مبانی کامپیوتر و برنامه نویسی به زبان C

فصل پنجم: تکمیل عبارات و فرمت ها، مقدمه ای بر اشاره گرها و توابع

۵-۱ مقدمه

یک شیء: یک قطعه ی قابل استفاده از حافظه متشکل از تعدادی بیت حاوی رشته ای از صفر و یک ها

وجود تعدادی مؤلفه برای هر شیء

دستورهای تعریف نوع برای تعیین نوع شیءها یا معنای بیت ها و بعضی مؤلفه های دیگر

محاسباتی بودن کلیه ی نوع ها در زبان C

گروه بندی نوع های صحیح و اعشاری، مقدار دهی اولیه

ثابت های نمادی و شمارشی و ماکروها

مطالب تکمیلی در ارتباط با عبارات و توابع ورودی و خروجی و عملگرها

اشاره گرها، نحوه ی تعریف و موارد ساده ی استفاده از آن ها

تعریف عملگرهای جدید در قالب توابع و روش استفاده از آن ها

۵-۲ تکمیل دستورهای تعریف نوع مقادیر

۵-۲-۱ طبقه بندی مقادیر صحیح

جدول ۱-۵ گروه های اعداد صحیح و محدوده ی قابل نمایش آن ها در Borland C Version ۳.۱

فایل limits.h در کنار کامپایلر C حاوی حدود نوع ها و گروه های مختلف مقادیر صحیح

جدول ۵-۱ : گروه بندی مقادیر صحیح همراه با اطلاعات مربوطه.

نحوه ی اعلام نوع	تعداد بایت	حداقل مقدار	حداکثر مقدار	توضیح
signed char char	یک	-۱۲۸	۱۲۷	یک بیت علامت و هفت بیت عدد
unsigned char	یک	صفر	۲۵۵	عدد مثبت هشت بیتی
signed short int signed short short short int signed int int	دو	-۳۲۷۶۸	۳۲۷۶۷	یک بیت علامت و ۱۵ بیت عدد
unsigned short int unsigned short unsigned int unsigned	دو	صفر	۶۵۵۳۵	عدد مثبت ۱۶ بیتی
signed long int signed long long long int	چهار	-۲۱۴۷۴۸۳۶۴۸	۲۱۴۷۴۸۳۶۴۷	یک بیت علامت و ۳۱ بیت عدد
unsigned long int unsigned long	چهار	صفر	۴۲۹۴۹۶۷۲۹۵	عدد مثبت ۳۲ بیتی

برنامه ی نمونه ی ۵-۱ حاوی مثال هایی از دستورهای تعریف نوع گروه های مختلف مقادیر صحیح

```
#include <stdio.h>
main()
{  signed short int ssi = ۳۲۷۶۷;      /* متغیرهای صحیح کوتاه با علامت */
   signed short ss = ۳۲۷۶۷;
   short s = ۳۲۷۶۷;   short int ti = ۳۲۷۶۷;
   signed int si = ۳۲۷۶۷;                  /* متغیرهای صحیح معمولی با علامت */
   unsigned short int usi = ۶۵۵۳۵;         /* متغیرهای صحیح کوتاه بدون علامت */
   unsigned short us = ۶۵۵۳۵;
   unsigned int ui = ۶۵۵۳۵;                /* متغیرهای صحیح معمولی بدون علامت */
   unsigned  un = ۰xFFFF;
   signed long int sli = ۰x۷FFFFFFFL;     /* متغیرهای صحیح بلند با علامت */
   signed long sl = ۰x۷FFFFFFFL;   long l = ۰x۷FFFFFFFL;
   long int li = ۰x۷FFFFFFFL;
   unsigned long int uli = ۰xFFFFFFFFUL; /* متغیرهای صحیح بلند بدون علامت */
   unsigned long ul = ۰xFFFFFFFFUL;
   printf("signed short int max:%d,min:%d\n", ssi, ssi + ۱);
   printf("    signed short max:%d,min:%d\n", ss, ss + ۱);
   printf("           short max:%d,min:%d\n", s, s + ۱);
   printf("      signed int max:%d,min:%d\n", si, si + ۱);
   printf("       short int max:%d,min:%d\n", ti, ti + ۱);
   printf("unsigned short int max:%u, min:%u\n", usi, usi + ۱);
   printf("    unsigned short max:%u, min:%u\n", us, us + ۱);
   printf("      unsigned int max:%u, min:%u\n", ui, ui + ۱);
   printf("          unsigned max:%u, min:%u\n", un, un + ۱);
   printf("signed long int max:%ld, min:%ld\n", sli, sli + ۱);
   printf("    signed long max:%ld, min:%ld\n", sl, sl + ۱);
   printf("       long int max:%ld, min:%ld\n", li, li + ۱);
   printf("           long max:%ld, min:%ld\n", l, l + ۱);
   printf("unsigned long int max:%lu, min:%lu\n", uli, uli + ۱);
   printf("    unsigned long max:%lu, min:%lu\n", ul, ul + ۱);
   scanf("٪۵hd ٪۸ld ٪۵lo ٪۶lX", &ss, &li, &sl, &l);
   printf("٪+۵d ٪-۲۰ld ٪۸lo ٪۶lX\n", ss, li, sl, l);
   scanf("٪۵i ٪۷li ٪۵hu ٪۷lu", &ti, &sli, &usi, &uli);
   printf("٪+۵i ٪۷lx % ۵u ٪-۷lu\n", ti, sli, usi, uli);
   return (۰);
}
-۱۱۲۳  ۴۴۳۳۲۲۱۱  ۵۴۳۲۱  ۱a۲bfc        داده‌های ورودی برنامه:
-۴۳۲۱  ۰x۱ab۲c  ۴۵۳۲۱  ۸۷۷۶۶۵۵

signed short int max:۳۲۷۶۷,min:-۳۲۷۶۸                    خروجی برنامه:
    signed short max:۳۲۷۶۷,min:-۳۲۷۶۸
           short max:۳۲۷۶۷,min:-۳۲۷۶۸
      signed int max:۳۲۷۶۷,min:-۳۲۷۶۸
       short int max:۳۲۷۶۷,min:-۳۲۷۶۸
unsigned short int max:۶۵۵۳۵, min:۰
    unsigned short max:۶۵۵۳۵, min:۰
      unsigned int max:۶۵۵۳۵, min:۰
          unsigned max:۶۵۵۳۵, min:۰
```

شکل ۵-۲: متن برنامهٔ ۵-۱، نمونه‌هایی از تعریف، تخصیص مقدار، حدود، خواندن و نوشتن مقادیر صحیح.

۵-۲-۲ طبقه بندی مقادیر اعشاری

اعشاری معمولی

اعداد اعشاری با دقت دوبل (double)

اعداد اعشاری با طول بلند و دقت دوبل (long double)

۱۰ بایت برای نوع بلند دوبل در کامپایلر Borland C Version ۳.۱

عدد اعشاری معمولی: یک بیت برای علامت، هشت بیت برای توان و علامت آن و ۲۳ بیت برای مانتیس

عدد دوبل: یک بیت علامت، ۱۲ بیت برای توان و علامت آن و ۵۱ بیت برای مانتیس

عدد بلند دوبل: یک بیت برای علامت، ۱۶ بیت برای توان و علامت آن و ۶۳ بیت برای مانتیس

اپسیلن: کوچکترین مقداری که اگر با یک جمع زده شود نتیجه بزرگتر از یک خواهد بود

جدول ۵-۲ گروه های اعشاری همراه با محدوده ی قابل نمایش آن ها در Borland C Version ۳.۱ **فایل** float.h **در کنار کامپایلر** C **حاوی حدود نوع ها و گروه های مختلف مقادیر اعشاری**

جدول ۵-۲ : گروه بندی مقادیر اعشاری همراه با اطلاعات مربوطه.

نحوه ی اعلام نوع	تعداد بایت	قدر مطلق حداقل و حداکثر مقدار	اپسیلن
float	۴	۱.۱۷۵۴۹۴۳۵E-۳۸ ۳.۴۰۲۸۲۳۴۷E+۳۸	۱.۱۹۲۰۹۲۹۰E-۰۷
double	۸	۲.۲۲۵۰۷۳۸۵۸۵۰۷۲۰۱۴E-۳۰۸ ۱.۷۹۷۶۹۳۱۳۴۸۶۲۳۱۶E+۳۰۸	۲.۲۲۰۴۴۶۰۴۹۲۵۰۳۱۳۱E-۱۶
long double	۱۰	۳.۳۶۲۱۰۳۱۴۳۱۱۲۰۹۳۵۱E-۴۹۳۲	۱.۰۸۴۲۰۲۱۷۲۴۸۵۵۰۴E-۱۹

۱.۱۸۹۷۳۱۴۹۵۳۵۷۲۳۱۷۶E+۴۹۳۲

برنامه ی نمونه ی ۵-۲ حاوی نمونه هایی از دستورهای تعریف نوع مقادیر اعشاری

```
#include <stdio.h>

main()
{  float fep = ۱.۱۹۲۰۹۲۹۰e-۰۷;                    /* متغیرهای اعشاری معمولی */
   float fmx = ۳.۴۰۲۸۲۳۴۷e۳۸;                      /* حداکثر توان ۳۸ */
   float fmn = ۱.۱۷۵۴۹۴۳۵E-۳۸F;                   /* حداقل توان -۳۸ */
   double dep = ۲.۲۲۰۴۴۶۰۴۹۲۵۰۳۱۳۱E-۱۶;          /* متغیرهای اعشاری دوبل */
   double dmx = ۱.۷۹۷۶۹۳۱۳۴۸۶۲۳۱۶e۳۰۷;           /* حداکثر توان ۳۰۷ */
   double dmn = ۲.۲۲۵۰۷۳۸۵۸۵۰۷۲۰۱۴E-۳۰۸;         /* حداقل توان -۳۰۸ */
   long double ldep = ۱.۰۸۴۲۰۲۱۷۲۴۸۵۵۰۴E-۱۹;     /* متغیرهای اعشاری بلند دوبل */
   long double ldfl = ۱۲۳۴۵۶۷۸۹۰.۱۲۳۴۵۶۷۸۹;
   long double ldmx = ۱.۱۸۹۷۳۱۴۹۵۳۵۷۲۳۱۷۶E۴۹۳۱; /* حداکثر توان ۴۹۳۱ */
   long double ldmn = ۳.۳۶۲۱۰۳۱۴۳۱۱۲۰۹۳۵۱E-۴۹۱۵; /* حداقل توان -۴۹۱۵ */

   printf("float max:%۱۴.۸e,\n       min:%۱۴.۸e,\n"
          "      epsilon:%۱۴.۸e\n", fmx, fmn, fep);
   printf("double max:%۲۲.۱۵e,\n       min:%۲۳.۱۶e,\n"
          "       epsilon:%۲۲.۱۶e\n", dmx, dmn, dep);
   printf("long double max:%۲۳.۱۵Le,\n            min:%۲۳.۱۵Le,\n"
          "             epsilon:%۲۱.۱۵Le\n", ldmx, ldmn, ldep);
   printf("long double example:%۲۰.۹Lf\n", ldfl);
   scanf("%e %۱۰g %۷G", &fmx, &fmn, &fep);
   printf("%+۱۲.۴e %-۲۰.۵E %۱۰.۵G\n", fmx, fmn, fep);
   scanf("%۱۲le %۱۶lg %۱۰lf", &dmx, &dmn, &dep);
   printf("%+۱۲.۴G %-۲۰.۵f %۱۰.۵g\n", dmx, dmn, dep);
   scanf("%۱۲Le %۱۶Lg %۱۰Lf", &ldmx, &ldmn, &ldfl);
   printf("%+۱۲.۴LG %-۲۰.۵Le %۱۰.۵Lg\n", ldmx, ldmn, ldfl);
   return (۰);
}
-۱۲۳.۴۵۶  -۹۷.۶۵۴e۲۹  ۳۴۵                    داده‌های ورودی:
۹۸۶  -۱۱۲۲۳۳۴۴۵۵e-۲۳۴  ۱۲۳۴۵۶۷۸۹۰
۹۸۷۶۵۴۳E۲۳۱۱  -۱۱۲۲۳۳.۴۴e-۱۲۳۴  ۰.۰۰۰۰۰۰۲۹

                                                  خروجی برنامه:
float max:۳.۴۰۲۸۲۳۴۷e+۳۸,
      min:۱.۱۷۵۴۹۴۳۵e-۳۸,
      epsilon:۱.۱۹۲۰۹۲۹۰e-۰۷
double max:۱.۷۹۷۶۹۳۱۳۴۸۶۲۳۱۶e+۳۰۷,
       min:۲.۲۲۵۰۷۳۸۵۸۵۰۷۲۰۱۴e-۳۰۸,
       epsilon:۲.۲۲۰۴۴۶۰۴۹۲۵۰۳۱۳۱e-۱۶
long double max:۱.۱۸۹۷۳۱۴۹۵۳۵۷۲۳۲e+۴۹۳۱,
```

۵-۲-۳ تعیین نوع مقادیر ثابت

اعداد صحیح

جدول ۵-۳ : نمونه هایی از مقادیر صحیح در عبارات همراه با نوع در نظر گرفته شده برای آن ها.

مقدار صحیح	نوع ذخیره	مقدار صحیح	نوع ذخیره
۵	int	۵ u	unsigned int
۳۲۷۶۷	int	۳۲۷۶۷ U	unsigned int
۳۲۷۶۸	long	۳۲۷۶۸ u	unsigned int
-۶۲۰۰۰	long	۶۲۰۰۰ U	unsigned int
۲۰۰۰۰۰۰۰۰	long	۲۰۰۰۰۰۰۰۰ u	unsigned long
۳۰۰۰۰۰۰۰۰	unsigned long	۰۱۷۷۷۷۷ U	unsigned int
۰۷	int	۰۷۱	long
۰۷۷۷۷	int	۵۱	long
۰۱۷۷۷۷۷	int	۰۱۲۳۴۵۶۷ Lu	unsigned long
۰۷۷۷۷۷۷	long	۰۷۷۷۷۷۷ u	unsigned long
۰ x۱۲ab	int	۰x۱۲ abu	unsigned int
۰ X ۷fffffff	long	.X FFFFFFFF	unsigned long

روش تعیین دقیق نوع در صورت وجود یا عدم وجود پسوند

عدد صحیح بدون پیشوند و پسوند در اولین نوع از انواع `int` , `long` **یا** `unsigned long` **که با توجه به کمیت آن، قابل نمایش باشد در نظر گرفته می شود.**

عدد صحیح مبنای هشت یا شانزده که بدون پسوند باشد در اولین نوع از انواع `int`، `unsigned int`، `long` **یا** `unsigned long` **که با توجه به کمیت آن، قابل نمایش باشد در نظر گرفته می شود.**

عدد صحیح با هر مبنایی که فقط پسوند u یا U داشته باشد در اولین نوع از انواع unsigned int یا unsigned long که با توجه به کمیت آن، قابل نمایش باشد در نظر گرفته می شود.

عدد صحیح با هر مبنایی که فقط پسوند l یا L داشته باشد در اولین نوع از انواع long یا unsigned long که با توجه به کمیت آن، قابل نمایش باشد در نظر گرفته می شود.

عدد صحیح با هر مبنایی که پسوند های l یا L و u یا U را با هم داشته باشد از نوع unsigned long در نظر گرفته می شود.

ترفیع نوع صحیح

در ۱ + 'a'، نوع char به نوع int تبدیل شده و نتیجه یک مقدار از نوع int است.

در عمل 'b' و 'a' مقادیر یک بایتی به int تبدیل شده و نتیجه نیز از نوع int خواهد بود.

اعداد اعشاری

جدول ۵-۴ : نمونه هایی از مقادیر اعشاری در عبارات همراه با نوع در نظر گرفته شده برای آن ها.

مقدار اعشاری	نوع ذخیره	مقدار اعشاری	نوع ذخیره
۵.	double	۵.L	long double
۵e.	double	۵e۳..	double
۵.f	float	۵e۳..L	long double
۵e.F	float	۵e۳...L	long double

۵-۲-۴ تبدیل نوع مقادیر – عملگر (cast)

دو متغیر total و units از نوع صحیح و متغیر avg از نوع اعشاری

عبارت avg = total / units خارج قسمت صحیح، با وجودی که avg از نوع اعشاری است.

نیاز به تبدیل حداقل یکی از عملوندهای تقسیم با استفاده از عملگر فوق به اعشاری

```
avg = (float) total / (float) units;
avg = (float) total / units;
avg = total / (float) units;
```

حاصل ضرب دو متغیر فوق را در متغیر `mult` **از نوع** `long`

عبارت `mult = total * units` **ممکن است جواب اشتباه در داخل** `mult` **بریزد.**

تولید نتیجه درست

```
mult = (long) total *  (long) units
mult = (long) total *  units
mult = total *  (long) units
```

هم تقدم با عملگرهای یکتایی مثل نقیض و منفی کردن

این عملگر هیچ تاثیری روی مقدار داخل متغیرها ندارد

نمی توان از آن برای عملوندهایی که آثار جانبی روی آن ها وجود دارد استفاده نمود.

چند نمونه

```
f = (float) ((int)d + ۱)      (long)  ('a' + ۱.۰)

          (double)  (x = ۷۷)          d = (double) i / ۳
```

استفاده اشتباه

```
(double) x = ۱۱۲۲۳۳۴۴۵۵۶۶۷۷
```

تبدیل ضمنی

یک عملوند صحیح و دیگری اعشاری، تبدیل صحیح به اعشاری و انجام عمل با نتیجه اعشاری

دو عملوند از یک نوع ولی با طول های متفاوت، هم طول کردن عملوند کوتاه و انجام عمل

شرح دقیق تبدیلات مزبور

اگر یکی از عملوندها بلند دوبل (long double) **باشد، قبل از انجام عمل، دیگری نیز به بلند دوبل تبدیل می شود.**

وگرنه در صورتی که یکی از عملوندها دوبل (double) **باشد، قبل از انجام عمل، دیگری نیز به دوبل تبدیل می شود.**

وگرنه در صورتی که یکی از عملوندها اعشاری معمولی (float) **باشد، قبل از انجام عمل، دیگری نیز به اعشاری معمولی تبدیل می شود.**

وگرنه ترفیع نوع صحیح به شرح بالا انجام می شود و در صورتی که یکی از عملوندها صحیح بلند بدون علامت (unsigned long int) **باشد، قبل از انجام عمل، دیگری نیز به صحیح بلند بدون علامت تبدیل می شود.**

وگرنه در صورتی که یکی از عملوندها صحیح بلند (long int) **و عملوند دیگر صحیح بدون علامت** (unsigned int) **باشد، بستگی به این دارد که نوع صحیح بلند بتواند مقدار صحیح بدون علامت را به طور کامل نشان دهد که در صورت فراهم بودن این امکان، قبل از انجام عمل، عملوند دوم به صحیح بلند تبدیل می گردد، در غیر این صورت هر دو عملوند به صحیح بلند بدون علامت تبدیل شده و عملگر روی آن ها اعمال می شود.**

وگرنه در صورتی که یکی از عملوندها صحیح بلند (long int) باشد، قبل از انجام عمل دیگری نیز به صحیح بلند تبدیل می شود.

وگرنه در صورتی که یکی از عملوندها صحیح بدون علامت (unsigned int) باشد، قبل از انجام عمل دیگری نیز به صحیح بدون علامت تبدیل می شود.

در غیر صورت های فوق هر دو عملوند از نوع صحیح (int) می باشند.

جدول ۵-۵ : نمونه هایی از عبارات با مقادیر غیر هم نوع همراه با نوع نتیجه ی آن ها با فرض داشتن تعاریف زیر.

```
char c;   short s;   int i;     long l;    unsigned u;
unsigned long ul;    float f;   double d;  long double ld;
```

عبارت	نوع نتیجه	عبارت	نوع نتیجه
c - s / i	int	c + ۵.۰	double
u * ۷ - i	unsigned	d + s	double
u * ۲.۰ - i	double	ld + c	long double
f * ۷ - i	float	u + ul	unsigned long
c + ۳	int	۲ * i / l	long
۷ * s * ul	unsigned long	u - l	به کامپیوتر بستگی دارد

تبدیل نوع در اثر عملگر تخصیص

مقدار سمت راست به نوع متغیر سمت چپ تبدیل شده و در آن ذخیره می گردد

همین مقدار ذخیره شده حاصل عملوند تخصیص مزبور نیز خواهد بود.

جزئیات این تبدیل ها در پایین خلاصه شده است.

در تبدیل مقدار با علامت به مقدار بدون علامت، اگر متغیر گیرنده ی مقدار (عملوند سمت چپ) از نظر تعداد بیت بلندتر باشد بیت های اضافی در سمت چپ با علامت مقدار محاسبه شده پر می شود وگرنه بیت های اضافی از سمت چپ مقدار حذف می گردد.

در تبدیل مقدار بدون علامت به مقدار بدون علامت، اگر متغیر گیرنده ی مقدار از نظر تعداد بیت بلندتر باشد بیت های اضافی در سمت چپ با صفر پر می شود وگرنه بیت های اضافی از سمت چپ مقدار حذف می گردد.

در تبدیل مقدار صحیح به مقدار با علامت، اگر متغیر گیرنده ی مقدار از نظر تعداد بیت گنجایش مقدار محاسبه شده را داشته باشد آن مقدار عیناً ذخیره می گردد وگرنه نتیجه ی کار در کامپیوترهای مختلف فرق می کند.

در تبدیل مقدار اعشاری به صحیح قسمت اعشار عدد حذف می گردد و نتیجه در متغیر ذخیره می گردد ولی اگر مقدار حاصل به دلیل بزرگی قابل نمایش نباشد نتیجه ی کار قابل پیش بینی نیست. حالت ویژه تبدیل عدد اعشاری منفی به صحیح بدون علامت است که تعریف نشده است.

در تبدیل مقدار صحیح به اعشاری اگر مقدار با توجه به کمیت آن، قابل ذخیره باشد به صورت نزدیکترین کمیت اعشاری بزرگتر یا کوچکتر از آن ذخیره می گردد ولی اگر قابل ذخیره نباشد نتیجه ی کار تعریف شده نیست.

در تبدیل یک عدد اعشاری کوتاه تر به یک عدد اعشاری بلندتر مقدار بدون هیچ گونه تغییری تبدیل می گردد. ترتیب اعداد اعشاری از کوتاه به بلند عبارت است از: اعشاری معمولی سپس اعشاری دوبل و نهایتاً بلند دوبل.

در تبدیل یک عدد اعشاری بلندتر به یک عدد اعشاری کوتاه تر، اگر مقدار با توجه به کمیت آن، قابل ذخیره باشد به صورت نزدیکترین کمیت اعشاری بزرگتر یا کوچکتر از آن ذخیره می گردد ولی اگر از نظر کمیت قابل ذخیره نباشد نتیجه ی کار تعریف شده نیست.

جدول ۵-۶ : نمونه هایی از عملگر تخصیص بین عملوندهایی از نوع های متفاوت همراه با نتیجه ی عمل با فرض داشتن تعاریف زیر برای متغیرهای استفاده شده در عبارات.

```
int  i = -۱;          long  l = -۱;          unsigned  u ;
unsigned long  ul ,     ul۱ = ۴۲۹۴۹۶۷۲۹۵ ;
float  f = ۹۹.۹۹۹۹  ,   fl = ۱e۱۵ ;
```

عبارت	نتیجه ی عمل تخصیص	توضیح
u = l	۶۵۵۳۵	شانزده بیت از سمت چپ مقدار l حذف شده بقیه ی بیت ها در u ذخیره شده است که به صورت ۱۱۱۱۱۱۱۱۱۱۱۱۱۱۱۱ در مبنای دو می باشد.
ul = i	۴۲۹۴۹۶۷۲۹۵	داخل i عددی متشکل از شانزده یک در مبنای دو است (۱- به صورت مکمل دو) شانزده بیت اضافه در سمت چپ با علامت این عدد یعنی یک پر می شود که حاصل عدد مبنای دویی متشکل از سی و دو تا یک است.
u = ul۱	۶۵۵۳۵	داخل ul۱ عددی متشکل از سی و دو یک در مبنای دو است که شانزده بیت از سمت چپ آن حذف شده بقیه در u ذخیره می شود که حاصل عدد مبنای دویی متشکل از شانزده تا یک است.
i = f	۹۹	قسمت اعشار عدد حذف می شود.
i = fl	؟؟؟؟؟	به دلیل بزرگ بودن مقدار سمت راست، این مقدار در عملوند سمت چپ قابل ذخیره نیست و نتیجه نامشخص خواهد بود.
f = ul۱	۴/۲۹۴۹۶۷E+۹	عدد صحیح به نزدیک ترین عدد اعشاری قابل نمایش تبدیل می شود.

۵-۳ سایر موارد مربوط به ثابت ها و متغیرها

۵-۳-۱ مقداردهی اولیه

```
int j = ۰, l, k, m = ۱۲۳;
char  esc = '\\';
long day = ۱۰۰۰ L * ۶۰ L * ۲۴ L * ۵۵ L;
```

تخصیص مقدار اولیه ی غیر قابل تغییر

```
const char prct = '%';
const int maliat = ۱۰۰۰, bimeh = ۲۵۰;
const long max long = ۲۱۴۷۴۸۳۶۴۷;
```

۵-۳-۲ ثابت های نمادی

قالب کلی:

define < نام ثابت نمادی > < مقدار یا متن جایگزین شونده >

عیب ثابت های نمادی نسبت به استفاده از const

مثال:

```
#define  SIZE         ۱۰۰
#define  MAXINT       ۳۲۷۶۷
#define  DBLEPSLN     ۲.۲۲۰۴۴۶۰۴۹۲۵۰۳۱۳۱E-۱۶
```

۵-۳-۳ ماکروها

مثال:

```
#define PRINT۳ (e۱, e۲, e۳) (printf("\n%c\t%c\t%d", (e۱), (e۲), (e۳)))
```

استفاده از ماکرو:

```
char char۱ = 'A', char۲ = 'B';
int num = ۹۹۹;
PRINT۳ (char۱, char۲ + ۵, num * ۲ + char۱);
```

دستور سوم در مرحله ی ترجمه

```
(printf("\n%c\t%c\t%d", (char۱), (char۲ + ۵), (num * ۲ + char۱)));
```

**علامت ادامه ی سطر قبلی: آخرین کاراکتر سطر قبلی علامت **

در تعریف ماکروی پارامتردار باید اسم ماکرو به پرانتز باز بعد از آن چسبیده باشد

برای جلوگیری از ابهام در تقدم اجرای عملگرها باید کل عبارت جایگزین شونده و هر یک از پارامترها داخل یک زوج پرانتز قرار گیرد.

تفاوت احضار ماکرو و احضار تابع

۵-۳-۴ ثابت های شمارشی

قالب کلی:

enum ‹اسم مجموعه› {‹لیست اسامی با یا بدون مقدار›} ;

مثال:

```
enum martial-status  {SINGLE, MARRIED, DIVORCED, WIDOWED};

enum months {FARVARDIN = ۱, ORDIBEHESHT, KHORDAD, TIR, MORDAD,
             SHAHRIVAR, MEHR, ABAN, AZAR, DAY, BAHMAN, ESFAND};
enum martial-status {SINGLE = ۲, MARRIED, DIVORCED = ۵, WIDOWED};

enum escapes {BELL    = '\a', BACKSPACE = '\b', TAB    = '\t',
              NEWLINE = '\n', VTAB      = '\v', RETURN = '\r'};
```

۵-۴ تکمیل مشخصه های فرمت

۵-۴-۱ سایر کدها در مشخصه ی فرمت چاپ

جدول ۵-۷ : کدهای قابل استفاده در مشخصه های تبدیل فرمت برای چاپ در خروجی.

کد	توضیح و تفسیر	نمونه ی خروجی	نوع مقدار متناظر
c	چاپ یک کاراکتر	p	char,short,int
d, i	چاپ مقدار صحیح به صورت عدد مبنای ده با علامت	-۹۸۷۲	char,short,int
ld, li	چاپ مقدار صحیح بلند به صورت عدد مبنای ده با علامت	۲۹۸۷۳۵۴۶۲۱	long int
o	چاپ مقدار صحیح به صورت عدد مبنای هشت بدون علامت	۳۵۷۶۰	char,short,int
lo	مشابه o برای مقدار صحیح بلند	۲۴۵۷۳۵۴۶۲۱	long int
x	چاپ مقدار صحیح به صورت عدد مبنای شانزده بدون علامت با استفاده از حروف a تا f برای ۱۰ تا ۱۵	۱۲af	char,short,int
x	مشابه x ولی با استفاده از حروف A تا F برای برای ۱۰ تا ۱۵.	۱۲AF	char,short,int
lx	برای مقدار صحیح بلند مشابهx	۱ab۲e۴۵f	long int

long int	۱AB۲E۴۵F	مشابه x برای مقدار صحیح بلند	lx
unsigned char, unsigned short, unsigned int	۵۴۷۱۲	چاپ مقدار صحیح به صورت عدد مبنای ده بدون علامت	u
unsigned long	۴۳۲۷۸۶۵۱۳۲	مشابه u برای مقدار صحیح بلند	lu
float, double	۲.۴۵۳۱۲۶e-۰۶	چاپ مقدار اعشاری معمولی یا دوبل با توان علمی و حرف e	e
float, double	۲.۴۵۳۱۲۶E-۰۶	مشابه e ولی توان علمی با حرف E	E
float, double	۲۴۵۳۱.۶۵	چاپ مقدار اعشاری معمولی یا دوبل بدون توان علمی	f
float, double	۴۵.۳۴۵۰۰۰	چاپ مقدار اعشاری معمولی یا دوبل با کوتاهترین فرمت از بین e یا f	g
float, double	۴.۵۶E-۱۴	مشابه g ولی با کوتاهترین فرمت از بین E یا f	G
long double	۴.۵۶۳۲۱e-۴۴۲	چاپ مقدار اعشاری بلند دوبل با توان علمی و حرف e	Le
long double	۴.۵۶۳۲۱E-۴۴۲	مشابه Le ولی توان علمی با حرف E	LE
long double	۴۵۶۳۱.۳۴۵۵۸۱۲۴	چاپ مقدار اعشاری بلند دوبل بدون توان علمی	Lf
long double	۴۵.۳۴۵e+۴۶۷	مشابهg برای مقادیر بلند دوبل	Lg
long double	۴۵.۳۴۵E+۴۶۷	مشابه G برای مقادیر بلند دوبل	LG
long double	%	چاپ کاراکتر %	%

نمونه هایی از کدهای فوق در برنامه های ۵-۱ و ۵-۲

اضافات قابل استفاده بلافاصله بعد از علامت %

- **کاراکتر – (منها) برای تنظیم چاپ فقره ی اطلاع مربوطه از چپ (left justify)**
- **کاراکتر + برای چاپ علامت عدد چه مثبت و چه منفی قبل از عدد**
- **کاراکتر # چاپ ۰، ۰x یا ۰X قبل از مقادیر به ترتیب با مشخصه ی فرمت مبناهای هشت و شانزده، چاپ ممیز اعشار (نقطه) برای مشخصه های فرمت e, E, f, g و G و چاپ صفرهای بی ارزش بعد از ممیز برای مشخصه های فرمت g و G.**
- **کاراکتر فاصله ی خالی برای قرار دادن یک فاصله ی خالی قبل از اعداد مثبت.**
- **کاراکتر ۰ برای پر کردن فاصله های اضافی سمت چپ با صفر به جای فاصله ی خالی.**
- **علامت × به جای طول میدان ماکزیمم و یا دقت**

مثال: متغیرهای صحیح m , n و p به ترتیب حاوی مقادیر ۵، ۶ و ۷

احضار ;(۲ * p, m+n, m, p, n, m, p, "%d,%*d,٪۵.*d,%*.*d"printf(

معادل ;(۲ * p, m+n, p, "%d,٪۶d,٪۵۰۵d,٪۶۰۵d"printf عمل خواهد نمود.

۵-۴-۲ سایر کدها در مشخصه ی فرمت خواندن

جدول ۵-۸: کدهای قابل استفاده در مشخصه های تبدیل فرمت برای خواندن از ورودی.

کد	توضیح و تفسیر	نمونه ی خروجی	نوع مقدار متناظر
c	خواندن یک کاراکتر	p	آدرس متغیر از نوعchar
d	خواندن و تبدیل مقدار صحیح مبنای ده به نوع int	-۱۹۸۷۲	آدرس متغیر از نوع int
hd	خواندن و تبدیل مقدار صحیح مبنای ده به نوعshort	۱۴۲۳۶	آدرس متغیر از نوع short
ld	خواندن و تبدیل مقدار صحیح مبنای ده به نوع long	۲۹۸۷۳۵۴۶۲۱	آدرس متغیر از نوعlong
o	خواندن و تبدیل مقدار صحیح مبنای هشت به نوع int	۳۵۷۶۰	آدرس متغیر از نوع int
ho	خواندن و تبدیل مقدار صحیح مبنای هشت به نوعshort	۳۵۷۶۰	آدرس متغیر از نوعshort
lo	خواندن و تبدیل مقدار صحیح مبنای هشت به نوع long	۲۴۵۷۳۵۴۶۲۱	آدرس متغیر از نوعlong
x, X	خواندن و تبدیل مقدار صحیح مبنای شانزده به نوع int	۱۲af, ۱۲AF	آدرس متغیر از نوعint
hx,hX	خواندن و تبدیل مقدار صحیح مبنای شانزده به نوعshort	۱۲af, ۱۲AF	آدرس متغیر از نوعshort
lx, lx	خواندن و تبدیل مقدار صحیح مبنای شانزده به نوع long	۱ab۲۵f, ۱AB۲۵F	آدرس متغیر از نوعlong

i	خواندن و تبدیل مقدار صحیح مبنای هشت (با پیشوند صفر)، ده یا شانزده (با پیشوند ۰x یا ۰X) به نوع int	۰۷۵۱ , ۱۲۶, ۰x۲af۹, ۰X۲AF۹	آدرس متغیر از نوع int
hi	مشابه i ولی برای تبدیل به نوع short	۱۲۶, ۰۷۵۱, ۰x۲af۹, ۰X۲AF۹	آدرس متغیر از نوعshort
li	مشابه i ولی برای تبدیل به نوع long	۱۲۳۴۶, ۰۷۲۳۶۵۱ ۰x۲af۹, ۰X۲AF۹	آدرس متغیر از نوعlong
u	خواندن و تبدیل مقدار صحیح به نوع usigned int	۴۲۳۴۶	آدرس متغیر از نوع unsigned int
hu	مشابه u ولی برای تبدیل به نوع usigned short	۴۲۳۴۶	آدرس متغیر از نوع usigned short
lu	مشابه u ولی برای تبدیل به نوع usigned long	۲۹۸۷۳۵۴۶۲۱	آدرس متغیر از نوع usigned long
e,E,f, g,G	خواندن و تبدیل عدد اعشاری با یا بدون توان علمی به نوع float	۲۴۱.۵, -۲۳.۵E-۲	آدرس متغیر از نوع float
le ,lE lf, lg , lG	خواندن و تبدیل عدد اعشاری با یا بدون توان علمی به نوع double	۲۴۱.۵۷۶۴۲۳۶, -۲۳, ۵۷۸۹۳۲E-۶۰۲	آدرس متغیر از نوعdouble
Le,LE, Lf,Lg, LG	خواندن و تبدیل عدد اعشاری با یا بدون توان علمی به نوع long double	۲۴۱.۵۷۶۴۲۳۶, -۲۳, ۵۷۸۹۳۲E-۶۰۲	آدرس متغیر از نوع usigned double

نکته ی مهم: تطبیق نوع متغیری که آدرس آن به عنوان آرگومان به تابع scanf ارسال می شود با نوع کد انتخاب شده در مشخصه ی فرمت متناظر آن

نمونه هایی از کاربرد این گونه مشخصه های فرمت در برنامه های ۵-۱ و ۵-۲

۵-۴-۳ خواندن و نوشتن با استفاده از فایل

تغییر مسیر ورودی و یا خروجی استاندارد از حالت عادی آن ها به یک فایل

برنامه ی قابل اجرا myprog

myprog < infile برای اجرای برنامه ی myprog و خواندن داده های ورودی از فایل infile

myprog > outfile باعث اجرای برنامه و نوشتن خروجی برنامه روی فایلی با نام outfile

myprog <imfile> outfile اجرای برنامه ی myprog با استفاده از فایل infile به عنوان ورودی استاندارد و outfile به عنوان خروجی استاندارد خواهد گردید.

۵-۵ مقدمه ای بر اشاره گرها

دسترسی به آدرس های حافظه از طریق اشاره گرها و عملگرهای مربوط به آن ها تنها راه انجام یک عمل در بعضی موارد استفاده از اشاره گرها است.

نوشتن برنامه های بهینه با استفاده ی درست از اشاره گرها

۵-۵-۱ نحوه ی تعریف اشاره گرها

اشاره گر متغیری است که در داخل آن می توان آدرس متغیر دیگری را ذخیره نمود.

به هر متغیر تعدادی بایت تخصیص داده می شود که آدرس اولین بایت آدرس متغیر است. هر اشاره گر ناظر بر یک نوع است (مؤید تعداد بایت شیء مورد اشاره)

هر اشاره گر بسته به کامپیوتر مورد استفاده دو یا چهار بایت را اشغال می کند. تعریف یک اشاره گر باعث ایجاد خانه ای از نوع معمولی که آن اشاره گر به آن اشاره کند نخواهد شد.

مثال:

```
int m = ۵, n = ۲, *pn, *pm;
```

```
float  x = ۱۰, y = ۲.۵, *px, *py;
```

۵-۵-۲ استفاده از اشاره گرها – عملگرهای آدرس و دستیابی غیرمستقیم

عملگر استخراج آدرس یعنی &

عملگر یکتایی دستیابی غیرمستقیم با علامت *

هم تقدم با سایر عملگرهای یکتایی از قبیل منفی کردن یا نقیض

مثال:

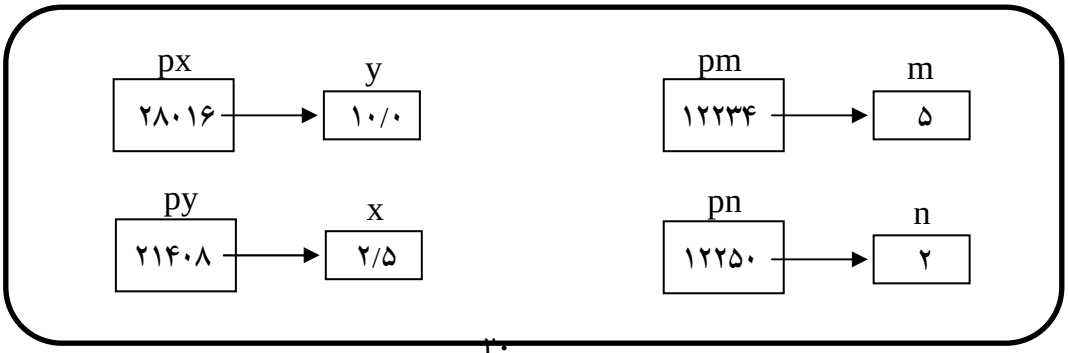
```
pm = &m;
px = &y;
pn = &n;
py = &x;
printf("%d, %f, %d, %f\n", m, y, *pm, *px);
scanf("%d%d%f%f", &m, pn, &x, px);
*pm = m + ۲ * *px;
y = *px + *py * *py;
```

ترکیب عملگر * با یک اشاره گر به عنوان یک شیء تغییر پذیر (lvalue)

به یک متغیر از نوع اشاره گر فقط می توان آدرسی از نوع خودش را نسبت داد

دستوری به شکل px = &m; توسط کامپایلر غلط گرفته می شود

ارتباط متغیرهای مورد استفاده در مثال بالا و محتویات آن ها در شکل ۵-۱.



شکل ۵-۱ : نمونه ای از ارتباط اشاره گرها و متغیرهای معمولی.

برنامه ی نمونه ی ۳-۵ حاوی نحوه ی تعریف و استفاده از اشاره گرها

```
#include <stdio.h>
#define START ۳                                   / * تعریف نقطه ی شروع اعداد اول * /

main()
{   int prm = START, dvd, *pdvd, n, *pn, rem;

    pn = &n;
    pdvd = &dvd;
    scanf ("%d", pn);                             / * استفاده از اشاره گر برای خواندن * /
    while (prm < =*pn)                            / * حلقه ی تکرار ساختن اعداد * /
    {   *pdvd = ۲;
        rem = ۱;
         while (rem > ۰ && dvd < prm)             / * حلقه ی تکرار بررسی اول بودن * /
           rem = prm %  (* pdvd) ++ ;
        if (rem > ۰)
          printf  ("%d\n", prm);
        prm + =  ۲;
    }
    return (۰);
}
```

شکل ۵-۴: متن برنامه ی ۵-۳ ، استخراج و چاپ اعداد اول از ۳ تا n.

۵-۶ سایر عملگرها

جدول ۵-۹ : عملگرهای زبان C همراه با تقدم و ترتیب اجرای آن ها (ادامه ی جدول در صفحه بعد).

ردیف	عملگر	نوع	توضیح	ترتیب اجرا	تقدم
۱	++	یکتایی	افزودن به صورت پسوند	از چپ به راست	اول(بالاترین)
۲	- -	یکتایی	کاستن به صورت پسوند	از چپ به راست	اول
۳	( )	یکتایی	داخل پرانتز احضار تابع	از چپ به راست	اول
۴	[ ]	دوتایی	زیرنویس آرایه ها	از چپ به راست	اول
۵	.	دوتایی	عضو رکورد	از چپ به راست	اول
۶	- >	دوتایی	دستیابی غیرمستقیم به عضو رکورد	از چپ به راست	اول

۷	-	یکتایی	منفی کردن	از راست به چپ	دوم
۸	+	یکتایی	مثبت کردن	از راست به چپ	دوم
۹	&	یکتایی	استخراج آدرس	از راست به چپ	دوم
۱۰	!	یکتایی	نقیض کردن	از راست به چپ	دوم
۱۱	++	یکتایی	افزودن به صورت پیشوند	از راست به چپ	دوم
۱۲	--	یکتایی	کاستن به صورت پیشوند	از راست به چپ	دوم
۱۳	~	یکتایی	مکمل کردن	از راست به چپ	دوم
۱۴	*	یکتایی	دستیابی غیرمستقیم	از راست به چپ	دوم
۱۵	(cast)	یکتایی	تبدیل نوع	از راست به چپ	دوم
۱۶	sizeof	یکتایی	تعیین اندازه	از راست به چپ	دوم
۱۷	*	دوتایی	ضرب	از چپ به راست	سوم
۱۸	/	دوتایی	تقسیم	از چپ به راست	سوم
۱۹	%	دوتایی	باقیمانده	از چپ به راست	سوم
۲۰	+	دوتایی	جمع	از چپ به راست	چهارم
۲۱	-	دوتایی	تفریق	از چپ به راست	چهارم
۲۲	>>	دوتایی	شیفت بیت ها به سمت راست	از چپ به راست	پنجم
۲۳	<<	دوتایی	شیفت بیت ها به سمت چپ	از چپ به راست	پنجم
۲۴	<	دوتایی	مقایسه ی کوچک تر است از	از چپ به راست	ششم
۲۵	<=	دوتایی	مقایسه ی کوچک تر است از یا مساوی است با	از چپ به راست	ششم
۲۶	>	دوتایی	مقایسه ی بزرگ تر است از	از چپ به راست	ششم
۲۷	>=	دوتایی	مقایسه ی بزرگ تر است از یا مساوی است با	از چپ به راست	ششم

ادامه ی جدول ۵-۹ : عملگرهای زبان C همراه با تقدم و ترتیب اجرای آن ها

ردیف	عملگر	نوع	توضیح	ترتیب اجرا	تقدم
۲۸	==	دوتایی	مقایسه ی تساوی	از چپ به راست	هفتم
۲۹	!=	دوتایی	مقایسه ی عدم تساوی	از چپ به راست	هفتم
۳۰	&	دوتایی	عطف بیت به بیت	از چپ به راست	هشتم
۳۱	^	دوتایی	فصل انحصاری بیت به بیت	از چپ به راست	نهم
۳۲	|	دوتایی	فصل بیت به بیت	از چپ به راست	دهم
۳۳	&&	دوتایی	عطف منطقی دو عبارت	از چپ به راست	یازده
۳۴	||	دوتایی	فصل منطقی دو عبارت	از چپ به راست	دوازده

۳۵	? :	سه تایی	عبارت شرطی	از راست به چپ	سیزده
۳۶	=	دوتایی	تخصیص	از راست به چپ	چهارده
۳۷	+ =	دوتایی	جمع زدن و تخصیص حاصل جمع	از راست به چپ	چهارده
۳۸	- =	دوتایی	تفریق کردن و تخصیص حاصل تفریق	از راست به چپ	چهارده
۳۹	* =	دوتایی	ضرب کردن و تخصیص حاصل ضرب	از راست به چپ	چهارده
۴۰	/ =	دوتایی	تقسیم کردن و تخصیص خارج قسمت	از راست به چپ	چهارده
۴۱	% =	دوتایی	محاسبه ی باقیمانده و تخصیص آن	از راست به چپ	چهارده
۴۲	| =	دوتایی	فصل بیت به بیت و تخصیص حاصل آن	از راست به چپ	چهارده
۴۳	^ =	دوتایی	فصل انحصاری بیت به بیت و تخصیص حاصل آن	از راست به چپ	چهارده
۴۴	& =	دوتایی	عطف بیت به بیت و تخصیص حاصل آن	از راست به چپ	چهارده
۴۵	>> =	دوتایی	شیفت بیت ها به سمت راست و تخصیص حاصل آن	از راست به چپ	چهارده
۴۶	<< =	دوتایی	شیفت بیت ها به سمت چپ و تخصیص حاصل آن	از راست به چپ	چهارده
۴۷	,	چندتایی	جدا سازی عبارت ها	از چپ به راست	پانزده

۵-۶-۱ عملگرهای یکتایی

افزودن یا کاستن

دو عملگر ++ و -- به ترتیب برای افزودن وکاستن، عجیب ترین عملگرها در زبان

به صورت پیشوند و پسوند و در هر دو حالت با اثر جانبی روی عملوند خود

به عنوان یک دستور به صورت پیشوند یا پسوند تفاوتی ندارد

مثال: با متغیرهای صحیح m و n به ترتیب حاوی مقادیر ۵ و ۱۰

در اثر اجرای هر کدام از دستورهای ; m++ یا ; ++m مقدار m برابر ۶ خواهد شد

در اثر اجرای هر کدام از دستورهای ; n-- یا ; --n مقدار n برابر ۹ خواهد گردید.

استفاده در خلال عبارات:

متغیر صحیح m حاوی مقدار ۵ و دستور ; n = ++m

متغیر صحیح m حاوی مقدار ۵ و دستور ; n = m++

متغیرهای صحیح k و l به ترتیب حاوی مقادیر ۵ و ۱۰ و دستور `m = (++k) * (l--);`

حالت پسوند دارای اولین تقدم ترتیب اجرا از چپ به راست

حالت پیشوند دارای دومین تقدم، مثل نظیر منفی کردن و نقیض و ترتیب اجرای از راست به چپ

عدم وجود هیچ قاعده ای در مورد نحوه ی محاسبه ی یک عبارت هنگامی که آثار جانبی در آن مطرح باشد در استاندارد زبان C

اجتناب از نوشتن عباراتی که نتیجه ی محاسبه ی آن به ترتیب انجام مواردی که در استاندارد زبان تعریف نشده است بستگی دارد

به عنوان مثال در کامپایلر Borland C Version ۳.۱

با متغیر صحیح a حاوی عدد ۴ حاصل عبارت `b = a++ * a++;` **برابر ۱۶ و در متغیر b**

ولی در دستور `printf ("%d\n" , b = a++ * a++);` **مقدار ۲۰ در خروجی و در متغیر b**

جدول ۵-۱۰ حاوی عبارت های نمونه، a و b صحیح و a قبل از اجرای عبارت حاوی عدد یک

جدول ۵-۱۰: نمونه هایی از عبارت دارای ابهام در ترتیب محاسبه همراه با نتیجه ی محاسبه در دو کامپایلر

عبارت با فرض این که a حاوی یک باشد	نتیجه ی اجرا	توضیح
b=(++a)+(a+۱)	a حاوی ۲ و b حاوی ۵ است	داخل زوج پرانتز سمت چپ اول انجام شده است.
b=(a+۱)+(++a)	a حاوی ۲ و b حاوی ۵ است	داخل زوج پرانتز سمت راست اول انجام شده است.
b=((a+۱)+۱)+(++a)	a حاوی ۲ و b حاوی ۶ است	داخل زوج پرانتز سمت راست اول انجام شده است.
b=(++a)+((a+۱)+۱)	a حاوی ۲ و b حاوی ۶ است	داخل زوج پرانتز سمت چپ اول انجام شده است.
b=(a+۱)+(a)+(a)*(++a)	a حاوی ۲ و b حاوی ۹ است	داخل زوج پرانتزها از راست به چپ محاسبه شده است.
b=(a)*(++a)+(a+۱)+(a)	a حاوی ۲ و b حاوی ۹ است	نخست زوج پرانتز دوم از سمت چپ، بعد زوج پرانتز اول از سمت چپ و سپس دو زوج پرانتز بعدی محاسبه شده است.

تشخیص اندازه ی یک متغیر

```
printf("%u bytes for int, %u bytes for float\n",
        Sizeof (int), sizeof (float);
printf("%u bytes for int, %u bytes for float\n",
  sizeof(m), sizeof x);
```

مکمل کردن

چهار عملگر برای انجام عملیات منطقی مستقیم روی بیت ها

عملگر مکمل کردن با علامت ~ ، تبدیل کلیه ی بیت های صفر عملوند به یک و کلیه ی بیت های یک آن به صفر

حاصل این عمل مکمل یک مقدار موجود در عملوند می باشد.

این عملگر دارای دومین تقدم برابر با تقدم سایر عملگرهای یکتایی و ترتیب از راست به چپ

در کامپیوترهایی که از روش مکمل دو برای نمایش اعداد صحیح استفاده می شود، اگر n یک متغیر حاوی یک مقدار صحیح باشد حاصل دو عبارت ۱ + n ~ و n- یکی خواهد بود

استفاده از عملوند اعشاری برای این عملگر توسط کامپایلر غلط گرفته خواهد شد.

۵-۶-۲ عملگرهای دوتایی

اعمال منطقی بیت به بیت

اعمال منطقی روی محتویات خانه های حافظه به صورت بیت به بیت

علامت & برای عطف (and) ، علامت | برای فصل (or) و علامت ^ برای فصل انحصاری (exclusive or) عملوندها باید از یکی از نوع های صحیح باشند و گرنه غلط نحوی گرفته می شود.

& دارای تقدم هشتم، ^ دارای تقدم نهم و | دارای تقدم دهم

تقدم کلی بعد از عملگرهای تساوی و نامساوی (تقدم هفتم) و ترتیب از چپ به راست

نحوه ی کار عملگرهای منطقی فوق در جدول ۵-۱۱

جدول ۵-۱۱: نحوه ی کار عملگرهای منطقی بیت به بیت.

bit_۱	bit_۲	bit_۲&bit_۱	bit_۲|bit_۱	bit_۱ ^ bit_۲	~bit_۱
۰	۰	۰	۰	۰	۱
۰	۱	۰	۱	۱	۱
۱	۰	۰	۱	۱	۰
۱	۱	۱	۱	۰	۰

عملوندهای این عملگرها فقط می توانند انواع مقادیر صحیح باشند

توجه به اطلاعات زیر برای استفاده از عملگرهای فوق

تعداد بیت در هر بایت که در کامپیوترهای فعلی عموماً هشت بیت است.

تعداد بایت های تشکیل دهنده ی نوع های صحیح.

استاندارد نمایش کاراکترها (مثلاً ASCII) .

سیستم نمایش اعداد صحیح (مکمل یک یا مکمل دو) .

یک مورد استفاده ی مهم عملگر عطف (&) انتخاب بیت های خاص از یک خانه ی حافظه

عملگر فصل (|) برای یک کردن بیت های خاصی از یک خانه ی حافظه

فصل انحصاری محتویات یک خانه با مقدار هم طولی که همه ی بیت هایش یک است برای تولید مکمل

مقدار اولیه.

جدول ۵-۱۲ نمونه هایی از عملگرهای فوق

جدول ۵-۱۲ : نمونه هایی از عملگرهای منطقی بیت به بیت.

عبارت	مقدار متغیر یا نتیجه ی عمل در مبنای دو	معادل نتیجه در مبنای ده
bits۱	۰۰۰۰۰۰۰۰۰۰۰۱۱۰۰	۱۲
bits۲	۱۱۱۱۱۱۱۱۱۱۰۱۱۱۰۱	-۳۵
~ bits۱	۱۱۱۱۱۱۱۱۱۱۱۱۰۰۱۱	-۱۳
~ bits۲	۰۰۰۰۰۰۰۰۰۰۱۰۰۰۱۰	۳
bits۱& bits۲	۰۰۰۰۰۰۰۰۰۰۰۱۱۰۰	۱۲
~ bits۱& bits۲	۱۱۱۱۱۱۱۱۱۱۰۱۰۰۰۱	-۴۷
~ bits۱& ~ bits۲	۰۰۰۰۰۰۰۰۰۰۱۰۰۰۱۰	۳۴
~ (bits۱& bits۲)	۱۱۱۱۱۱۱۱۱۱۱۱۰۰۱۱	-۱۳

bits۱ | bits۲	۱۱۱۱۱۱۱۱۱۱۰۱۱۱۰۱	-۳۵
~ (bits۱ | bits۲)	۰۰۰۰۰۰۰۰۰۰۱۰۰۰۱۰	۳۴
bits۱ ^ bits۲	۱۱۱۱۱۱۱۱۱۰۱۰۰۰۱	-۴۷
~ (bits۱ ^ bits۲)	۰۰۰۰۰۰۰۰۰۰۱۰۱۱۱۰	۴۶

انتقال بیت ها (شیفت) به راست و چپ

انتقال بیت ها به راست یا شیفت به راست با علامت << و به چپ با علامت >>

عملوندهای این دو عملگر نیز باید از یکی از نوع های صحیح باشند

انتقال بیت های عملوند سمت چپ به تعداد عملوند سمت راست

عملوند سمت راست باید مثبت کم تر یا مساوی تعداد بیت عملوند دیگر باشد

شیفت به چپ: منطقی و شیفت به راست: منطقی یا حسابی

شیفت به چپ: ضرب در دو و شیفت به راست: تقسیم بر دو

دارای تقدم پنجم بلافاصله بعد از جمع و تفریق و ترتیب از چپ به راست

مثال: متغیر صحیح a حاوی ۱۲ و اجرای دو دستور `b = a >>۲; c = a<<۲`

تعویض جای دو بایت

```
k = ((j >> ۸) & ۰x۰۰FF) | (j << ۸);
```

انتخاب n بیت از a از بیت m ام به طرف راست (شماره بیت های a از راست به چپ و از صفر)

```
b = a >> (m + ۱ - n) & ~(~۰ << n);
```

عملگرهای محاسبه و تخصیص

این عملگرها عبارتند از: +=، -=، *=، /=، %=، &=، |=، ^=، <<= و >>=

مثال:

```
a += ۵;

sum += amount * ۱۰۰;
```

دارای تقدم چهاردهم است مساوی تقدم عملگر تخصیص معمولی و ترتیب اجرای از راست به چپ

۵-۶-۳ عملگرهای سه تایی و چندتایی

عبارت شرطی

انتخاب از بین دو عبارت بر مبنای صفر یا غیر صفر بودن مقدار یک عبارت دیگر

قالب کلی

« عبارت ۳ » : «عبارت ۲ » ؟ «عبارت ۱ »

مثال:

```
printf ("min(%d, %d) is : %d\n", m, n, m < n ? m : n) ;
max = (t = (t = a  > b ? a : b) > c ? t : c) > d ? t : d;
max = (t = a > b ? a : b) > (s = c > d ? c : d) ? t : s;
```

عملگر کاما

کاراکتر کاما به صورت جدا کننده در لیست ها

کاما در نقش عملگر (دستور for)

محاسبه ی عملوندهای عملگر مزبور از چپ به راست

نتیجه نهایی از نظر نوع و مقدار، نتیجه ی سمت راستی ترین عملوند

دارای آخرین (پانزدهمین) تقدم و ترتیب اجرای از چپ به راست

مثال متغیرهای a ، b ، c و d به ترتیب حاوی مقادیر ۴ ، ۱۲ ، ۷ و ۱۸

```
m = (b += a, b++, c*= b, d = a + b + c);
```

۵-۷ تعریف عملگرهای جدید با استفاده از توابع

توابع برای رفع کمبود دستورها

توابعی آماده در فایل های سرآمد مثل stdio.h و math.h

تعریف عملگرهای جدید با نوشتن تابع

تقسیم برنامه های بزرگ به واحدهای کوچک با استفاده از توابع

۵-۷-۱ نحوه ی نوشتن یک تابع

قالب کلی تابع

(لیست حاوی « اسم پارامتر نوع پارامتر » برای هر پارامتر) اسم تابع نوع نتیجه ی تابع

{ ‹ دستورات تعریف نوع متغیرهای محلی›

‹ دستورات اجرایی (بدنه ی تابع) ›

}

تابع محاسبه ی مجموع رقم های یک عدد

```
int  sum_of_digits (int n)
{  int  sd = ۰;
   While (n)
   {  sd += n % ۱۰;
      n /= ۱۰;
   }
   return sd;
}
```

نوع نتیجه: `return ‹عبارت›;` **یا** `return (‹عبارت›);`

این دستور فقط یک مقدار برمی گرداند

اسم تابع: `sum_of_ digits` **وسیله ی شناسایی تابع**

لیست پارامترها: در داخل زوج پرانتز بعد از اسم تابع، انتقال پارامتر انتقال مقدار

جملات تعریف نوع متغیرهای محلی

جملات اجرایی (بدنه ی تابع) : وجود یک (یا بیشتر) دستور `return`

ترتیب قرار گرفتن توابع: لازم نیست ترتیب خاصی رعایت گردد

همه ی توابع از جمله تابع `main` **به طور مجزا و همه در یک سطح نوشته می شوند.**

هر تابع در صورتی در یک تابع دیگر حتی `main` **قابل احضار است که قبل از آن تعریف شده باشد.**

فعلاً فرض می کنیم که کلیه ی توابع و از جمله تابع `main` **در یک فایل قرار دارند .**

۵-۷-۲ نحوه ی احضار یک تابع

ذکر نام آن و فراهم آوردن آرگومان های متناسب با پارامترهای آن

لزوم تطبیق تعداد، ترتیب و نوع آرگومان های ارسالی با تعداد، ترتیب و نوع پارامترهای متناظرشان

۵-۸ برنامه های نمونه

برنامه ی ۵-۴ : برنامه ای بنویسید که تعدادی عدد صحیح و مثبت کوچک تر از ۳۲۷۶۷ را از ورودی خوانده و هر عدد را همراه با تعداد رقم ها، جمع رقم ها و جمع فاکتوریل رقم هایش در خروجی چاپ نماید. خاتمه ی داده های ورودی باید توسط برنامه کنترل گردد و در این صورت برنامه متوقف شود.

```
#include <stdio.h>

main()
{  int num, sd, nd, dg, tn;
   long df, sf;

   while (scanf("%d", &num) != EOF)     /* حلقهٔ تکرار خواندن عددها و انجام عملیات */
   {  sd = nd = sf = ۰;
      tn = num;
      while (num)                       /* حلقهٔ تکرار پیمایش رقم‌ها */
      {  sd += (dg = num % ۱۰);
         nd++;
         df = ۱;
         while (dg)                     /* حلقهٔ تکرار محاسبهٔ فاکتوریل هر رقم */
            df *= dg--;
         sf += df;                      /* محاسبهٔ مجموع فاکتوریل رقمها */
         num /= ۱۰;
      }
      printf("Number:%d, No of digits:%d, Sum of digits:%d",
             tn, nd, sd);
      printf(", Sum of factorial of digits:%ld\n", sf);
   }
   return (۰);
}
```

شکل ۵-۵: متن برنامهٔ ۵-۴، انجام عملیات روی ارقام عددها.

برنامه ی بالا با استفاده از توابع

```
#include <stdio.h>

int sum_of_digits (int n)                /* تعریف تابع محاسبه ی جمع ارقام عدد n */
{ int sd = ۰;                            /* دستور تعریف متغیر محلی */

   While (n)                             /* حلقهٔ تکرار محاسبه ی جمع رقمها */
   {  sd += n % ۱۰;
      n /= ۱۰;
   }
   return sd;                            /* دستور بازگشت برای خاتمه تابع و برگرداندن نتیجه */
}
```



```
long nfact (int n)                       /* تعریف تابع محاسبه ی فاکتوریل رقم n */
{  long nf = ۱;                          /* دستور تعریف متغیر محلی */
```

```
#include <stdio.h>
#define START ۳                                  /* تعریف نقطهٔ شروع اعداد اول */

main()
{  int prm = START, dvd, *pdvd, n, *pn, rem;

   pn = &n;
   pdvd = &dvd;
   scanf("%d", pn);                              /* استفاده از اشاره‌گر برای خواندن */
   while (prm <= *pn)                            /* حلقهٔ تکرار ساختن اعداد */
   {  *pdvd = ۲;
      rem = ۱;
      while (rem > ۰ && dvd < prm)               /* حلقهٔ تکرار بررسی اول بودن */
         rem = prm % (*pdvd)++;
      if (rem > ۰)
         printf("%d\n", prm);
      prm += ۲;
   }
   return (۰);
}
```

```
#include <stdio.h>

main()
{  int num, sd, nd, dg, tn;
   long df, sf;

   while (scanf("%d", &num) != EOF)     /* حلقهٔ تکرار خواندن عددها و انجام عملیات */
   {  sd = nd =sf = ۰;
      tn = num;
      while (num)                       /* حلقهٔ تکرار پیمایش رقم‌ها */
      {  sd += (dg = num % ۱۰);
         nd++;
         df = ۱;
         while (dg)                     /* حلقهٔ تکرار محاسبهٔ فاکتوریل هر رقم */
            df *= dg--;
         sf += df;                      /* محاسبهٔ مجموع فاکتوریل رقمها */
         num /= ۱۰;
      }
      printf("Number:%d, No of digits:%d, Sum of digits:%d",
              tn, nd, sd);
      printf(", Sum of factorial of digits:%ld\n", sf);
   }
   return (۰);
}
```

شکل ۵-۵: متن برنامهٔ ۵-۴، انجام عملیات روی ارقام عددها.

```
int gcd(int m, int n)               /* تابع محاسبهٔ بزرگ‌ترین مقسوم‌علیه مشترک دو عدد */
{  int ir;

   while (ir = m % n)               /* حلقهٔ تکرار محاسبهٔ ب م م دو عدد */
   {  m = n;
      n = ir;
   }
   return (n);                      /* برگرداندن جواب */
}
```

شکل ۵-۷: متن برنامهٔ ۵-۶، تابعی برای محاسبهٔ ب م م دو عدد و برگرداندن آن به‌عنوان نتیجه.

تعریف غیر بازگشتنی (تکراری): $n ! = ۱ \times ۲ \ ۳ \times \ \ldots \ \ \times (n - ۱) \times \ \ n$

تعریف بازگشتنی:

$$n ! = \begin{cases} n \times (n - ۱)! & for \ n \rangle ۱ \\ ۱ & for \ n = ۱ \end{cases}$$

شکل ۵-۹ الف: تعریف غیر بازگشتنی و بازگشتنی برای محاسبه ی فاکتوریل اعداد، برنامه ی ۵-۸

```
#include <stdio.h>
#define MAXNUM ۱۶

long fact(int n)        /* تابع بازگشتی برای محاسبهٔ فاکتوریل */
{  long nfact;

   if (n == ۱)          /* بررسی مقدار آرگومان برای تصمیم‌گیری روی نیاز به احضار بازگشتی */
      nfact = ۱;        /* اعلام مقدار یک به عنوان جواب به دلیل عدم نیاز به احضار بازگشتی */
   else
      nfact = n * fact(n-۱); /* انجام احضار بازگشتی با توجه به مقدار آرگومان و محاسبهٔ نتیجه */
   return nfact;        /* برگرداندن نتیجه */
}

main()                  /* برنامهٔ اصلی جهت خواندن هر عدد و احضار تابع برای محاسبهٔ فاکتوریل آن */
{  int num;

   while (scanf("%d", &num) != EOF)
   {  if (num > MAXNUM)
         printf("%d: Number is too large!\n", num);
      else
         printf(" n = %d, n! = %ld\n", num, fact(num));
   }
   return ۰;
}
```

شکل ۵-۹ ب: متن برنامهٔ ۵-۸، محاسبهٔ فاکتوریل اعداد به روش بازگشتی.