به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

ورودی و خروجی

ورودی و خروجی استاندارد

- توابع مربوط به ورودی و خروجی که برای دریافت ورودی از ورودی استاندارد و فایل‌ها و درج خروجی برروی خروجی استاندارد و فایل‌ها طراحی شده‌اند، جزئی از زبان سی نیستند بلکه در کتابخانه‌ها پیاده‌سازی شده‌اند.
- یک جریان متنی [1] یا استریم متنی دنباله‌ای است از چند خط به طوری که هر خط با کاراکتر خط جدید پایان می‌یابد.
- ساده‌ترین سازوکار دریافت ورودی توسط یک برنامه از محیط، دریافت یک کاراکتر از ورودی استاندارد است که از طریق صفحه کلید دریافت می‌شود. تابع پیاده‌سازی در کتابخانهٔ استاندارد بدین منظور `int getchar(void)` است.
- تابع `getchar` یک کاراکتر از ورودی استاندارد دریافت کرده و بازمی‌گرداند. در صورت دریافت کاراکتر پایان فایل مقدار `EOF` توسط تابع بازگردانده می‌شود. این تابع در کتابخانه `stdio.h` تعریف شده است.

[1] text stream

ورودی و خروجی استاندارد

- در یک برنامهٔ سی می‌توان یک فایل را جایگزین ورودی استاندارد کرد، بدین معنی که برنامه به جای دریافت برنامه از ورودی استاندارد مقادیر ورودی را از فایل دریافت کند. این کار توسط علامت > انجام می‌شود.
- برای مثال فرض کنید برنامهٔ `prog` توسط تابع `getchar` مقادیری را از ورودی استاندارد دریافت می‌کند. در این‌صورت می‌توان توسط اجرای `prog < infile` فایل ورودی `infile` را جایگزین ورودی استاندارد کرد. توجه کنید برای این‌کار هیچ نیازی به تغییر برنامه `prog` نیست و همچنین `prog` نیازی نیست فایل ورودی را در لیست ورودی‌های خود توسط `argv` دریافت کند.
- همچنین در سیستم عامل‌های پایهٔ یونیکس می‌توان خروجی یک برنامه را به عنوان ورودی به برنامهٔ دیگر ارسال کرد. برای مثال اجرای `prog | otherprog` خروجی برنامهٔ `otherprog` را به عنوان ورودی برنامهٔ `prog` استفاده می‌کند.

ورودی و خروجی استاندارد

- تابع `int putchar (int)` برای ارسال یک خروجی از برنامهٔ سی استفاده می‌شود. با فراخوانی تابع `putchar (c)` کاراکتر `c` برروی خروجی استاندارد قرار می‌گیرد که به صورت پیش فرض صفحه نمایش است. تابع مقدار نوشته شده را باز می‌گرداند و در صورت بروز خط مقدار `EOF` را بازمی‌گرداند. خروجی استاندارد را می‌توان توسط علامت `>` برروی فایل ارسال کرد.
- برای مثال اگر برنامهٔ `prog` از تابع `putchar` استفاده کند، می‌توان توسط اجرای `prog > outfile` خروجی برنامه را به جای نمایش بر روی صفحه، برروی فایل ذخیره کرد. همچنین اجرای `prog | anotherprog` خروجی `prog` را به عنوان ورودی به `anotherprog` ارسال می‌کند.
- خروجی تابع `printf` نیز بر روی خروجی استاندارد چاپ می‌شود.
- همهٔ این توابع در کتابخانهٔ `stdio` تعریف شده‌اند. این کتابخانه را می‌توان توسط `#include ⟨ stdio.h ⟩` به برنامه ضمیمه کرد. در سیستم‌عامل‌های پایهٔ یونیکس فایل `stdio.h` به طور پیش‌فرض در آدرس `/usr/include` قرار گرفته است.

ورودی و خروجی استاندارد

- برنامهٔ زیر رشته‌ای را از ورودی دریافت می‌کند و پس از تبدیل حروف بزرگ به حروف کوچک، نتیجه را برروی خروجی استاندارد چاپ می‌کند.

```
۱ #include <stdio.h>
۲ #include <ctype.h>
۳ int main ()     /* lower: convert input to lower case */
۴ {
۵   int c while ((c = getchar ()) != EOF)
۶     putchar (tolower (c));
۷   return 0;
۸ }
```

- تابع `tolower` در کتابخانهٔ `ctype.h` تعریف شده است که یک حرف بزرگ را به حرف کوچک تبدیل می‌کند.

خروجی قالب‌بندی شده

- تابع `printf` تعدادی از مقادیر را به صورت رشته قالب‌بندی کرده و برروی خروجی استاندارد چاپ می‌کند.
- تابع `int printf (char *format, arg1, arg2, ...)` رشته‌ای که در پارامتر اول دریافت می‌کند را با جایگزین کردن زیر رشته‌های آن با مقادیر پارامترهای دوم و سوم و ... در خروجی استاندارد چاپ می‌کند. تابع تعداد کاراکترهای چاپ شده را بازمی‌گرداند.
- رشته دریافت شده در پارامتر اول رشته قالب‌بندی نامیده می‌شود. این رشته شامل تعدادی زیر رشته است که به طور معمول برروی خروجی استاندارد چاپ می‌شوند و همچنین شامل تعدادی عملگر تبدیل است که یک نوع را به رشته به صورتی قالبی معین تبدیل می‌کنند. هر عملگر تبدیل با علامت % آغاز می‌شود.

خروجی قالب‌بندی شده

- عملگر تبدیل دارای چند بخش است که در اینجا به بخش‌های آن اشاره می‌کنیم.
- علامت منفی در ابتدا باعث می‌شود مقدار از سمت چپ تراز شود. در صورتی که علامت منفی وجود نداشته باشد مقدار رشتهٔ تبدیل شده از سمت راست تراز می‌شود.
- عددی که در ابتدا عملگر نوع نوشته می‌شود، عرض رشتهٔ چاپ شده را تعیین می‌کند. به عبارت دیگر رشتهٔ تبدیل شده در یک فضا با عرض تعیین شده چاپ می‌شود.
- پس از عرض فضا، توسط علامت نقطه و یک عدد صحیح، دقت چاپ یک عدد را مشخص کرد. به عبارت دیگر برای اعداد اعشاری می‌توان تعداد ارقام مورد نیاز برای چاپ را پس از علامت نقطه تعیین کرد. برای اعداد صحیح حداقل تعداد ارقام و برای رشته‌ها حداکثر تعداد کاراکترها را می‌توان مشخص کرد.
- سپس برای متغیرهای `short` از حرف `h` و برای متغیرهای نوع `long` از `l` استفاده می‌کنیم.

خروجی قالب‌بندی شده

- برای مثال با فراخوانی `printf ("%*s", max, s)` حداکثر تعداد `max` کاراکتر از رشتهٔ `s` بر خروجی استاندارد چاپ می‌شود.
- مثال‌های زیر روش استفاده از عملگر تبدیل نوع را برای رشته‌ها نشان می‌دهد.

```
:%s:          :hello, world:
:%10s:        :hello, world:
:%.10s:       :hello, wor:
:%-10s:       :hello, world:
:%.15s:       :hello, world:
:%-15s:       :hello, world   :
:%15.10s:     :     hello, wor:
:%-15.10s:    :hello, wor     :
```

خروجی قالب‌بندی شده

- تابع `sprintf` شبیه `printf` عمل می‌کند با این تفاوت که خروجی قالب بندی شده را به جای چاپ برروی خروجی استاندارد در یک رشته کپی می‌کند.

```
int sprintf (char *string, char *format, arg1, arg2, ...)
```

لیست آرگومان‌ها با طول متغیر

- تعداد پارامترهای تابع `printf` متغیر است. در زبان سی می‌توان توابع با تعداد پارامترهای متغیر تعریف کرد.
- در اینجا می‌خواهیم تابعی شبیه به `printf` با تعداد پارامترهای متغیر پیاده‌سازی کنیم.
- در کتابخانهٔ `stdarg` تعدادی ماکرو برای ایجاد امکان تعداد پارامترهای متغیر تعریف شده است.
- نوع `va-list` یک لیست شامل تعدادی متغیر تعریف می‌کند. توسط ماکروی `va-start` می‌توان به ابتدای لیست متغیرها اشاره کرد. با هر بار فراخوانی `va-arg` در لیست متغیرها، یک متغیر به جلو حرکت می‌کنیم. در پایان `va-end` عملیات پایانی و آزادسازی حافظه‌های غیر مورد نیاز را انجام می‌دهد.

لیست آرگومان‌ها با طول متغیر

- تابع حداقلی چاپ به نام minprintf به صورت زیر پیاده‌سازی می‌شود.

```
۱  #include <stdarg.h>
۲  /* minprintf: minimal printf with variable argument list */
۳  void
۴  minprintf (char *fmt, ...)
۵  {
۶    va_list ap;         /* points to each unnamed arg in turn */
۷    char *p, *sval;
۸    int ival;
۹    double dval;
۱۰   va_start (ap, fmt);      /* make ap point to 1st unnamed arg */
```

لیست آرگومان‌ها با طول متغیر

```
11    for (p = fmt; *p; p++)
12      {
13           if (*p != '%')
14                {
15                     putchar (*p);
16                     continue;
17                }
18          switch (*++p)
19      {
20          case 'd':
21              ival = va_arg (ap, int);
22              printf ("%d", ival);
23              break;
```

لیست آرگومان‌ها با طول متغیر

```
۲۴      case 'f':
۲۵          dval = va_arg (ap, double);
۲۶          printf ("%f", dval);
۲۷          break;
۲۸      case 's':
۲۹           for (sval = va_arg (ap, char *); *sval; sval++)
۳۰                 putchar (*sval);
۳۱           break;
۳۲       default:
۳۳          putchar (*p);
۳۴          break;
۳۵        }
۳۶    }
۳۷    va_end (ap);      /* clean up when done */
۳۸  }
```

لیست آرگومان‌ها با طول متغیر

- تابع `scanf` شبیه تابع `printf` است با این تفاوت که به جای چاپ رشته قالب بندی شده برروی خروجی استاندارد، یک رشته قالب‌بندی شده را از ورودی استاندارد دریافت می‌کند.
- این تابع رشته‌ای را به صورتی که در آرگومان اول تابع تعیین شده دریافت می‌کند و عملگرهای تبدیل را با متغیرهای آرگومان‌های دوم و سوم و ... جایگزین می‌کند. به عبارت دیگر ورودی دریافت شده از کاربر را در متغیرهای آرگومان‌های دوم و سوم و ... ذخیره می‌کند.
- برای این که تابع `scanf` بتواند مقادیر متغیرهای ورودی خود را تغییر دهد، نیاز به فراخوانی با ارجاع است، بنابراین آرگومان‌های دوم به بعد از نوع اشاره‌گر هستند. برای ارسال یک متغیر به این تابع باید آدرس آن با استفاده از عملگر & ارسال شود.
- در صورتی که ورودی خوانده شده با آرگومان اول همخوانی نداشته باشد، تابع مقدار خطا بازمی‌گرداند.
- در صورت موفقیت، تابع تعداد کاراکترهای خوانده شده را بازمی‌گرداند و در صورت رسیدن به کاراکتر پایان فایل مقدار `EOF` را بازمی‌گرداند.

لیست آرگومان‌ها با طول متغیر

- تابع sscanf شبیه scanf عمل می‌کند با این تفاوت که به جای دریافت ورودی از ورودی استاندارد، آن را از یک رشته دریافت می‌کند.

```
int scanf (char *format, arg1, arg2, ...);
int sscanf (char *string, char *format, arg1, arg2, ...);
```

لیست آرگومان‌ها با طول متغیر

- برنامه زیر تعدادی عدد از ورودی دریافت کرده، مجموع آن‌ها را چاپ می‌کند.

```
۱ #include <stdio.h>
۲ int main ()        /* rudimentary calculator */
۳ {
۴   double sum, v;
۵   sum = 0;
۶   while (scanf ("%lf", &v) == 1)
۷     printf ("\t%.2f\n", sum += v);
۸   return 0;
۹ }
```

لیست آرگومان‌ها با طول متغیر

- فرض کنید می‌خواهیم یک تاریخ را از ورودی دریافت کنیم. این کار به صورت زیر انجام می‌شود.

```
int day, year;
char monthname[20];
scanf("%d %s %d", &day, monthname, &year);
```

لیست آرگومان‌ها با طول متغیر

- حال فرض کنید می‌خواهیم یک تاریخ را به صورت dd Month yy یا dd/mm/yy دریافت کنیم.
می‌توانیم ابتدا یک خط از ورودی را دریافت و در یک رشته ذخیره کنیم و سپس توسط sscanf مقادیر مورد نظر را از روی رشته خوانده شده بخوانیم.

```
۱ while (getline (line, sizeof (line)) > 0)
۲   {
۳     if (sscanf (line, "%d %s %d", &day, monthname, &year) == 3)
۴       printf ("valid: %s\n", line);   /* 25 Dec 1988 form */
۵     else if (sscanf (line, "%d/%d/%d", &month, &day, &year) == 3)
۶       printf ("valid: %s\n", line);   /* mm/dd/yy form */
۷     else
۸       printf ("invalid: %s\n", line);    /* invalid form */
۹   }
```

فایل‌ها

- در مثال‌های قبل برنامه‌ها مقادیر ورودی را از ورودی استاندارد (مانند کیبورد) می‌خواندند و مقادیر خروجی را بر روی خروجی استاندارد (مانند صفحه نمایش) چاپ می‌کردند.
- حال می‌خواهیم برای ورودی و خروجی از فایل‌ها استفاده کنیم. یک فایل یک واحد از اطلاعات است که برروی حافظه‌های بلندمدت ذخیره می‌شود و با یک نام و آدرس یکتا قابل دسترسی است.
- یکی از برنامه‌های جانبی که در سیستم عامل‌های بر پایهٔ یونیکس برای چاپ محتوای فایل‌ها برروی خروجی استاندارد استفاده می‌شود، برنامه `cat` است. با اجرای `cat x.c y.c` محتوای دو فایل `x.c` و `y.c` برروی صفحه نمایش نشان داده می‌شود.
- در کتابخانهٔ استاندارد دستوراتی برای کار با فایل‌ها مهیا شده‌اند.

فایل‌ها

- قبل از استفاده از یک فایل باید آدرس فایل از سیستم‌عامل گرفته شود و عملیات مقدماتی برای خواندن از فایل و نوشتن روی فایل انجام شود. برای این کار از دستور `fopen` برای بازکردن فایل استفاده می‌شود.
دستور `fopen` یک اشاره‌گر بازمی‌گرداند که اشاره‌گر فایل نامیده می‌شود. این اشاره‌گر به ساختمانی در حافظه اشاره می‌کند که اطلاعاتی در مورد فایل (مانند بافر آن در حافظه، مکان فعلی در هنگام خواندن و نوشتن فایل، وضعیت فایل و غیره) را نگهداری می‌کند.
- یک فایل به صورت زیر تعریف و باز می‌شود.

```
1 FILE * fp;
2 FILE *fopen (char *name, char *mode);
```

- در اینجا `fp` اشاره‌گری است به داده‌ای از نوع `FILE` و تابع `fopen` با دریافت نام فایل و حالت فایل، یک اشاره‌گر بازمی‌گرداند.

فایل‌ها

- یک فایل می‌تواند با استفاده از رشته "r" برای خواندن، با رشتهٔ "w" برای نوشتن و با رشته "a" برای افزودن به فایل باز شود.
- اگر یک فایل که در حالت نوشتن باز می‌شود وجود نداشته باشد، فایل ساخته می‌شود. وقتی یک فایل موجود برای نوشتن باز شود، محتوای قبلی از بین می‌رود. اگر بخواهیم از فایلی که وجود ندارد بخوانیم، با پیام خطا روبرو می‌شویم. در هنگام بروز خطا تابع fopen مقدار NULL بازمی‌گرداند.
- با استفاده از تابع int getc (FILE *fp) می‌توانیم یک کاراکتر از فایل بخوانیم. در صورتی که به پایان فایل برسیم، تابع مقدار EOF بازمی‌گرداند.
- با استفاده از تابع int putc (int c, FILE *fp) کاراکتر c برروی فایل نوشته می‌شود.
- توجه کنید که getc و putc و getchar و putchar ماکرو هستند نه تابع، بنابراین سربار فراخوانی توابع را ندارند.

فایل‌ها

- وقتی یک برنامه سی آغاز می‌شود، سه فایل توسط سیستم‌عامل باز می‌شوند که این فایل‌ها عبارتند از فایل‌های ورودی استاندارد، خروجی استاندارد و پیام خطا استاندارد. سه اشاره‌گر `stdin` ، `stdout` و `stderr` بازگردانده می‌شوند که این فایل‌ها در کتابخانه `stdio.h` تعریف شده‌اند. تابع `getchar()` معادل است با تابع `getc(stdin)` ، بنابراین ماکروهای زیر تعریف شده‌اند :

```
1 #define getchar()  get(stdin)
2 #define putchar(c)  putc((c), stdout)
```

- برای ورودی و خروجی قالب بندی شده در فایل توابع `fprintf` و `fscanf` تعریف شده‌اند که به عنوان پارامتر اول یک اشاره‌گر به فایل دریافت می‌کنند.

```
1 int fscanf (FILE *fp, char *format, ...)
2 int fprintf (FILE *fp, char *format, ...)
```

فایل‌ها

- حال می‌خواهیم برنامه‌ای شبیه به cat بنویسیم که فایل‌های دریافت شده را برروی خروجی استاندارد چاپ کند.

```
۱  #include <stdio.h>
۲  /* cat: concatenate files, version 1 */
۳  int main (int argc, char *argv[])
۴  {
۵    FILE *fp;
۶    void filecopy (FILE *, FILE *) if (argc == 1)
۷    /* no args; copy standard input */
۸      filecopy (stdin, stdout);
۹    else
۱۰     while (--argc > 0)
۱۱           if ((fp = fopen (*++argv, "r")) == NULL)
۱۲             {
۱۳                printf ("cat: can't open %s\n", *argv);
```

```
۱۵                return 1;
۱۶             }
۱۷     else
۱۸        {
۱۹           filecopy (fp, stdout);
۲۰           fclose (fp);
۲۱        }
۲۲     return 0;
۲۳  }
۲۴  /* filecopy: copy file ifp to file ofp */
۲۵  void
۲۶  filecopy (FILE * ifp, FILE * ofp)
۲۷  {
۲۸    int c;
۲۹    while ((c = getc (ifp)) != EOF)
۳۰      putc (c, ofp);
۳۱  }
```

فایل‌ها

- اشاره‌گرهای `stdin` و `stdout` از نوع `FILE` هستند.
- تابع `int fclose (FILE *fp)` برای خاتمه‌دادن به عملیات برروی فایل و بازیابی حافظه به کار می‌رود. این تابع عملیاتی برعکس عملیات مورد نیاز برای بازکردن فایل انجام می‌دهد و به عبارت دیگر فایل را می‌بندد. همچین تابع `fclose` هرآنچه در بافر نوشتن برروی فایل قرار دارد را در فایل ذخیره می‌کند.

فایل‌ها

- فرض کنید می‌خواهیم خروجی یک برنامه را برروی یک فایل ذخیره کنیم. اگر پیام‌های خطا برروی خروجی استاندارد چاپ شوند، خروجی و پیام‌های خطا همگی برروی فایل ذخیره می‌شوند. طراحی بهتر این است که پیام‌های خطا از خروجی برنامه جدا شوند تا بتوان خطاها را به طور جداگانه ذخیره و بررسی کرد.
- اشاره‌گر `stderr` یک خروجی استاندارد برای چاپ خطاهای برنامه است.

فایل‌ها

- در برنامهٔ زیر از خروجی استاندارد خطا برای چاپ خطاها استفاده شده است.

```
۱  #include <stdio.h>
۲  /* cat: concatenate files, version 2 */
۳  int main (int argc, char *argv[])
۴  {
۵    FILE *fp;
۶    void filecopy (FILE *, FILE *);
۷    char *prog = argv[0];   /* program name for errors */
۸    if (argc == 1)    /* no args; copy standard input */
۹        filecopy (stdin, stdout);
۱۰   else
۱۱       while (--argc > 0)
```

فایل‌ها

```
12        if ((fp = fopen (*++argv, "r")) == NULL)
13           {
14              fprintf (stderr, "%s: can't open %s\n", prog, *argv);
15              exit (1);
16           }
17        else
18          {
19              filecopy (fp, stdout);
20              fclose (fp);
21           }
22    if (ferror (stdout))
23      {
24           fprintf (stderr, "%s: error writing stdout\n", prog);
25           exit (2);
26       }
27    exit (0);
28  }
```

فایل‌ها

- در این برنامه پیام خطا برروی خروجی خطای استاندارد قرار می‌گیرد و همچنین در صورت بروز خطا با تابع `exit(1)` برنامه متوقف می‌شود.
- در تابع `main` دستور `return x` و `exit(x)` معادل یکدیگرند. مزیت تابع `exit` این است که از توابع دیگر غیر از `main` نیز می‌تواند فراخوانی شود که باعث توقف برنامه می‌شود.
- تابع `int ferror (FILE *fp)` درصورتی که خطایی در فایل رخ دهد مقدار غیر صفر بازمی‌گرداند.

فایل‌ها

- در کتابخانهٔ استاندارد تابعی برای خواندن یک خطا از یک فایل وجود دارد. این تابع یک اشاره‌گر به فایل دریافت می‌کند، و یک خط از ورودی را خوانده و اشاره‌گری به ابتدای آن بازمی‌گرداند.

```
char *fget (char *line, int maxline, FILE *fp)
```

- خط خوانده شده که طول آن حداکثر `maxline-1` است در متغیر `line` قرار می‌گیرد. در کاراکتر آخر مقدار `'\0'` قرار می‌گیرد.
- در صورت موفقیت این تابع اشاره‌گر `line` را بازمی‌گرداند و در غیراینصورت مقدار `NULL` بازگردانده می‌شود.
- برای نوشتن یک خط برروی یک فایل از تابع `fputs` استفاده می‌شود.

```
int fputs (char *line, FILE *fp)
```

- همچنین توابع `puts` و `gets` برای نوشتن رشته برروی خروجی استاندارد و خواندن رشته از ورودی استاندارد به‌کار می‌روند.

فایل‌ها

- توابع fgets و fputs به صورت زیر پیاده‌سازی می‌شوند.

```
۱  /* fgets: get at most n chars from iop */
۲  char *
۳  fgets (char *s, int n, FILE * iop)
۴  {
۵    register int c;
۶    register char *cs;
۷    cs = s;
۸    while (--n > 0 && (c = getc (iop)) != EOF)
۹      if ((*cs++ = c) == '\n')
۱۰       break;
۱۱   *cs = '\0';
۱۲   return (c == EOF && cs == s) ? NULL : s;
۱۳ }
```

فایل‌ها

```
۱۴ /* fputs: put string s on file iop */
۱۵ int
۱۶ fputs (char *s, FILE * iop)
۱۷ {
۱۸   int c;
۱۹   while (c = *s++)
۲۰     putc (c, iop);
۲۱   return ferror (iop) ? EOF : 0;
۲۲ }
```

فایل‌ها

- تابع getline برای دریافت یک خط از ورودی را می‌توانیم به صورت زیر با استفاده از fgets پیاده‌سازی کنیم.

```
1  /* getline: read a line, return length */
2  int
3  getline (char *line, int max)
4  {
5    if (fgets (line, max, stdin) == NULL)
6      return 0;
7    else
8      return strlen (line);
9  }
```

توابع متفرقه

- در کتابخانه استاندارد توابع زیادی وجود دارند که می‌توانند مورد استفاده قرار بگیرند.
- در کتابخانهٔ `string.h` توابع زیر تعریف شده‌اند.

```
۱  strcat(s,t); // concatenate t to end of s
۲  strncat(s,t,n); // concatenate n characters of t to end of s
۳  strcmp(s,t);
۴      // return negative, zero, or positive for s < t, s == t, s > t
۵  strncmp(s,t,n); // same as strcmp but only in first n characters
۶  strcpy(s,t); // copy t to s
۷  strncpy(s,t,n); // copy at most n characters of t to s
۸  strlen(s); // return length of s
۹  strchr(s,c); // return pointer to first c in s, or NULL if not present
۱۰ strrchr(s,c); // return pointer to last c in s, or NULL if not present
```

توابع متفرقه

- در کتابخانهٔ `ctype.h` توابع زیر تعریف شده‌اند.

```
۱ isalpha(c); // non-zero if c is alphabetic, 0 if not
۲ isupper(c); // non-zero if c is upper case, 0 if not
۳ islower(c); // non-zero if c is lower case, 0 if not
۴ isdigit(c); // non-zero if c is digit, 0 if not
۵ isalnum(c); // non-zero if isalpha(c) or isdigit(c), 0 if not
۶ isspace(c); // non-zero if c is blank, tab, newline, return, formfeed,
۷ toupper(c); // return c converted to upper case
۸ tolower(c); // return c converted to lower case
```

توابع متفرقه

- در کتابخانهٔ استاندارد همچنین تابع `ungetch` تعریف شده است که یک کاراکتر خوانده شده را به جریان ورودی بازمی‌گرداند. به طور مشابه تابع `int ungetc (int c, FILE *fp)` یک کاراکتر خوانده شده از فایل را به جریان ورودی بازمی‌گرداند.
- تابع `void *malloc (size_t n)` اشاره‌گری به `n` بایت تخصیص داده شده در حافظه بازمی‌گرداند و حافظه مورد نیاز را تخصیص می‌دهد.
- تابع `void *malloc (size_t n, size_t size)` اشاره‌گری به `n` شیء هرکدام با اندازه `size` در حافظه بازمی‌گرداند و حافظه مورد نیاز را تخصیص می‌دهد.

توابع متفرقه

- برای مثال برای یک آرایه n تایی از اعداد صحیح می‌نویسیم :

```
1 int *ip;
2 ip = (int *) calloc (n, sizeof (int));
```

- تابع free(p) فضای حافظه‌ای که توسط اشاره‌گر p تخصیص داده شده است را آزاد می‌کند. اگر فضای حافظه یک بار آزاد شود، بار دوم با خطا روبرو می‌شویم.
- بنابراین در برنامه زیر با خطا روبرو می‌شویم.

```
1 for ( p = head; p != NULL ; p = p -> next ) /*WRONG*/
2     free (p)
```

توابع متفرقه

- روش درست برای آزادسازی حافظه در این مثال به صورت زیر است:

```
for ( p = head ; p != NULL ; p = q ) {
        q = p -> next;
        free (p);
}
```

توابع متفرقه

- توابع ریاضی در کتابخانهٔ `math.h` پیاده‌سازی شده‌اند که می‌توانند مورد استفاده قرار بگیرند.

```
۱  sin(x); // sine of x, x in radians
۲  cos(x); // cosine of x, x in radians
۳  atan2(y,x); // arctangent of y/x, in radians
۴  exp(x); // exponential function e^x
۵  log(x); // natural (base e) logarithm of x (x>0)
۶  log10(x); // common (base 10) logarithm of x (x>0)
۷  pow(x,y); // x^y
۸  sqrt(x); // square root of x (x>0)
۹  fabs(x); // absolute value of x
۱۰ rand(); // generate a random number
۱۱ srand(x);  // sets the seed for rand()
```