTER tuşuna basmak yerine daha fazla karakter girilse bile, fazladan girilen karakterlerin herhangi bir etkisi olmaz. Yine ilk iki karakter değişkenlere atanır. Son olarak, program printf() fonksiyonunu kullanarak değişken değerlerini ekrana yazar. Eğer, tek bir karakter girip ENTER tuşuna basarsanız, program cd2 değişkenine herhangi bir değer atamaz, sadece cd1 değişken değerini ekrana yazar.

putchar() fonksiyonu ise, ekrana tek bir karakter yazar. putchar() fonksiyonunun parametresi int bir değerdir, ancak bu değer fonksiyon tarafından unsigned char bir değere çevrilir. Eğer işlem başarılı olursa, putchar() fonksiyonu parametre olarak verilen karakteri, aksi takdirde EOF makro değerini geri verir.

Şimdi getchar() ve putchar() fonksiyonlarının birlikte kullanıldığı örnekleri incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

char cd;

printf("Tek bir karakter girip ENTER tuşuna basınız: ");

cd = getchar();

printf("Girilen karakter: %c", putchar(cd));

return 0;

}

Program, klavyeden girilen karakteri getchar() fonksiyonu ile cd değişkenine atar ve putchar() fonksiyonu ile değişken değerini ekrana yazar.

**getche(), getch() ve kbhit() fonksiyonları**

**conio.h** başlık dosyasını kullanan getche() ve getch() fonksiyonlarının genel yapıları aşağıdaki gösterilmektedir:

int getche(void);

int getch(void);

getche() fonksiyonu tıpkı getchar() fonksiyonu gibi klavyeden yazılan tek bir karakter okur. Klavyeden bir karakter girilir girilmez okur, bir bellek değişkenine atar. Ayrıca ENTER tuşuna basılmasına gerek yoktur. Klavyeden girilen karakter unsigned char bir değer olarak okunur ve int bir değere çevrilir. Sonuçta, klavyeden okunan değer char bir değişkene atanır. Bu fonksiyonu bir örnek üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

#include <conio.h>

int main(void)

{

char cd;

printf("Bir karakter giriniz: ");

cd = getche();

printf("\n");

putchar(cd);

return 0;

}

Program, bir karakter girildiğinde, girilen karakteri ekrana yazar ve cd değişkenine atar. putchar() fonksiyonunu kullanarak cd değişken değerini ekrana yazar.

getch() fonksiyonu da tıpkı getche() fonksiyonu gibi klavyeden yazılan tek bir karakter okur. getch() fonksiyonunu getche() fonksiyonundan ayıran en önemli özellik girdiğiniz karakteri ekrana yazmamasıdır. Şimdi, bu fonksiyonu da bir örnek üzerinde inceleyelim:

Örnek

#include <stdio.h>

#include <conio.h>

int main(void)

{

char cd;

printf("Bir karakter giriniz: ");

cd = getch();

printf("\n");

putchar(cd);

return 0;

}

Yukarıdaki örnekte, program bir karakter girdiğinizde, girdiğiniz karakteri ekrana yazmadan cd değişkenine atar. Son olarak program putchar() fonksiyonunu kullanarak cd değişken değerini ekrana yazar.

**getchar()**: Girilen karakteri okuma işlemi için ENTER tuşuna basılması gerekir.

**getche()**: Karakter girilir girilmez okuma işlemi gerçekleşir.

**getch()**: getche() fonksiyonundan tek farkı girilen karakterin ekrana yazılmamasıdır.

Standart kütüphane fonksiyonlarından biri de kbhit() fonksiyonudur. Bu fonksiyonun genel yapısı aşağıdaki gösterilmektedir:

int kbhit(void);

**conio.h** başlık dosyasını kullanan kbhit() fonksiyonu bir tuşa basılıp basılmadığını belirler. Eğer bir tuşa basılırsa, kbhit() fonksiyonu sıfır olmayan bir değer geri verir, ancak girilen karakteri okumaz. Eğer bir tuşa basılmazsa, kbhit() fonksiyonu 0 değerini geri verir.

Şimdi, kbhit() fonksiyonunun kullanılmasını bir örnek ile incelemeye çalışalım:

Örnek

#include <stdio.h>

#include <conio.h>

int main(void)

{

for ( ; ; ) {

printf("%d ", 21);

if (kbhit()) break; // 1

}

return 0;

}

Program, sonsuz bir for döngüsü içine girer ve sürekli olarak 21 sayısını ekrana yazmaya başlar. Klavyeden bir tuşa basıldığında, kbhit() fonksiyonu 0 olmayan bir değer geri vereceğinden, 1 sayısı ile gösterilen işlem satırındaki break komutu devreye girdiğinden, döngü ve program sona erer.

**fgets() ve fputs() fonksiyonları**

Klavyeden girilen karakter dizilerini okutmak için fgets() fonksiyonunu ve karakter dizilerini ekrana yazmak için puts() fonksiyonunu kullanabiliriz. **stdio.h** başlık dosyasını kullanan her iki fonksiyonun genel yapıları aşağıda gösterilmekte:

char\* fgets(char \*str, int count, FILE \*stream);

int fputs(const char \*str, FILE \*stream);

fgets() fonksiyonu, en fazla count parametre değerinin 1 eksiği kadar karakter okur ve okuduğu karakterleri str parametresi ile gösterilen karakter dizisine atar. ENTER tuşuna basıldığında, belirtilen sayıda karakter okumadan işlem sona erer. Klavyeden değer atamak için stream parametresi için stdin değeri kullanılır. Klavyeden Okunan karakterler str ile gösterilen karakter dizisine atandıktan sonra, str sonuna otomatik olarak, dizi sonu belirlemek üzere, NULL bir karakter ('\0') eklenir. Girilebilecek maksimum karakter sayısından (count-1) daha az karakter girilirse, giriş işlemini sona erdiren yeni satır karakteri (10), '\0' karakterinden önce diziye eklenir. Başarı durumunda, fonksiyon str parametresinin içeriğini geri döndürür. Eğer bir hata oluşursa, NULL bir işaretçi döndürülür.

fputs() fonksiyonu, str parametresi ile gösterilen ve boş bir sonlandırma karakteri ('\0') ile biten karakter dizisini ekrana yazar. Karakter dizisinin sonunda yer alan '\0' karakteri yazılmaz. Başarı durumunda, negatif olmayan bir değer geri döndürür. Hata durumunda EOF değeri geri döndürülür ve akış hata göstergesi ayarlanır.

Şimdi, fgets() ve fputs() fonksiyonlarının kullanılmasını bir örnek üzerinde çalışalım:

Örnek

#include <stdio.h>

unsigned int bg\_strlen(const char \*str);

int main(void)

{

char cdizi[20];

char \*cp;

const char \*cpstr;

unsigned int len;

// Klavye girişini okuma

printf("Bir karakter dizisi giriniz: ");

// ENTER tuşuna basılıncaya kadar karakter girişi devam eder,

// ancak sadece ilk 19 karakter işlem görür. Fazla karakter girilse bile dikkate alınmaz.

cp = fgets(cdizi, 20, stdin);

// cdizi dizisine aktarılan karakter sayısı

len = bg\_strlen(cdizi);

printf("cdizi dizisine aktarılan karakter sayısı: %d\n", len);

// 0 ('\0') ve 10 ('\n') karakterlerine kadar olan cdizi içeriğini yazma

for (cpstr = cp; \*cpstr && \*cpstr!='\n'; ) {

printf("%c", \*cpstr++);

}

// 0 ('\0') ve 10 ('\n') karakterlerini yazma

printf(" %d", (int) \*cpstr);

// Eğer girilen karakter sayısı dizi boyutunun bir eksiğinden az ise

// giriş yapılan karakterlerin sonuna eklenen '\n' (10) karakteri yerine '\0' karakteri konur.

if (len<(sizeof(cdizi)-1)) {

printf(" %d", (int) \*(cpstr+1));

cdizi[len] = '\0'; // Dizide yer alan '\n' karakterini devre dışı bırakmak için

}

printf("\n");

// 0 ('\0') karakterine kadar olan cdizi içeriğini yazma

for (cpstr = cp; \*cpstr; ) {

printf("%c", \*cpstr++);

}

printf("\n");

fputs(cdizi, stdout);

return 0;

}

unsigned int bg\_strlen(const char \*str)

{

const char \*s;

for (s = str; \*s && \*s!=10; ++s);

return(s - str);

}

Program klavyeden girilen bir karakter dizisini fgets() fonksiyonu ile okuyarak, cdizi adlı diziye atar. bg\_strlen() fonksiyonu ile, cdizi dizisine aktarılan karakter sayısını ('\n' ve '\0' karakterleri hariç) hesaplayarak len adlı değişkene atar. Önce, '\n' ve '\0' karakterleri hariç olmak üzere dizi içeriğini, daha sonra varsa '\n' karakterini ve '\0' karakterini ekrana yazar. Eğer '\n' karakteri varsa, yerine '\0' karakteri koyar. Dizi içeriğini bir for döngüsü ve fputs() fonksiyonu ile iki kez ekrana yazar.

**printf() fonksiyonu**

printf() fonksiyonunun genel yapısı aşağıdaki gösterilmektedir:

int printf (char \*kontrol-dizisi, par1, par2, ...);

printf() fonksiyonunun ilk parametresi **kontrol-dizisi** ifadesi ile gösterilen ve kontrol dizisi adı verilen bir karakter dizisidir. printf() fonksiyonu, " " işaretleri arasında yer alan bu karakter dizisini ekrana yazar.

Kontrol dizisinin içinde hem normal karakterler hem de (**%**) işareti ile başlayan **format tanımlayıcıları** adı verilen ifadeler yer alabilir. printf() fonksiyonu " " işaretleri arasında yer alan normal karakterleri sıra ile ekrana yazar. Kontrol dizisi içinde yer alan her bir format tanımlayıcısı için par ifadesi ile gösterilen bir değer tanımlanır. Ancak, burada dikkat edilmesi gereken 2 önemli nokta vardır:

1. Kontrol dizisi içinde yer alan format tanımlayıcısı kadar par ifadesi ile gösterilen argüman tanımlanmalıdır.

2. Her argümanın veri türü kendisine karşılık gelen format tanımlayıcısının veri türüne uygun olmalıdır.

par ifadesi ile gösterilen argümanlar bir değişken, sabit, ifade veya bir fonksiyon çağrısı olabilir.

printf() fonksiyonu, ekrana yazdığı karakter sayısını int bir değer olarak geri verir. printf() fonksiyonunun çalışmasında bir hata meydana gelirse, fonksiyon negatif bir sayı geri verir.

printf() fonksiyonunun çalışma prensibini farklı kelimelerle tekrar ifade etmek gerekirse:

Fonksiyon çalışmaya başladığında, önce kontrol dizisi içinde yer alan ilk karaktere bakar. Eğer normal bir karakter ise hemen ekrana yazar. Eğer bir format tanımlayıcısı ise, bu format tanımlayıcısına karşılık gelen ilk argüman değerini ekrana yazar. Kontrol dizisi içindeki bütün karakterleri sıra ile bu şekilde kontrol edip işlem yaptıktan sonra çalışması sona erer.

Format tanımlayıcıları kendilerine karşılık gelen argümanların ekrana nasıl yazılacağını belirler. printf() fonksiyonu format tanımlayıcıları ile karşılaştığında ekrana farklı bir veri yazacağını ve bu verinin printf() fonksiyonu çağrılırken karakter dizisinden sonra tanımlanmış olan argüman değerlerinden sıra ile alınacağını anlar.

printf() fonksiyonu ile kullanılmakta olan format tanımlayıcıları aşağıdaki tabloda gösterilmektedir:

|  |  |
| --- | --- |
| **Format tanımlayıcısı** | **Karşılığı** |
| c | Karakter |
| d | signed int |
| i | signed int |
| e | Bilimsel gösterim |
| E | Bilimsel gösterim |
| f | Ondalıklı sayı |
| g | e ve %f'den kısa olanı kullanır. |
| G | E ve %f'den kısa olanı kullanır. |
| o | unsigned 8'lik sayı |
| s | Karakter dizisi |
| u | unsigned int |
| x | unsigned 16'lık sayı (Küçük harf) |
| X | unsigned 16'lık sayı (Büyük harf) |
| p | İşaretçi |
| n | Karşılık gelen argüman ekrana yazılan karakterlerin sayısının atandığı int bir değeri gösteren bir işaretçi olmalıdır. |
| % | % işaretini ekrana yazar. |

%e veya %E format tanımlayıcılarını kullanarak, float sayı değerleri bilimsel bir şekilde ekranda gösterilebilir. Bunlarla birlikte l(double) ve L(long double) değiştirici ifadelerini kullanarak sırası ile double ve long double değerleri de bilimsel bir şekilde ekranda gösterilebilir.

Şimdi, anlattıklarımızı bir örnek üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

float fd1, fd2;

fd1 = fd2 = 726.548317;

printf("%e\n%E", fd1, fd2);

return 0;

}

Yukarıdaki programı derleyip çalıştırdığımızda, aşağıdaki ifadeleri ekrana yazar:

7.265483e+002

7.265483E+002

Şimdi, bir format tanımlayıcısının genel yapısını ele alıp, her bir elemanını ayrı ayrı incelemeye çalışalım:

**Format tanımlayıcı genel yapısı**

**%**[**göstergeler**][**genişlik**][**.hassasiyet**][**uzunluk**]**format tanımlayıcısı**

Bir format tanımlayıcısı toplam 6 farklı ifadeden oluşmaktadır. En baştaki % işareti ile en sonda yer alan format tanımlayıcı harfi mutlaka kullanılmalıdır. [] işaretleri arasında yer alan diğer dört ifade isteğe bağlı olarak kullanılır.

**scanf() Fonksiyonu**

scanf() fonksiyonunun genel yapısı aşağıda gösterilmektedir:

int scanf (char \*kontrol dizisi,...);

Nokta işaretleri fonksiyon parametrelerini gösterir. scanf() fonksiyonu klavyeden girilen verileri okuyarak belirli değişkenlere atar.

Burada, dikkat edilmesi gereken 2 önemli nokta vardır:

1. Format parametresi ile gösterilen kontrol dizisi içinde yer alan format tanımlayıcısı kadar par ifadesi ile gösterilen parametre tanımlanmalıdır.

2. Her parametrenin veri türü kendisine karşılık gelen format tanımlayıcısının veri türüne uygun olmalıdır.

scanf() fonksiyonuna geçirilen ilk argüman " " işaretleri ile arasında tanımlanan bir karakter dizisidir. **Kontrol dizisi** olarak adlandırılan bu karakter dizisinin içinde hem normal karakterler hem de (%) işareti ile başlayan format tanımlayıcıları adı verilen ifadeler yer alabilir. scanf() fonksiyon format tanımlayıcıları, scanf() fonksiyonunun, kontrol dizisinden sonra yer alan argümanlar (değişken adresleri) tarafından adresleri gösterilen değişkenlere klavyeden okuduğu bilgileri hangi yapıda atayacağını belirler. scanf() fonksiyonunun kontrol dizisi içinde tanımlanan format tanımlayıcı sayısı kadar argüman tanımlanması gerekir. scanf() fonksiyonu bir format tanımlayıcısı ile karşılaştığında, klavyeden bir veri okuyacağını ve bu verinin format tanımlayıcısına karşılık gelen argümanda adresi gösterilen değişkene atanacağını anlar.

scanf() fonksiyonu bellek adresleri argüman olarak verilen değişkenlere atanan değer sayısını geri verir. Fonksiyonun çalışmasında herhangi bir hata meydana gelirse, EOF değerini geri verir.

**Format içinde yer alan her bir değerin genel yapısı**

**%**[**\***][**genişlik**][**uzunluk**]**format tanımlayıcısı**

Bu fonksiyon için bir format tanımlayıcısı toplam 5 farklı ifadeden oluşmaktadır. En baştaki % işareti ile en sonda yer alan format tanımlayıcı harfi mutlaka kullanılmalıdır. [] işaretleri arasında yer alan diğer üç ifade isteğe bağlı olarak kullanılır.

**%** : Mutlaka tanımlanması gereken bu karakter format parametresinin değeri içinde yer alan ve fonksiyona geçirilecek parametrelere atanacak değerlerin yapısını gösteren her bir değerin başlangıcını gösterir.

**\*** : İsteğe bağlı olarak tanımlanan bu karakter verilerin akıştan okunması gerektiğini ancak göz ardı edildiğini, yani karşılık gelen argümanda depolanmadığını gösterir.

**genişlik** : İsteğe bağlı olarak tanımlanan bu değer geçerli okuma işleminde okunacak maksimum karakter sayısını gösterir.

**uzunluk** : İsteğe bağlı olarak tanımlanan bu değer okutulacak argümanın boyutunu gösterir. hh, h, l, ll, j, z, t, L değerlerinden biri kullanıldığında, karşılık gelen argümanın gösterdiği kaydetme tipini değiştirir.

scanf() fonksiyonu ile kullanılmakta olan format tanımlayıcıları aşağıdaki tabloda yer almaktadır:

|  |  |
| --- | --- |
| **Format tanımlayıcı** | **Karşılığı** |
| c | Karakter |
| d | signed int |
| i | signed int |
| e | Bilimsel gösterim |
| E | Bilimsel gösterim |
| f | Ondalıklı sayı |
| g | e ve %f'den kısa olanı kullanır. |
| G | E ve %f'den kısa olanı kullanır. |
| o | unsigned 8'lik sayı |
| s | Karakter dizisi |
| u | unsigned int |
| x | unsigned 16'lık sayı (Küçük harf) |
| X | unsigned 16'lık sayı (Büyük harf) |
| p | İşaretçi |
| n | Okunan karakter sayısını içeren int bir değer alır. |
| [ ] | Bir karakter setini tarar. |
| % | % karakterini ifade eder. |

Şimdi, klavyeden girilen int bir değerin scanf() fonksiyonu ile okunmasını bir örnek üzerinde incelemeye çalışalım:

Örnek

#include <stdio.h>

int main(void)

{

int id;

printf("int bir değer giriniz: ");

scanf("%d", &id);

printf("Girdiğiniz değer: %d", id);

return 0;

}

Program, klavyeden girilen int bir değeri ekrana yazar.