Standart (Hazır) Kütüphane

Bilindiği üzere **ANSI C**, C programlama dili için uluslararası bir standart olarak kabul edilir ve dile ait temel kuralları ve yapıları belirler. Turbo C, Microsoft C ve UNIX tabanlı C derleyicileri gibi çeşitli derleyiciler ANSI C standardını destekleyerek kullanıcıların yazdığı kodların farklı platformlarda sorunsuz çalışmasını mümkün kılar.

Her C derleyicisi, temel standartların yanı sıra kendi özel özelliklerini ve kütüphanelerini sunar. Bu ek özellikler, belirli platformlara özgü işlevsellik veya kullanım kolaylığı sağlamak için tasarlanmıştır. Örneğin, ANSI C'nin standart kütüphaneleri grafik fonksiyonlarını içermez; bu tür özellikler dilin kapsamı dışında bırakılmıştır. Ancak Turbo C, kullanıcılarına grafik fonksiyonları içeren özel bir kütüphane sunar ve bu, grafiksel uygulamalar geliştirmek isteyen programcılar için büyük bir avantaj sağlar.

ANSI C standart kütüphanelerini temel fonksiyonlar olarak aşağıda gösterilen şekilde gruplamak mümkündür:

- Standart Giriş/Çıkış fonksiyonları
- Matematiksel fonksiyonlar
- Zaman ve Tarih fonksiyonları
- Karakterler üzerinde işlem yapan fonksiyonlar
- Metin dizileri üzerinde işlem yapan fonksiyonlar
- Genel amaçlı fonksiyonlar

Standart kütüphanede bu fonksiyonların dışında başka işlemler için de fonksiyonlar bulunmaktadır. Bunlar ise şu başlıklar altında verilebilir.

- Esnek argüman aktarım fonksiyonları
- Yerel olmayan Atlama (Goto) fonksiyonlar
- Ayrıcalıklı Durum (Exceptional handler) Kontrolü ile ilgili komutlar
- Hata ayıklama (Diagnostics) fonksiyonları
- Yerelleştirme (Localization) fonksiyonları

Standart Kütüphane listesi

C dilinde 29 hazır kütüphane bulunmaktadır. Bunlar aşağıda listelenmiştir.

İsim	Açıklama
<assert.h>	Bir programda hata ayıklarken mantıksal hataları ve diğer hata türlerini tespit etmeye yardımcı olmak için kullanılan assert makrosunu bildirir.
<complex.h>	Karmaşık sayıları işlemek için bir dizi işlev tanımlar.
<ctype.h>	Karakterleri türlerine göre sınıflandırmak veya kullanılan karakter kümesinden (tipik olarak ASCII veya uzantılarından biri, ancak EBCDIC kullanan uygulamalar da bilinmektedir) bağımsız bir şekilde büyük ve küçük harf arasında dönüşüm yapmak için kullanılan işlevler kümesini tanımlar.
<errno.h>	Kütüphane fonksiyonları tarafından bildirilen hata kodlarını test etmek için.
<fenv.h>	Kayan nokta ortamını kontrol etmek için bir dizi fonksiyon tanımlar.
<float.h>	Kayan nokta kütüphanesinin uygulamaya özgü özelliklerini belirten makro sabitleri tanımlar.
<inttypes.h>	Tam genişlikli tamsayı türlerini tanımlar.
<iso646.h>	Çeşitli standart simgeleri ifade etmek için alternatif yollar uygulayan birkaç makro tanımlar. ISO 646 değişken karakter kümelerinde programlama için.
<limits.h>	Tamsayı türlerinin uygulamaya özgü özelliklerini belirten makro sabitlerini tanımlar.
<locale.h>	Yerelleştirme fonksiyonlarını tanımlar.
<math.h>	Yaygın matematiksel fonksiyonları tanımlar.
<setjmp.h>	Yerel olmayan çıkışlar için kullanılan setjmp ve longjmp makrolarını bildirir.
<signal.h>	Sinyal işleme fonksiyonlarını tanımlar.

İsim	Açıklama
<stdalign.h>	Nesnelerin hizalanmasını sorgulamak ve belirtmek için.
<stdarg.h>	Fonksiyonlara aktarılan farklı sayıdaki argümanlara erişim için.
<stdatomic.h>	İş parçacıkları arasında paylaşılan veriler üzerinde atomik işlemler için.
<stdbool.h>	Boolean veri tipini tanımlar.
<stddef.h>	Çeşitli faydalı türleri ve makroları tanımlar.
<stdint.h>	Tam genişlikli tamsayı türlerini tanımlar.
<stdio.h>	Temel giriş ve çıkış işlevlerini tanımlar.
<stdlib.h>	Sayısal dönüştürme işlevlerini, rasgele sayı üretme işlevlerini, bellek tahsisini, işlem kontrol işlevlerini tanımlar.
<stdnoreturn.h>	Geri dönmeyen fonksiyonları belirtmek için.
<string.h>	Dizgi işleme işlevlerini tanımlar
<tgmath.h>	Genel tip matematiksel fonksiyonları tanımlar.
<threads.h>	Çoklu iş parçacıklarını, muteksleri ve koşul değişkenlerini yönetmek için işlevler tanımlar.
<time.h>	Tarih ve zaman işleme işlevlerini tanımlar
<uchar.h>	Unicode karakterleri işlemek için tipler ve fonksiyonlar.
<wchar.h>	Geniş dize işleme işlevlerini tanımlar.
<wctype.n>	Geniş karakterleri türlerine göre sınıflandırmak veya büyük ve küçük harf arasında dönüşüm yapmak için kullanılan fonksiyonlar kümesini tanımlar

Standart Giriş/Çıkış fonksiyonları – ön bilgi

C dilinde giriş ve çıkış işlemleri, bir dosyaya veya çevre birimine erişim için tampon (stream) adı verilen bir arabellek üzerinden gerçekleştirilir. Bu tampon, veri akışını düzenlemek ve performansı artırmak amacıyla kullanılan bir ara depolama alanıdır. Bir dosya açıldığında, bu dosya için otomatik olarak bir tampon atanır ve tüm giriş/çıkış işlemleri bu tampon aracılığıyla gerçekleştirilir. Dosya kapatıldığında ise bu tampon belleği serbest bırakılarak sistemde başka işlemler için kullanılabilir hale gelir.

Dosya işlemleriyle ilgili tüm bilgiler, **FILE tipi** adı verilen özel bir veri yapısında saklanır. Bu yapı, dosyaya erişim için gerekli olan bilgilerin tümünü bir arada tutar. Hangi üyeleri içerdiği kullanılan sisteme bağlı olmakla birlikte, temel üyeler genellikle aynıdır. Bu üyeler, dosya ile ilgili durum bilgilerini ve işlevselliği tanımlamak için gereklidir. Ancak, birçok sistem bu temel üyelerin yanı sıra ek özellikler sağlayan üyeler de içerebilir. Bu yapı, programcıların dosyalarla daha etkili bir şekilde etkileşim kurmasını sağlar. FILE veri tipi şu şekilde tanımlanmıştır:

```
typedef struct{
    int cnt;                    /*arabellekteki kullanılmamış yer sayısı*/
    unsigned char *b_ptr;       /* bir sonraki erişim için*/
    unsigned char *base;        /* arabellek başlangıcı */
    int bufsize;                /* arabellek boyutu */
    short flags;                /* her bir biti bayrak olarak kullanılır*/
    char fd;                    /* dosya tanımlayıcısı */
} FILE;
```

C programlarında en sık kullanılan fonksiyonlar, standart giriş ve çıkış işlemleri için tasarlanmış olanlardır. Bu fonksiyonların çoğu, **stdio.h** adlı başlık dosyasında tanımlanmış ve giriş-çıkış işlemleri sırasında kullanılan FILE, size_t gibi yeni veri türlerini ve ilgili makroları içerir. Ayrıca, tamsayı olmayan türlere sahip fonksiyonların bildirimleri de bu başlık dosyasında yer alır.

Bir programda standart giriş-çıkış fonksiyonlarını kullanmak için, programın bildirim kısmında **#include <stdio.h>** ifadesini yazmak gereklidir.

Standart Giriş/Çıkış fonksiyonları

stdio.h başlık kütüphanesinde üç yeni değişken tipi, birkaç simgesel sabit ve veri giriş/çıkış işlemleri için çeşitli fonksiyonlar tanımlanmaktadır.

Yeni veri tipleri: pos_t (long), size_t (unsigned), va_list (char *)

Simgesel sabitler:

BUFSIZE	arabellek boyu
EOF	dosya sonu karakteri (-1)
FİLENAME_MAX	dosya adının en fazla karakter sayısı
FOPEN_MAX	aynı anda açılabilecek dosya sayısı
NULL	Boş adresi (NULL) gösterir (0 ya da 0L)
_IOFBF, _IOLBF, _IONBF	setvbuf() için tam tamponlu (0), satır tamponlu ve tamponsuz (0x04 (0x80)
SEEK_SET, SEEK_CUR, SEEK_END	fseek() ,için konum ayarla (0), mecvut konum (1) ve dosya sonu konum (2)
stdin, stdout, stderr	Standart giriş, çıkış ve hata işaretçileri

Fonksiyonlar:

Kütüphanede giriş/çıkış işlemleri için tanımlanmış 41 fonksiyon bulunmaktadır. Bunlar alfabetik olarak şöyle sıralanır:

clearerr	fgetc	fputc	fseek	getchar	putchar	scanf	tmpfile
fclose	fgetpos	fputs	fsetpos	gets	puts	setbuf	tmpnam
feof	fgets	fread	ftell	perror	remove	setvbuf	ungetc
ferror	fopen	freopen	fwrite	printf	rename	sprintf	vfprintf
fflush	fprintf	fscanf	getc	putc	rewind	sscanf	vprintf
							vsprintf

Detaylı bilgi için: https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

Matematiksel fonksiyonlar

math.h başlık kütüphanesinde çeşitli matematiksel fonksiyonlar ve bazı simgesel sabitler tanımlanmıştır. Bu kütüphanede bulunan tüm fonksiyonlar argüman olarak **double** alır ve sonuç olarak **double** döndürür. Kütüphane içinde üstel fonksiyonlar, kuvvet fonksiyonları, trigonometrik fonksiyonlar, hiperbolik fonksiyonlar, hata ve gama fonksiyonları, En yakın tamsayı kayan nokta işlemleri, kayan nokta manipülasyon fonksiyonları şeklinde çeşitli fonksiyonlar mevcuttur.

Simgesel sabitler:

EDOM	Error Domain, matematiksel fonksiyonun tanımlandığı etki alanının dışında olması.
HUGE_VAL	Ondalık sayılarla ifade edilemeyen durumları belirtir.
ERANGE	Bir girdinin matematiksel fonksiyonun tanımlandığı aralığın dışında olması.

C dilinde, bir fonksiyona gelen parametreler geçerli sınırların dışındaysa, bu durum bir hata olarak işlenir ve **errno.h** başlık dosyasında tanımlanmış olan *errno* adlı değişkene EDOM değeri atanır ve bu bir hata durumunu belirtir. Ayrıca bir fonksiyonun hesapladığı sonuç, çift duyarlı bir sayı (double) ile ifade edilemeyecek kadar büyük veya küçük olduğu durumda *errno* değişkenine ERANGE değeri atanır.

Fonksiyonlar:

Kütüphanede sıklıkla kullanılan fonksiyonların bir kısmı şöyledir:

acos(x)	cbrt(x)	expm1(x)	fmax(x, y)	lgamma(x)	logb(x)	nexttoward(x, y)	scalbn(x, y)
acosh(x)	ceil(x)	erf(x)	fmin(x, y)	llrint(x)	lrint(x)	pow(x, y)	sin(x)
asin(x)	copysign(x, y)	erfc(x)	fmod(x, y)	llround(x)	lround(x)	remainder(x, y)	sinh(x)
asinh(x)	cos(x)	fabs(x)	frexp(x, y)	log(x)	modf(x, y)	remquo(x, y, z)	sqrt(x)
atan(x)	cosh(x)	fdim(x)	hypot(x, y)	log10(x)	nan(s)	rint(x)	tan(x)
atan2(y, x)	exp(x)	floor(x)	ilogb(x)	log1p(x)	nearbyint(x)	round(x)	tanh(x)
atanh(x)	exp2(x)	fma(x, y, z)	ldexp(x, y)	log2(x)	nextafter(x, y)	scalbln(x, y)	tgamma(x)
Detaylı bilgi için: https://www.w3schools.com/c/c_ref_math.php							trunc(x)

Zaman ve Tarih fonksiyonları

time.h başlık kütüphanesinde üç yeni değişken tipi, birkaç simgesel sabit ve tarih ve saat işlemleri için çeşitli fonksiyonlar tanımlanmaktadır. sistem saatine erişim, zaman bilgilerini işleme ve farklı biçimlerde tarih ve saat bilgileri oluşturma gibi işlevleri sağlar. Özellikle zaman ölçümleri, gecikme işlemleri veya zaman bazlı algoritmalar için oldukça faydalıdır.

Veri tipleri: time_t (long), clock_t (unsigned), struct tm , size_t (unsigned)

```
struct tm {
    int tm_sec;     /* saniye için 0..59 */
    int tm_min;     /* dakika için 0..59 */
    int tm_hour;    /* saat için 0..23 */
    int tm_mday;    /* gün için 1..31 */
    int tm_mon;     /* ay için 0..11 */
    int tm_year;    /* yıl için 1900... */
    int tm_wday;    /* Haftanın günleri için 0(pazar)..6 */
    int tm_yday;    /* yılın günleri için 0..365 */
    int tm_isdat;   /* >0 ise yaz saati uygulanıyor */
}
```

Simgesel sabitler:

CLOCKS_PER_SEC	bir saniyede işlemci saati tıklama sayısını belirtir (clock_t)
NULL	geçersiz bir göstericiyi temsil eder (-1)

Fonksiyonlar:

Kütüphanede zaman tarih işlemleri için tanımlanmış 10 fonksiyon bulunmaktadır. Bunlar alfabetik olarak şöyledir:

asctime	clock	ctime	difftime	gmtime
localtime	mktime	strftime	time	wcsftime

Detaylı bilgi için: https://www.tutorialspoint.com/c_standard_library/time_h.htm

time.h kullanımına ait örnek kodlar

Geçerli tarih ve saat öğrenme

```c
#include <stdio.h>
#include <time.h>

int main() {
   time_t now = time(NULL); // Geçerli zamanı al.

   // Zamanı okunabilir bir metin olarak yazdır.
   printf("Şu anki zaman: %s", ctime(&now));

   return 0;
}
```

```
Şu anki zaman: Mon Dec 30 12:05:59 2024
```

İki zaman arasındaki fark

```c
#include <stdio.h>
#include <time.h>

int main() {
   time_t bas, son, zfark;
   bas = time(NULL);

   // Biraz bekleyelim.
   for (volatile int i = 0; i < 100000000; i++);
   son = time(NULL);
   zfark = difftime(son, bas);

   printf("Geçen zaman: %.2f saniye\n", zfark);
   return 0;
}
```

```
Geçen zaman: 0.00 saniye
```

Zamanı formatlama

```c
#include <stdio.h>
#include <time.h>

int main() {
   time_t simdi = time(NULL); // Geçerli zamanı al.
   struct tm *yerel = localtime(&simdi);

   char zaman[50];
   strftime(zaman, sizeof(zaman), "Tarih: %d/%m/%Y - Saat: %H:%M:%S", yerel);

   printf("%s\n", zaman);

   return 0;
}
```

```
Tarih: 30/12/2024 - Saat: 12:19:49
```

Karakterler üzerinde işlem yapan fonksiyonları

ctype.h başlık kütüphanesi karakterlerin türlerini analiz etmek ve dönüştürmek için kullanılan bir kütüphanedir. Bu kütüphane, karakterlerin belirli bir kategoriye ait olup olmadığını kontrol eden işlevler ve küçük-büyük harf dönüşümü gibi işlemler için fonksiyonlar içerir. Genellikle metin işleme ve karakter analizinde kullanılır.

Simgesel sabitler: _ISalnum (alfanümerik), _ISalpha (alfabetik), _Isdigit (sayısal) karakter sınıfı

Karakter setleri

1.	Rakamlar	{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } tam sayılar kümesidir .
2.	Onaltılık basamaklar	{ 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f } kümesidir.
3.	Küçük harfler	{ a b c d e f g h i j k l m n o p q r s t u v w x y z } küçük harfler kümesidir.
4.	Büyük harfler	{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z } büyük harfler kümedir.
5.	Harfler	Küçük ve büyük harflerden oluşan kümedir.
6.	Alfanümerik karakterler	Rakamlar, küçük harfler ve büyük harflerden oluşan bir kümedir.
7.	Noktalama karakterleri	{! “ # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~} semboller kümesidir
8.	Grafik karakterler	Alfasayısal ve noktalama karakterlerinden oluşan kümedir.
9.	Boşluk karakterleri	Sekme, yeni satır, dikey sekme, form besleme, satır başı ve boşluktan oluşan kümedir.
10.	Yazdırılabilir karakterler	Alfasayısal, noktalama karakterleri ve boşluk karakterlerinden oluşan kümedir.
11.	Kontrol karakterleri	ASCII'de bu karakterlerin oktal kodları 000 ila 037 ve 177'dir (DEL).
12.	Boş karakterler	Boşluk ve sekmelerden oluşan kümedir.
13.	Alfabetik karakterler	Küçük harfler ve büyük harflerden oluşan kümedir.

Fonksiyonlar:

Kütüphanede karakter işlemleri için tanımlanmış 14 fonksiyon bulunmaktadır. Bunlar alfabetik olarak şöyledir

isalnum(int c)	isdigit(int c)	isprint(int c)	isupper(int c)	tolower(int c)
isalpha(int c)	isgraph(int c)	ispunct(int c)	isxdigit(int c)	toupper(int c)
iscntrl(int c)	islower(int c)	isspace(int c)	isblank(int c)	

Detaylı bilgi için: https://www.tutorialspoint.com/c_standard_library/ctype_h.htm

ctype.h kullanımına ait örnek kodlar

Karakter türü kontrolü

```c
#include <stdio.h>
#include <ctype.h>

int main() {
   char harf = 'B';

   if (isalpha(harf)) {
      printf("%c bir harftir.\n", harf);
   } else {
      printf("%c bir harf değildir.\n", harf);
   }

   return 0;
}
```

```
B bir harftir.
```

Karakter dönüştürme

```c
#include <stdio.h>
#include <ctype.h>

int main() {
   char harf = 'm';
   char bharf = 'E';

   printf("Büyük harf: %c\n", toupper(harf));

   printf("Küçük harf: %c\n", tolower(bharf));

   return 0;
}
```

```
Büyük harf: M
Küçük harf: e
```

Alfanümerik kontrol

```c
#include <stdio.h>
#include <ctype.h>

int main() {
   char harf = 'B';  // '&' için

   if (isalnum(harf)) {
      printf("%c bir alfanümerik karakterdir.\n", harf);
   } else {
      printf("%c bir alfanümerik karakter değildir.\n", harf);
   }

   return 0;
}
```

```
B bir alfanümerik karakterdir.
& bir alfanümerik karakter değildir.
```

Metin dizileri üzerinde işlem yapan fonksiyonları

string.h başlık kütüphanesi dizeler (karakter dizileri) ve belleği işlemek için kullanılan bir standart kütüphanedir. Bu kütüphane, dizelerle ilgili yaygın işlemleri gerçekleştirmek için birçok fonksiyon sağlar. Ayrıca bellek blokları arasında kopyalama, karşılaştırma ve temizleme gibi işlemleri gerçekleştirmek için de kullanılır.

Veri tipi: size_t (unsigned)

Simgesel sabit:

NULL metin dizisi sonlandırma karakteri, '\0'

Dizi düzenleme fonksiyonları

strlen(dizi)	Bir dizedeki karakter sayısını döndürür (null karakter dahil değildir).
strcpy(hedef, kaynak)	Kaynak dizenin içeriğini hedef dizeye kopyalar.
strncpy(hedef, kaynak, sayı)	Kaynak dizenin en fazla n karakterini hedefe kopyalar.
strcat(hedef, kaynak)	Kaynak dizeyi hedef dizeye ekler.
strncat(hedef, kaynak, sayı)	Kaynak dizenin en fazla n karakterini hedef dizeye ekler.
strcmp(dizi_1, dizi_2)	İki dizenin alfabetik sırasını karşılaştırır.
strncmp(dizi_1, dizi_2, sayı)	İki dizenin ilk n karakterini karşılaştırır.
strchr(dizi, harf)	Bir dizede belirtilen karakterin ilk geçtiği yeri bulur.
strrchr(dizi, harf)	Bir dizede belirtilen karakterin son geçtiği yeri bulur.
strstr(dizi, dizi_bul)	Bir dizede başka bir dizenin ilk geçtiği yeri bulur.
strtok(dizi, ayraç)	Bir dizgeyi belirli ayırıcılarla parçalara böler (tokenize eder).

Bellek işleme fonksiyonları

memcpy(hedef, kaynak, sayı)	Kaynak bellek bloğunu hedef bellek bloğuna kopyalar.
memmove(hedef, kaynak, sayı)	Kaynak ve hedef bloklar çakışsa bile güvenli şekilde veri kopyalar.
memcmp(ptr1, ptr2, sayı)	İki bellek bloğunu karşılaştırır.
memset(ptr, değer, sayı)	Bir bellek bloğunu belirtilen değerle doldurur.
memchr(ptr, değer, say)	Bellek bloğunda belirtilen değerin ilk geçtiği yeri bulur.

Detaylı bilgi için: https://www.tutorialspoint.com/c_standard_library/string_h.htm

string.h kullanımına ait örnek kodlar

Metin dizisini kopyalama

```c
#include <stdio.h>
#include <string.h>

int main() {
   char kaynak[] = "Bilgisayar";
   char hedef[20];

   strcpy(hedef, kaynak);

   printf("Kopyalanan metin: %s\n", hedef);

   return 0;
}
```

```
Bilgisayar
```

Metin dizisinin bir parçasını bulma

```c
#include <stdio.h>
#include <string.h>

int main() {
   char str[] = "Merhaba dünya!";
   char *substr = strstr(str, "dünya");

   if (substr != NULL) {
      printf("Alt dize bulundu: %s\n", substr);
   } else {
      printf("Alt dize bulunamadı.\n");
   }
   return 0;
}
```

```
dünya!
```

Belleği kopyalama

```c
#include <stdio.h>
#include <string.h>

int main() {
   char kaynak[10] = "123456";
   char hedef[20];

   memcpy(hedef, kaynak, 6);
   printf("Kopyalanan bellek: %s\n", hedef);

   return 0;
}
```

```
123456
```

Belleği doldurma

```c
#include <stdio.h>
#include <string.h>

int main() {
   char havuz[10];
   memset(havuz, '*', sizeof(havuz)-1);

   havuz[9] = '\0'; // sonlandırıcı ekle

   printf("Doldurulan bellek: %s\n", havuz);

   return 0;
}
```

```
*********
```

Genel amaçlı fonksiyonlar

stdlib.h başlık kütüphanesi genel amaçlı yardımcı işlevler sağlayan bir standart başlık dosyasıdır. Bu kütüphane, dinamik bellek yönetimi, sayısal dönüşümler, rastgele sayı üretimi ve çıkış işlemleri gibi birçok temel işlemi gerçekleştirmek için kullanılan fonksiyonlar, bazı veri tipler ve birkaç sabit içerir.

Veri tipleri: size_t (unsigned), wchar_t (unsigned), div_t (unsigned), ldiv_t (unsigned)

Simgesel sabit:

NULL	boş bir göstericiyi temsil eder.
EXIT_SUCCESS	başarılı program çıkışı için durum kodu (0).
EXIT_FAILURE	hatalı program çıkışı için durum kodu (1).
RAND_MAX	rand() fonksiyonu tarafından döndürülebilecek maksimum rastgele sayı değeri.
MB_CUR_MAX	MB_LEN_MAX değerinden büyük olmayan metin dizisindeki maksimum bayt sayısıdır.

Sayısal Dönüşüm Fonksiyonları	atoi()	atof()	atol()	strtod()	strtol()	strtoul()
Dinamik Bellek Fonksiyonları	calloc()	malloc()	realloc()	free()		
Rastgele Sayı Üretimi Fonksiyonları	rand()	srand()				
Program Kontrol Fonksiyonları	abort()	exit()	atexit()			
Matematiksel Fonksiyonlar	abs()	div()	labs()	ldiv()		
Metin Dizisi Fonksiyonları	getenv()	mblen()	mbstowcs()	mbtowc()	wcstombs()	wctomb()
Yardımcı Fonksiyonları	bsearch()	qsort()	system()			

Detaylı bilgi için: https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

stdlib.h kullanımına ait örnek kodlar

Sayısal Dönüşüm

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   char str[] = "1234";

   // Dizeyi tamsayıya dönüştür
   int num = atoi(str);

   printf("Dönüştürülen sayı: %d\n", num);

   return 0;
}
```

```
1234
```

Rastgele sayı üretimi

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    // Tohumu mevcut zamana göre ayarla
    srand(time(NULL));

    for (int i = 0; i < 5; i++) {
        printf("Rastgele sayı: %d\n", rand());
    }
   return 0;
}
```

```
Rastgele sayı: 17052
Rastgele sayı: 23999
Rastgele sayı: 11041
Rastgele sayı: 14473
Rastgele sayı: 31194
```

Diziyi Sıralama

```c
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}
int main() {
   int arr[] = {5, 2, 9, 1, 7};
   size_t size = sizeof(arr) / sizeof(arr[0]);
   qsort(arr, size, sizeof(int), compare);

   printf("Sıralı dizi: ");
   for (size_t i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   return 0;
}
```

```
Sıralı dizi: 1 2 5 7 9
```

Dinamik Bellek Yönetimi

C dilinde dinamik bellek yönetimi, bir programın çalışma zamanında ihtiyaç duyduğu kadar bellek ayırmasını ve artık kullanılmayan belleği serbest bırakmasını sağlayan bir sistemdir. Bu mekanizma, programın başlangıcında bellek boyutunun sabit olmadığı durumlarda veya büyük ve esnek veri yapılarına ihtiyaç duyulduğunda oldukça önemlidir. Örneğin, kullanıcıdan alınan verilere bağlı olarak değişen boyutlarda bir dizi oluşturmak veya programın çalışması sırasında yeni veri yapıları eklemek gerektiğinde dinamik bellek yönetimi devreye girer.

Dinamik bellek yönetime ait işlemler **stdlib.h** başlık dosyasında tanımlı fonksiyonlar kullanılarak gerçekleştirilir. Bu fonksiyonlar sayesinde bellek kullanımını optimize etmek, programın esnekliğini artırmak ve gereksiz bellek tüketimini önlemek mümkün olur. Dinamik bellek yönetimi, özellikle karmaşık ve uzun süreli çalışan uygulamalarda verimli bir şekilde kullanıldığında büyük bir avantaj sağlar.

Dinamik Bellek Yönetiminin Temel Kavramları

C dilinde bellek yönetimi, işletim sistemi tarafından sağlanan bir bellek havuzundan (heap) yapılır. Bu havuz, program çalışırken dinamik olarak genişletilebilir veya daraltılabilir. Bellek yönetiminde 3 temel kavram vardır ve bunlar aşağıdaki şekilde özetlenebilir.

1. **Bellek Tahsisi (Memory Allocation)**
   Bellek tahsisi, çalışma zamanında belirli bir miktarda belleğin program tarafından kullanılmak üzere ayrılmasıdır. Bellek tahsisi için kullanılan fonksiyonlar **malloc** ve **calloc** fonksiyonlarıdır.

2. **Belleği Yeniden Boyutlandırma (Reallocation)**
   Tahsis edilen bir belleğin boyutunu arttırılması veya azaltılması işlemidir. Bunun için **realloc** fonksiyonu kullanılır.

3. **Belleğin Serbest Bırakılması (Memory Deallocation)**
   Kullanılmayan bir bellek bloğunun serbest bırakması işlemidir, böylece sistemde başka işlemler için serbest bırakılan bellek kullanılabilir duruma gelir. Bu işlem için **free** fonksiyonu kullanılır.

Dinamik bellek yönetiminde önemli bazı noktalar

C dilinde dinamik bellek yönetimi oldukça güçlü bir araçtır, ancak dikkat edilmesi gereken bazı önemli noktalar vardır. Doğru kullanılmadığında, bellek sızıntıları, belirsiz davranışlar veya program çökmelerine neden olabilir. İşte bu konuda dikkat edilmesi gereken temel noktalar şu şekilde verilebilir.

1. Bellek tahsis edildiğinden emin olma: Bellek tahsis işlemleri her zaman başarılı olmayabilir. Örneğin, yeterli bellek yoksa malloc, calloc, veya realloc NULL döndürür. Eğer bu dönüş değeri kontrol edilmeden bellek kullanılmaya çalışılırsa, program çökebilir.

2. Serbest bırakılan belleğe erişim: Bir bellek bloğu serbest bırakıldıktan sonra bu bloğa erişmeye çalışmak, tanımsız davranışa neden olur. Bu durum "dangling pointer" olarak adlandırılır.

3. Bellek sızıntısı (memory leak): Dinamik olarak tahsis edilen bir bellek bloğu free ile serbest bırakılmazsa, bu bellek bir daha kullanılamaz hale gelir. Uzun süreli programlarda bellek sızıntısı, sistem kaynaklarının tükenmesine neden olabilir.

4. Çift Serbest Bırakma (Double Free): Aynı bellek bloğunu birden fazla kez serbest bırakmak, tanımsız davranışlara yol açar ve genellikle programın çökmesine neden olur.

5. Bellek Boyutunu Yeniden Ayarlarken Dikkatli Olma: realloc başarısız olursa, NULL döner, ancak orijinal bellek bloğu korunur. Eğer dönüş değeri doğrudan orijinal göstericiye atanırsa, orijinal bloğa erişim kaybolur ve bu durum bellek sızıntısına yol açar.

6. Tahsis edilen belleğin doğru kullanımı: Tahsis edilen bellek doğru şekilde kullanılmazsa, tanımsız davranışa neden olabilir.

7. Belleği sıfırlama gerekliği: malloc fonksiyonu belleği sıfırlamaz; tahsis edilen bellek, daha önce bu alanı kullanan programlardan kalan rastgele veriler içerebilir.

8. Büyük bellek tahsislerinde dikkat: Çok büyük bellek tahsisleri (örneğin, birden fazla gigabayt) başarısız olabilir veya sistemi yavaşlatabilir.

9. Bellek alanını serbest bırakma sırası: Birden fazla bellek bloğu tahsis ettiyseniz, serbest bırakma sırasına dikkat edin. Bir bloğa diğer bir bloğun adresi yazılmışsa, yanlış sırayla serbest bırakmak, belirsiz davranışlara neden olabilir.

10. Bellek sınırları dışına erişim (buffer overflow): Bellek sınırlarının dışına veri yazmak veya okumak, belleğin yanlışlıkla değiştirilmesine neden olabilir ve bu durum güvenlik açıklarına yol açabilir.

Bellek tahsisi – malloc fonksiyonu

C dilinde **malloc (memory allocation)**, dinamik bellek tahsisi için kullanılan bir fonksiyondur. **malloc**, programın çalışma zamanında ihtiyaç duyduğu bellek miktarını ayırır ve bu bellek bloğunun başlangıç adresini döndürür. Bu, özellikle veri yapılarının boyutunun önceden bilinmediği veya esnek bir bellek yönetiminin gerektiği durumlarda kullanışlıdır. Başarısızlık durumunda NULL döndürülür. Fonksiyonun yazım düzeni şöyledir.

```
void *malloc(size_t alan);
```

Dinamik dizi tahsisi

```
int main() {
   // 5 tamsayı için bellek ayırır
   int *arr = (int *)malloc(5 * sizeof(int));
   if (arr != NULL) {
      for (int i = 0; i < 5; i++) {
         arr[i] = i + 1; // Değer atama
      }
   }
   // Tahsis edilen belleği serbest bırakır
   free(arr);
   return 0;
}
```

Yapılar için bellek tahsisi

```
int main() {
   typedef struct {
      int id;
      char name[20];
   } Ogrenci; // Öğrenci veri tipi
   // Bir öğrenci için bellek ayırır
    Ogrenci *ogr = (Ogrenci *)malloc(sizeof(Ogrenci));
   if (ogr != NULL) {
      ogr[0].id = 101;
      strcpy(ogr[0].name, "Ahmet");
   }
   free(ogr); // Tahsis edilen belleği serbest
    return 0;
}
```

Tahsis edilen bellek, genellikle **yığın (heap) alanından** tahsis edilir ve rastgele veriler içerebilir. Geri dönen değer bir **void** türüdür, bu nedenle dönen adres, kullanılacak veri türüne uygun şekilde dönüştürülmesi gereklidir.

Bellek tahsisi – calloc fonksiyonu

C dilinde **calloc (contiguous allocation)**, dinamik bellek tahsisi için kullanılan bir fonksiyondur. **calloc**, bir dizi eleman için sürekli bellek alanı tahsis eder ve tahsis edilen belleği sıfırlarla doldurur. Bu, özellikle sıfırlanmış bellek gerektiğinde avantajlıdır. Başarısızlık durumunda NULL döndürülür. Fonksiyonun yazım düzeni şöyledir.

```
void *calloc(size_t sayı, size_t tür);
```

Dinamik dizi tahsisi

```
int main() {
    // 5 tamsayı için sıfırlanmış bellek ayırır
    int *arr = (int *)calloc(5, sizeof(int));
    if (arr != NULL) {
    for (int i = 0; i < 5; i++) {
        // Tüm elemanlar başlangıçta sıfırdır
        printf("%d ", arr[i]);
        }
    }
    // Tahsis edilen belleği serbest bırakır
    free(arr);
    return 0;
}
```

Yapılar için bellek tahsisi

```
int main() {
    typedef struct {
    int id;
    char name[20];
    } Ogrenci; // Öğrenci veri tipi
    // 3 öğrenci için bellek ayırır
     Ogrenci *ogr = (Ogrenci *)calloc(3, sizeof(Ogrenci));
    if (ogr != NULL) {
        ogr[0].id = 101;
    strcpy(ogr[0].name, "Ali");
    }
    free(ogr); // Tahsis edilen belleği serbest bırakır
    return 0;
}
```

Belleği otomatik olarak sıfırlar, bu nedenle ek sıfırlama işlemi gerekmez. Dizi oluşturmak veya yapı türlerini sıfırlı bir şekilde başlatmak için uygundur. Ancak **malloc** ile karşılaştırıldığında, sıfırlama işlemi nedeniyle biraz daha yavaş olabilir. Yanlış parametrelerle kullanıldığında bellek taşması riski taşımaktadır.

Belleği yeniden boyutlandırma – realloc

C dilinde **realloc (reallocation)**, daha önce **malloc**, **calloc** veya **realloc** ile tahsis edilmiş bir bellek bloğunu yeniden boyutlandırmak için kullanılır. Bu, mevcut bellek bloğunu genişletmek veya küçültmek gerektiğinde kullanışlıdır. Eğer yeniden boyutlandırma başarısız olursa NULL döner. Fonksiyonun yazım düzeni şöyledir.

```
void *realloc(void gösterge, size_t alan);
```

Dinamik dizi boyutunu büyütme

```
int main() {
   // ilk tahsis
   int *arr = (int *)malloc(5*sizeof(int));

   if (arr == NULL) {
      printf("Bellek tahsisi başarısız!\n");
      return EXIT_FAILURE;
   }
   // Belleği genişletme
   arr = (int *)realloc(arr, 10*sizeof(int));
   if (arr == NULL) {
      printf("Bellek genişletme başarısız!\n");
      return EXIT_FAILURE;
   }
   // Belleği serbest bırakma
   free(arr);
   return 0;
}
```

Yapılar için boyut büyütme

```
int main() {
   typedef struct {
      int id;
      char name[20];
   } Ogrenci;
   // ilk tahsis
   Ogrenci *ogr = (Ogrenci *)calloc(2, sizeof(Ogrenci));
   if (ogr == NULL) {
      printf("Bellek tahsisi başarısız!\n");
      return EXIT_FAILURE;
   }
   // Yeni öğrenci için belleği yeniden boyutlandırma
   ogr = (Ogrenci *)realloc(ogr, 4*sizeof(Ogrenci));
   if (ogr == NULL) {
      printf("Bellek genişletme başarısız!\n");
      return EXIT_FAILURE;
   }
   free(ogr); // Belleği serbest bırakma
   return 0;
}
```

realloc ile dinamik olarak tahsis edilen belleğin boyutunu değiştirmek kolaydır ve mevcut veriyi otomatik olarak yeni bellek alanına kopyalar. Ancak veri kopyalandığında performans düşebilir ve hem önceki bellek alanı ve yeni bellek alanı aynı aktif olduğundan bellek kullanımı artacaktır.

realloc örnek kod - Dinamik dizi genişletme

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n = 5;
   int *arr = (int *)malloc(n * sizeof(int));

   if (arr == NULL) {
      printf("Bellek tahsisi başarısız!\n");
      return EXIT_FAILURE;
   }

   // Başlangıçta diziyi doldurma
   for (int i = 0; i < n; i++) {
      arr[i] = i + 1;
   }

   // Daha fazla eleman için belleği yeniden boyutlandırma
   int new_n = 10;
   int *new_arr = (int *)realloc(arr, new_n * sizeof(int));
   if (new_arr == NULL) {
         printf("Bellek genişletme başarısız!\n");
         free(arr);
         return EXIT_FAILURE;
    }

    arr = new_arr;

    // Yeni elemanları doldurma
    for (int i = n; i < new_n; i++) {
       arr[i] = i + 1;
    }

    // Diziyi yazdırma
    for (int i = 0; i < new_n; i++) {
      printf("%d ", arr[i]);
    }

    // Belleği serbest bırakma
    free(ogr);
    return EXIT_SUCCESS;
}
```

Belleğin serbest bırakılması – free

C dilinde **free** fonksiyonu, dinamik bellek yönetiminde tahsis edilmiş bellek alanını serbest bırakmak için kullanılır. Bu, **malloc**, **calloc**, veya **realloc** ile tahsis edilen belleğin kullanımı tamamlandıktan sonra belleğin geri kazandırılmasını sağlar. Fonksiyonun her hangi bir geri dönüş değeri yoktur. Fonksiyonun yazım düzeni şöyledir.

```
void free(void gösterge);
```

Dinamik belleği serbest bırakma

```
int *arr = (int *)malloc(5*sizeof(int));

if (arr != NULL) {
   // Bellek kullan
   free(arr); // Belleği serbest bırak
}
```

calloc veya realloc ile kullanım

```
int *arr = (int *)calloc(5, sizeof(int));

if (arr != NULL) {
   int *arr = (int *)calloc(5, sizeof(int));
   free(arr); //yeni bellek alanını serbest bırak
}
```

NULL kullanımı

```
int *ptr = NULL;

free(ptr);// Hiçbir işlem yapmaz, güvenlidir
```

Dikkat Edilmesi Gerekenler

1. **Belleği tekrar serbest bırakmamak:** Aynı bellek alanını birden fazla kez serbest bırakmak tanımsız davranışa yol açar.
2. **Serbest bırakılan göstericiye erişim:** free sonrası bellek alanına erişim tehlikeli ve yanlıştır.
3. **NULL atama:** Belleği serbest bıraktıktan sonra, göstericiye NULL atamak iyi bir pratiktir. Bu, kazara erişimi engeller.
4. **Statik veya otomatik bellek kullanımı:** free, yalnızca dinamik olarak tahsis edilen bellek alanları için kullanılır. Statik veya otomatik olarak tahsis edilen belleği serbest bırakmaya çalışmak hatalıdır.