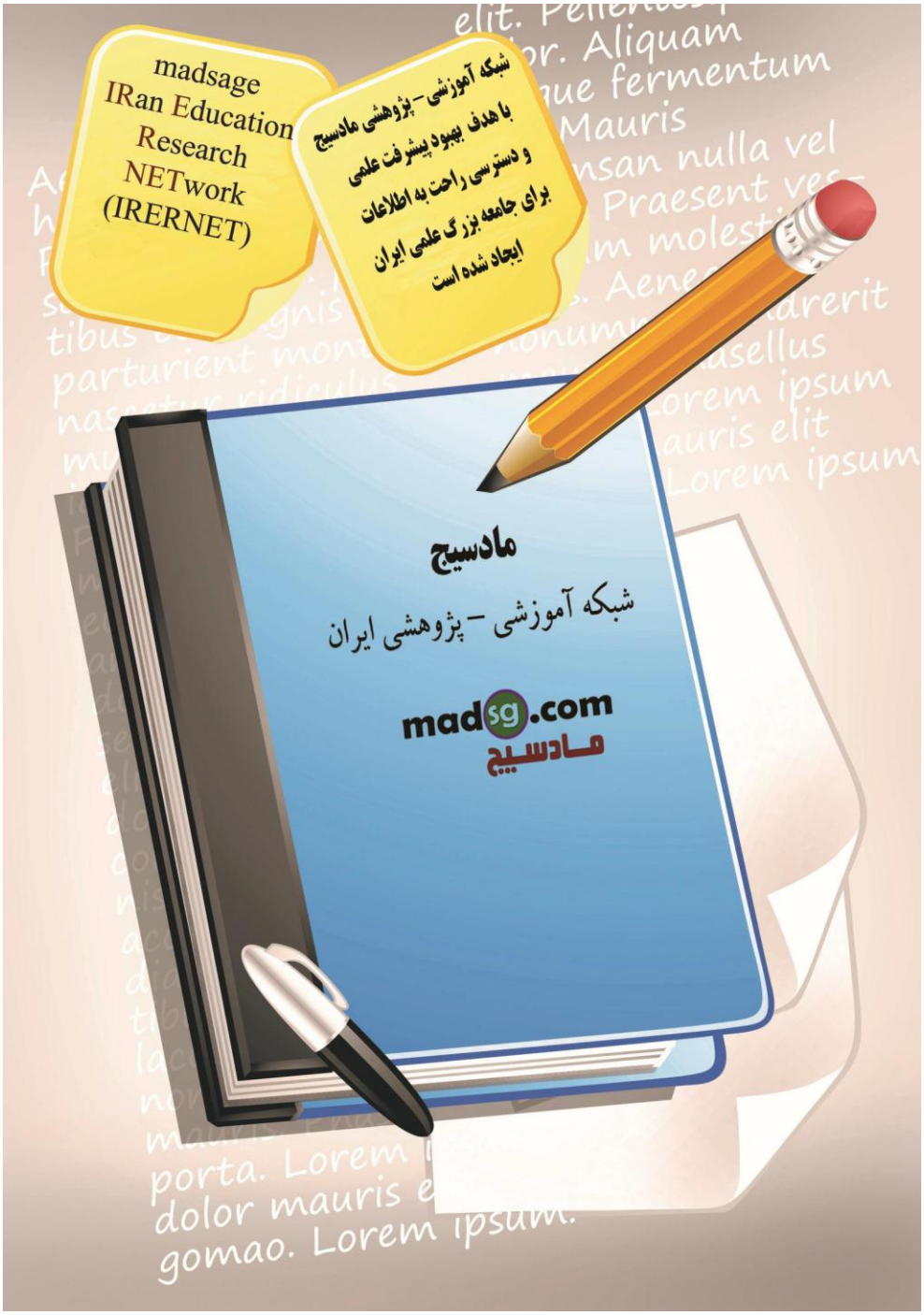
madsage
IRan Education
Research
NETwork
(IRERNET)
شبکه آموزشی-پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
مادسیج
شبکه آموزشی -پژوهشی ایران
madsg.com
مادسیج

فصل 2
انواع داده‌ها

هدف کلی
آشنایی با انواع داده‌های زبان برنامه‌نویسی C و کاربردها و شیوه‌های معرفی آنها

هدفهای رفتاری
از دانشجو انتظار می‌رود پس از خواندن این فصل:
1. انواع داده‌های زبان C را نام برد.
2. مقادیر متغیر و ثابت داده‌ها را بشناسد.
3. انواع مقادیر ثابت را نام برد و تشریح کند.
4. داده‌های اسکالر و مجموعه‌ای را بشناسد.
5. چگونگی اعلان متغیرها در زبان C را بداند.
6. ویژگیهای داده‌های صحیح و شیوه‌های معرفی آنها را بداند.
7. ویژگیهای مقادیر ثابت صحیح بر مبنای 8، 10 و 16 و چگونگی معرفی آنها را بداند.
8. در داده‌های‌اعشاری، روشهای‌نوشتن‌ثابتهای‌با ممیزشناور را شرح‌دهدوکاربرد هریک‌را بداند.
9. در داده‌های کاراکتری کد اسکی و ebedic را تعریف کند.
10. تفاوت بین عدد و کاراکتر را در زبان C بداند.
11. طریقة شناساندن ثابتهای حرفی به مفسر را بداند.
12. طریقة شناسایی حروف کوچک و بزرگ را در کد اسکی بداند.
13. رشته یا ثابت رشته‌ای و طریقة معرفی آن را به مفسر بداند.
14. تفاوت ثابت حرفی و ثابت رشته‌ای تک‌حرفی را بداند.
15. مقداردهی اولیة متغیرها را بداند.
16. وظیفة عملگر cast را شرح دهد.
17. داده‌های تهی و void را بشناسد.
18. پیش‌پردازنده و شیوة معرفی آن را بشناسد.
19. وظیفة فرمان include# و چگونگی تعریف آن را بداند.
20. وظیفة فرمان define# و فواید آن را بداند.

مقدمه
دسته‌بندي داده‌ها به انواع مختلف، يكي از تواناييهاي جديد زبانهاي برنامه‌نويسي است. زبان C مجموعة كاملي از انواع داده‌ها را پشتيباني مي‌كند كه عبارت‌اند از:
- داده‌هاي صحيح (integer)
- داده‌هاي اعشاري (floating point)
- داده‌هاي كاراكتری (character).

همچنين داده‌ها در زبانهاي برنامه‌نويسي به صورت مقادير ثابت و مقادير متغير به كار می‌روند. متغيرها در طول اجراي برنامه، مقادير مختلفي از داده‌ها را می‌پذيرند. اما مقادير ثابت مقاديري‌اند كه در طول برنامه تغيير نمي‌كنند. در زبان C چهار نوع ثابت وجود دارد كه عبارت‌اند از ثابتهاي صحيح، ثابتهاي با ممیز شناور، ثابتهاي كاراكتري و ثابتهاي رشته‌اي.

مقادير صحيح ثابت را مي‌توان علاوه بر روش معمول دهدهي، در مبناهاي هشت و شانزده نيز نوشت. مجموعه داده‌هاي از نوع صحيح و اعشاري را داده‌هاي از نوع محاسباتي يا arithmetic مي‌نامند. دو نوع ديگر از داده‌ها، نوع اشاره‌گر يا pointer و نوع شمارشي يا enumerated است، كه همراه با نوع محاسباتي، داده‌هاي نوع اسكالر ناميده مي‌شود. اين نوع داده‌ها را از اين لحاظ اسكالر مي‌نامند كه قابل مقايسه يا قابل سنجش با همنوع خودند. علاوه بر داده‌هايي از نوع اسكالر، داده‌هايي از نوع مجموعه‌اي وجود دارند از جمله آرايه، ركورد، ساختار و اجتماع كه در سازماندهي متغيرهايي مفيدند كه به طور منطقي به يكديگر مرتبطاند. اين نوع داده‌ها را نيز در فصلهای بعدي بررسي می‌كنيم.

اعلان متغيرها
اعلان، گروهي از متغيرها را به نوع داده خاصي مربوط مي‌سازد. در زبان C هر متغير، پيش از آنكه در دستوري از برنامه به كار رود، بايد تعريف شود. دستورهای مربوط به تعريف متغيرها، اطلاعات لازم در مورد نوع داده‌هايي را كه متغيرهاي مورد نظر مي‌پذيرند و اينكه چند بايت حافظه اشغال مي‌كنند و چگونگي تفسير آنها را در اختيار كامپايلر قرار مي‌دهند. اعلان شامل نوع داده و به همراه آن نام يک يا چند متغير است و به سميكولون ختم مي‌شود. براي اعلان يا تعريف متغيرهايي از نوع صحيح

(integer) كلمة كليدي int و به دنبال آن اسامي متغيرهاي مورد نظر را كه با كاما از يكديگر تفكيك مي‌شوند مي‌نويسيم.

```
int a , b , c ;
```

البته مي‌توان هريك از متغيرها را در دستوري جداگانه و يا در سطري جداگانه معرفي كرد.

```
int a ;
int b ; int c ;
```

كه در سطر اول يک متغير اعلان شده و در سطر دوم با دو دستور جداگانه متغير دوم و سوم اعلان شده است. واضح است روش اول كه در آن هر سه متغير در يك سطر و با يك دستور اعلان شده ساده‌تر است. كلمات كليدي براي اعلان داده‌هايي از نوع اسكالر در جدول 2ـ1 نشان داده شده است.

جدول 2ـ1 كلمات كليدي در اعلان متغيرها

اصلاح/توصيف‌كننده	اصلي
Short long signed unsigned	int float char double enum

پنج كلمة ستون اول نوع اصلي يا پايه‌اي‌اند. چهار كلمة ستون دوم را اصلاح‌كننده[1] يا توصيف‌كننده[2] نامند كه به طريقي پنج نوع اصلي را توصيف مي‌كنند. به عبارت ديگر مي‌توان پنج نوع اصلي را اسم و چهار نوع توصيف‌كننده را صفت براي آن اسامي تصور كرد.

هرگونه اعلان متغيرها در داخل بلاك بايد قبل از اولين دستور اجرايي قرار گيرد. اما ترتيب اعلان آنها فرق نمي‌كند. براى مثال دو روش اعلان زير از نظر نتيجه يكسان‌اند.

روش اول

```
float x , y;
float z ;
int a ;
int b ;
```

روش دوم

```
int a , b ;
float x , y , z ;
```

داده‌هاي صحيح

زبان C از لحاظ بزرگي عناصر و همچنين از نظر نمايش داخلي آنها استاندارد ويژه‌اي به كار نمي‌برد. به طور كلي اعداد صحيح مثبت، منفي و صفر و نيز متغيرهايي كه مقادير صحيح را مي‌پذيرند، 16 يا 32 بيت حافظه اشغال مي‌كنند. فرم اولية داده‌هايي از نوع صحيح، يا همان int مقدار صحيح در نظر گرفته مي‌شود. اما اندازه يا بزرگي آن برحسب نوع ماشين و كامپايلر فرق مي‌كند.

در هنگام تعريف متغيرهاي از نوع int توصيف‌كننده‌هاي short، long، signed و يا unsigned تركيبي از آنها نيز ممكن است به كار رود.

داده‌هايي كه با اين كلمات توصيف مي‌شوند، ممكن است از كامپايلري به كامپايلر ديگر تفسير متفاوت داشته باشند، ولي اساس آنها يكسان است. اگر مقادير صحيح در كامپايلرى در حالت عادي 2 بايت باشد، بين short int و int فرقي نخواهد بود و هر دو 16 بيت يا 2 بايت حافظه اشغال مى‌كنند. در ضمن short int را مي‌توان فقط به صورت short نيز به كار برد (يعني پيش فرض آن است كه short همان short int است). در چنين حالتي long int نيز 4 بايت حافظه اشغال خواهد كرد كه آن را هم مي‌توان فقط به صورت long به كار برد (يعني در اينجا نيز پيش فرض آن است كه long همان long int است). ولي چنانچه مقادير صحيح در حالت عادي 4 بايت حافظه اشغال كنند، short int يا فقط short 2 بايت حافظه به كار خواهد برد. اما بين long int و int (يا فقط long) تفاوتي وجود نخواهد داشت و هر دو 4 بايت حافظه اشغال خواهند كرد.

در مواردي متغيرها، فقط داراي مقادير غيرمنفي خواهند بود. مثلاً متغيري كه براي شمارش به كار مى‌رود، يكي از اين موارد و هميشه مثبت است. زبان C اجازه مي‌دهد كه‌اين گونه متغيرها را با به كار بردن توصيف‌كنندة unsigned، بدون علامت اعلان كنيم. يك مقدار صحيح بدون علامت از نظر ميزان حافظة اشغالي با مقدار صحيح معمولي فرقي ندارد. تفاوت ميان آنها در بيت سمت چپ است كه

1. modifier
2. qualifier

بيت علامت ناميده مي‌شود و در مورد مقادير صحيح بدون علامت، اين بيت نيز مثل ساير بيتها براي نمايش مقدار عدد به كار مي‌رود و در نتيجه مقادير صحيح بدون علامت هميشه غيرمنفي است و بزرگي آن ممكن است تقريباً تا دو برابر مقدار صحيح معمولي باشد. براي مثال عدد صحيح معمولي از 32768- تا 32767+ (در مورد مقادير صحيح دو بايتي) تغيير مي‌كند، بنابراين مقدار صحيح بدون علامت از صفر تا 65535 تغيير خواهد كرد.

در استاندارد ريچي توصيف‌كنندة signed وجود ندارد. ولي استاندارد ANSI آن را پشتيباني مي‌كند. در اغلب موارد به صورت پيش‌فرض، متغيرها signedاند. بنابراين، به استفاده از توصيف‌كنندة signed در آنها نيازی نخواهد بود. در اغلب مفسرها پيش‌فرض براي داده‌هاي كاراكتری signed char است.

متغيرهايي كه معرف اعداد صحيح‌اند به صورتهاي زير توصيف می‌شوند.

short int
int
unsigned int
signed int
long int
unsigned long int
unsigned short int

ϖ **مثال 1.2** در زير نمونه‌هايي از نحوة معرفي متغيرهايي از نوع مقادير صحيح نمايش داده شده است.

1) long int temp , Pnoor ;
2) short int y1 , y2 , y3 ;
3) unsigned int m , n ;
4) unsigned long sum , average ;
5) unsigned short pt , qt ;

تعريف متغيرها از نوع long int و long هم‌ارز است، بنابراين مثال 1 را مي‌توان به صورت long temp , Pnoor ; نوشت. همچنين short int و short نيز معادل هم‌اند. پس مثال 2 را مي‌توان به صورت short y1 , y2 , y3 ; نوشت. مثال 3 را نيز مي‌توان اين طور نوشت: unsigned m , n ;.

ϖ

مقادير ثابت صحيح

در زبان C يكي ديگر از انواع داده‌ها، مقادير ثابت صحيح است. يك مقدار ثابت صحيح عدد و يا دنباله‌اي از ارقام است كه در مبناي 8، 10 و يا 16 تعريف شده باشد. اعداد زير نمونه‌هايي از اعداد با مقادير ثابت صحيح در مبناي 10 اند.

36925 , 9999 , +835 , 512 , 0

در C به طور پيش فرض اعداد صحيح در مبناي 10 تعريف شده‌اند. اما مبناهاي 8 و 16 نيز كاربرد زيادي دارند، زيرا 8 و 16 توانهايي از مبناي 2 اند و اين گونه سيستمهاي عددنويسي براي كامپيوترها مناسب‌تر است. براي مثال عدد 65536 در يك ماشين 16 بيت همان عدد 10000 در مبناي 16 است.

حال ببينيم كامپيوتر چگونه تشخيص مي‌دهد كه عددي در مبناي 8 يا 10 يا 16 تعريف شده؟ براي مشخص ساختن آن از پيشوندهاي 0 براي مبناي 8 و 0x براي مبناي 16 استفاده مي‌شود. مبناي 10 هم كه پيش فرض است و پيشوند ندارد.بنابراين در مورد اعداد

0751 , 0666 +04163 , -0326

صفر سمت چپ به معناي آن است كه اعداد مزبور در مبناي 8 اند، لذا اگر عدد در مبناي 10 باشد، اولين رقم سمت چپ آن نمي‌تواند صفر باشد. بديهي است در مبناي 8 فقط هشت نشانة صفر تا 7 در ارقام به كار مي‌روند. همچنين در مبناي 16 نيز، شانزده نشانة مختلف به كار مي‌رود كه ده نشانة آن همان نشانه‌هاي متداول در مبناي 10 يعني صفر تا 9 است و شش نشانة ديگر حروف A , B , C , D , E , F است كه به ترتيب معادل 10 , 11 , 12 , 13 , 14 , 15 در مبناي 10 اند. مثالهاي زير نمونه‌اي از اعداد مبناي 16 اند.

0xF1E6 , 0x5AB , 0x327 , 0x99

اگر طول هر كلمه در ماشين مورد نظر 16 بيت باشد، طول آن از 32k- تا 32k+ يعني از 32768- تا 32767+ تغيير خواهد كرد كه معادل 2^{15}-1 و يا معادل 077777 مبناي 8 و يا 7FF مبناي 16 است. ولي اگر طول هر كلمه 32 بيت باشد، طول آن از 2G- تا 2G+ خواهد بود يعني از 2,147,483,648- تا 2,147,483,647 كه معادل 2^{31}-1 است.

مقادير ثابت صحيح بدون علامت يا unsigned integer constants با قرار دادن u، حرف اول كلمة unsigned، و همين‌طور مقادير ثابت صحيح طولاني يا long integer constants با قراردادن حرف l،

حرف اول کلمه long، در سمت راست آنها مشخص مي‌گردد كه l و u را می‌توان به هر دو صورت بزرگ يا كوچك نوشت. همچنين اگر عددي هر دو صفت مذكور را داشته باشد (يعني هم بدون علامت و هم به‌صورت طولاني باشد)، با دو حرف ul (u در سمت چپ، l در سمت راست آن) متمايز مي‌شود.

ϖ **مثال 2.2** جدول زير مثالهايي از انواع مقادير ثابت صحيح را نشان مي‌دهد.

مقدار ثابت صحيح	سيستم عددي
546780u	مبناي 10 (بدون علامت)
1234567l	مبناي 10 (طولاني)
1234567ul	مبناي 10 (بدون علامت و طولاني)
0123456u	مبناي 8 (بدون علامت)
0123456ul	مبناي 8 (بدون علامت و طولاني)
0563214l	مبناي 8 (طولاني)
02677u	مبناي 8 (بدون علامت)
0x12545678l	مبناي 16 (طولاني)
0xABCDEFu	مبناي 16 (بدون علامت)
0xEF1A3Abul	مبناي 16 (بدون علامت و طولاني)

توابع scanf و printf در فرمت مربوط به خواندن و نوشتن مقادير صحيح در مبناي 8 و 16 به ترتيب حروف o و x را مشخص‌كنندة فرمت در رشته كنترل فرمت به كار مي‌برند.

ϖ

ϖ **مثال 3.2** برنامة زير عددي در مبناي 16 (با پيشوند 0x يا بدون آن) را از طريق ترمينال مي‌خواند و معادل آن را در مبناهاي 10 و 8 چاپ مي‌كند.

```
# include<stdio.h>
main ()
  {
        int n ;
        printf ("Enter a hexadecimal constant:  \n") ;
        scanf ("%x",&n) ;
        printf ("The decimal equivalent of %x is:  %d ", n , n) ;
        printf ("\n The octal equivalent of %x is:  %o\n", n , n) ;
  }
```

ϖ

داده‌هاي اعشاري

در زبان C اعداد اعشاري نيز قابل نمايش است. داده‌هاي صحيح در بسياري موارد مناسب است. اما براي مقادير خيلي بزرگ و براي مقادير كسري كوچك كه در اغلب زمينه‌هاي علمي كاربرد دارد، به داده‌هاي از نوع ممیز شناور يا floating point نياز است. براي نوشتن ثابتهاي با ممیز شناور دو روش به كار می‌رود.

روش اول كه ساده‌ترين راه است، آن است كه از علامت ممیز (كه در انگليسي "." است) استفاده كنيم. مثالهاي زير از اين نوع است.

0.996 , 15.0 , 3.1415 , 7. ,.275

روش دوم كه نمايش علمي[1] نيز ناميده مي‌شود، روش كوتاه‌نويسي مفيدی است. در اين روش هر مقدار شامل دو جزء است: يك قسمت عددي كه آن را مانتيس نامند و به‌دنبال آن يك قسمت نما يا توان مي‌آيد كه قسمت مانتيس بايد در 10 به توان نما ضرب شود. بين اين دو قسمت حرف E يا e (كه حرف اول exponent است) به مفهوم نما يا توان به كار می‌رود. براي مثال3E2 به مفهوم $3*10^2$ و 125.7E-3- به مفهوم $-125.7*10^{-3}$ است. در واقع يك مقدار ثابت با ممیز شناور، عددي است در مبناي 10 كه شامل يك ممیز يا علامت اعشار يعني "." يا شامل يك نما يا هر دو است؛ مانند مثالهاي زير

182.25 , -3.1415 , 5E-5 , 0.775E-3 0. , 0.0 , 1. , 0.92

البته قسمت نما نمي‌تواند يك عدد كسري باشد، پس 29.5E3.4 درست نيست. در بعضي

1. scientific notation

نسخه‌هاي C براي اينكه مشخص كنند كه مقادير مورد نظر يك كلمه اشغال كرده است حرف F (يا f) را به آخر آن اضافه مي‌كنند، مانند 6.125E5F .

همچنين براي مشخص كردن اينكه مقادير مورد نظر فضايي به طول دو كلمه را اشغال كرده است، حرف L (يا l) به آخر آن اضافه مي‌شود مانند 0.123456789E-25L .

دقت مقادير ثابت با ممیز شناور ممكن است برحسب نسخه‌هاي مختلف تغيير كند، ولي همة آنها حداقل 6 رقم با معني را مي‌پذيرند.

براي اعلان متغيرهايي از نوع floating point از دو كلمة كليدي "float" و "double" استفاده مي‌شود، مانند

```
float a , b , c ;
double x , y , z ;
```

كه در آن كلمه "double" به مفهوم دقت مضاعف يا double precesion است و در اغلب ماشينها طول فضايي كه براي متغيرهاي توصيف شده با آن رزرو مي‌شود، دو برابر "float" است. يك متغير توصيف شده با "float" به‌طور متعارف 4 بايت در حافظه اشغال مي‌كند، پس در "double" اين فضا 8 بايت خواهد شد و نمايش دروني مقادير floating-point نيز از ويژگيهاي معماري سخت‌افزار كامپيوتر است و هنوز كاملاً استاندارد نيست. در مورد ميزان دقت float و double نيز بايد به مستندات كامپايلر مربوط مراجعه كرد. در برخي كامپايلرها براي اعلان متغيرهاي از نوع double نيز مي‌توان آنها را به صورت long float تعريف كرد. بنابراين دو روش اعلان زير معادل‌اند.

```
long float a , b , c ;          double a , b , c ;
```

به هرحال اگر كلمة توصيف‌كنندة long به تنهايي جلوي متغيري به كار رود، آن متغير به صورت پيش‌فرض از نوع مقادير صحيح خواهد بود. بنابراين با دستور long a , b , c ; سه متغير a , b , c از نوع صحيح خواهند بود.

داده‌هاي كاراكتري

يكي از انواع داده‌هايي كه در برنامه‌نويسي استفاده مي‌شود داده‌هاي كاراكتري است. در بسياري از زبانهاي برنامه‌سازي داده‌هاي عددي و داده‌هاي كاراكتري با يكديگر تفاوت دارند. مثلاً عدد 2 داده‌ای عددی و حرف A داده‌ای كاراكتري است. در عمل هم، كاراكترها به صورت عدد در حافظة كامپيوتر ذخيره مي‌شوند، و هر كاراكتر داراي يك كد عددي است. كدهاي مختلفي وجود دارد كه دو نوع آن عبارت‌اند از اسكی (ascii)[1] يا كد استاندارد آمريكايي براي تبادل اطلاعات و ديگري ebcdic[2] يا كد توسعه‌يافتة bcd كه آی‌بی‌ام روي سيستم بزرگ خود به كار مي‌برد و بيشتر متداول است. البته در زبان C، كد اسكی متداول است. در همة كدگذاريها براي هر كاراكتر نماد عددي وابسته‌ای وجود دارد كه در كدگذاري اسكي به آن ascii code گويند و مقدار آن از صفر تا 255 است.

در زبان C تفاوت بين اعداد و كاراكتر ناچيز است. در اين زبان يكي از انواع داده‌ها char ناميده مي‌شود، اما در حقيقت كاراكتر مقدار صحيح يك بايتي است كه هم براي نگهداري اعداد و هم براي نگهداري كاراكترها به كار می‌رود.

ثابتهاي حرفي در داخل يك زوج گيومه قرار مي‌گيرد. اين گيومه‌ها به كامپايلر اعلان مي‌كند كه كد عددي كاراكتر مورد نظر را به دست آورد. براي مثال در دستورهاي

```
char a , b ;
b = 5 ;
a = '5';
```

مقدار a برابر 53، يعني برابر كد اسكي كاراكتر '5' و مقدار b برابر 5 خواهد بود.

مثال 4.2 برنامة زير كاراكتری را از ورودي می‌خواند و كد عددي آن را نمايش مي‌دهد.

```
#include <stdio.h>
main()
  {
        char ch ;
        scanf ("%c", &ch) ;
        printf (" The numeric code is:  %d \n ", ch) ;
  }
```

1. American standard code for information interchange
2. Extended binary-coded decimal interchange

همچنين در توابع printf و scanf نماد %c فرمت متغيرهاي كاراكتري است مانند %d كه براي متغيرهاي مقادير صحيح و %f كه براي متغيرهاي مقادير اعشار به كار می‌رود (توابع ورودي و خروجي و فرمت متغيرها را در فصل 3 بررسي می‌كنيم).

ϖ

در مجموعه كاراكتر اسكي، ترتيب كد كاراكترها براساس ترتيب كاراكترهاست. براي مثال، كد حرف 'A' برابر 65، 'B' برابر 66، ...، و 'Z' برابر 90 است. ~~كد حروف كوچك از 97 تا 122 است. با توجه به~~ ارزش كد حروف، تابع زير حروف بزرگ را به حروف كوچك تبديل مي‌كند (توابع را در فصل 6 بررسي مي‌كنيم).

```
char Up-to-low (ch)
char ch ;
   {
        return ch + 32 ;
   }
```

تابع مذكور به كد اسكي هر كاراكتر دريافتي 32 واحد اضافه مي‌كند كه درنتيجه حروف بزرگ به حروف كوچك تبديل مي‌شود. مثلاً كد اسكي حرف 'A' (65) 32 واحد از كد اسكي حرف 'a' (97) كوچك‌تر است. مشابه آن مي‌توان تابعي براي تبديل حروف كوچك به بزرگ تعريف كرد كه در اين حالت بايد از كد اسكي حرف مورد نظر 32 واحد كسر گردد تا به حرف بزرگ مشابه خود تبديل شود.

در سيستم كدگذاري غير اسكی، مانند ebcdic، تفاوت عددي كد حروف بزرگ و كوچك 32 نيست. بنابراين در چنين حالتي تابع تعريف شدة بالا نتيجة مطلوب را نمي‌دهد. براي جلوگيري از چنين اشتباهي زبان C داراي توابع كتابخانه‌اي به اسامي toupper و tolower است كه به ترتيب عمل تبديل كاراكترها را از بزرگ به كوچك و از كوچك به بزرگ انجام مي‌دهند.

ثابتهاي رشته‌اي

رشته يا ثابت رشته‌اي از تعدادي كاراكتر متوالي تشكيل می‌شود كه بين دو گيومه قرار می‌گيرند. به عبارت ديگر شامل دنباله‌اي از كاراكترهاست كه در بين دو گيومه قرار دارند، مانند نمونه‌هاي زير.

"university" , "256" , "payam noor" , "1380-02-06" , "five$" , "p4"

همچنين بايد توجه داشت كه " " نيز رشته‌ای تهي (empty) يا null است.

ϖ **مثال 5.2** ثابت رشته‌اي زير شامل سه كاراكتر مخصوص است كه با escape sequence متناظرشان نشان داده شده‌اند.

```
"\t to continue , press the \"RETURN\" KEY\n"
```

كه در آن نشانه‌ها يا كاراكترهاي مخصوص عبارت‌اند از

\t horizontal tab،

\" گيومه يا quotation mark دوبل كه دو بار به كار رفته است،

\n خط جديد يا new line.

ϖ

كامپايلر به طور خودكار يك كاراكتر null (\0) در پايان هر ثابت رشته‌اي قرار مي‌دهد كه آخرين كاراكتر در داخل رشته (قبل از بسته شدن گيومه) خواهد بود. اين كاراكتر وقتي كه رشته نمايش داده شود رؤيت‌پذير نيست. به هرحال مي‌توان هريك از كاراكترها را در داخل رشته امتحان كرد كه آيا كاراكتر null است يا نه. در خيلي موارد مشخص ساختن پايان يك رشته با يك كاراكتر مخصوص مانند كاراكتر null نياز به تعيين حداكثر طول براي رشته را از بين مي‌برد. به عنوان مثال رشتة فوق 38 كاراكتر دارد كه شامل پنج فضاي خالي و چهار كاراكتر مخصوص است كه با escape sequence معرفي شده‌اند و در پايان كاراكتر null است كه انتهاي رشته را مشخص مي‌سازد.

يك ثابت حرفي مانند 'P' با يك ثابت رشته‌اي تك‌حرفي متناظر آن مانند "P" هم‌ارز نيست. همچنين به‌خاطر داشته باشيد كه در جدول كد اسكي، هر كاراكتر داراي يك مقدار عددي است، ولي يك رشتة تك‌حرفي اين‌طور نيست. در واقع يك رشتة تك‌حرفي متشكل از دو كاراكتر است كه كاراكتر دوم همان كاراكتر null است كه پايان رشته را مشخص مي‌سازد.

باز هم توجه داشته باشيد كه يك رشتة n كاراكتري به آرايه n+1 عنصري نياز خواهد داشت، زيرا كاراكتر null نيز به طور خودكار به عنوان كاراكتر پاياني در آن قرار داده خواهد شد. براي مثال اگر رشتة "COMPUTER" در يك آراية يك بعدي كاراكتري به نام book ذخيره گردد، خانه اول آن، يعني book[0]، شامل كاراكتر C و خانة آخر، يعني book[8] شامل كاراكتر null خواهد بود كه معرف پايان رشته است.

دربارة رشته‌ها و آرايه‌ها و كاربرد آنها در فصل جداگانه‌اي به طور مشروح بحث خواهيم كرد.

مقداردهی اولیة متغیرها

در صورتي كه از پيش مقدار شروع متغير را بدانيم مي‌توانيم به هنگام تعريف، مقدار اولية مورد نظر را نيز به آن اختصاص دهيم. براي اين كار در تعريف متغيرها، به دنبال نام آن، اپراتور جايگزيني '=' را همراه با مقدار اوليه به كار مي‌بريم. برای مثال هر يك از روشهاي زير متغيرهاي a , b , c را توصيف می‌کنند و به ترتيب مقادير 25- , 13 , 12 را به آنها اختصاص مي‌دهند.

روش اول	روش دوم	روش سوم
int a =12; int b =13 ; int c = -25 ;	int a =12 , b =13 ; int c = -25 ;	int a =12 , b =13 , c = -25 ;

اما در دستور زير فقط به b مقدار اوليه داده شده است.

```
int a , b = 15 , c ;
```

در اين گونه موارد براي جلوگيري از اشتباه بهتر است متغيرهايي را كه مقدار اوليه مي‌پذيرند جدا از ساير متغيرها توصيف كرد، مانند مثال زير.

```
int a =12 , b =13 , c =25 ;
int d , e , f ;
```

ϖ **مثال 6.2** چند نمونة ديگر از مقداردهی اولية متغيرها در زير نشان داده شده است.

```
int sum = 5 ;
char Str = ' # ' ;
float tmp = 10.2 ;
double p1 = 0.1234E-6 ;
```

ϖ

عملگر cast

مي‌توان تبديل يك نوع به نوع ديگر را به صورت صريح انجام داد. اين كار به كمك عملگر cast انجام مي‌گيرد. پس ساختار cast نوع ديگر از تبديل است. براي اين كار كافي است نوع جديد داده مورد نظر را در داخل پرانتز مستقيماً جلوي عبارت قرار دهيم؛ براي مثال

```
k = (float)2 ;
```

مقدار صحيح 2 را قبل از اختصاص‌دادن به k به float تبديل مي‌كند و سپس آن را به k اختصاص مي‌دهد. بنابراين، اپراتور cast اپراتور يكانی است؛ يعني فقط يك اپرند دارد.

در موارد متعددي روش casting خيلي مفيد است. براي مثال حالت زير را درنظر بگيريد.

```
int i =2 , k =3 ;
float h = k / i ;
```

در اينجا مقدار k / i (يعني 2 / 3) برابر 1.5 خواهد شد. سپس نتيجه به float يعني 1.0 تبديل و به h نسبت داده مي‌شود. حال می‌خواهيم مقدار 1.5 را كه نتيجة واقعي عبارت رياضي 3/2 است به k و i يا هر دوي آنها را با cast به float تبديل كنيم، مثلاً

```
(float) k / i ;
```

در اينجا به‌طور صريح k به float تبديل مي‌گردد، پس نتيجه برابر 1.5 خواهد شد. عبارت مزبور را مي‌توان به صورت k / (float) i ; يا (float) k / (float) i ; نيز نوشت كه نتيجه باز هم 1.5 مي‌گردد، يعني نتيجه سه روش مزبور هم‌ارز است.

از مثالهاي بالا نتيجه مي‌شود كه به كمك casting مي‌توان در وسط جمله نوع داده را به نوع ديگري تبديل كرد. بنابراين اپراتور cast به‌عنوان نوع يا type عمل مي‌كند؛ يعني type conversion است و فرمت آن به‌اين طريق است كه نوع جديد متغير يا عبارت مورد نظر جلوي آن متغير يا عبارت در داخل پرانتز نوشته شود. براي مثال دستور (int)d1+d2 يعني اول d1 به int تبديل مي‌شود بعد با d2 جمع مي‌شود. درحالي كه دستور (int)(d1+d2) يعني نتيجة d1+d2 به int تبديل مي‌شود.

بنابراين فرمت اپراتور cast به صورت زير است.

```
(data type) expression
```

حال براي آنكه نقش اپراتور cast را بهتر متوجه شويد، به نتيجه و عملكرد دو مجموعه دستورهای زير توجه كنيد.

مثال اول

```
float x ;
printf ("%d\n",(int)x) ;
```

مثال دوم

```
short int x ;
printf ("%s\n",(char)x) ;
```

در مثال اول براي متغير x كه از نوع float اعلان شده است، 4 بايت حافظه پيش‌بيني مي‌شود. ولي در نتيجة اجراي دستور printf سطر دوم، به دليل دستور (int)x مقدار آن به نوع int تبديل مي‌گردد و نمايش داده مي‌شود. بنابراين، اگر براي مثال محتواي حافظه به‌صورت

0.26 E+7

باشد، مقدار 2600000 نمايش داده خواهد شد.

همين‌طور در مثال دوم براي متغير x كه از نوع short int اعلان شده است، 2 بايت حافظه پيش‌بيني مي‌شود، ولي درنتيجة اجراي دستور printf سطر دوم، به دليل (char)x مقدار آن به نوع كاراكتر تبديل مي‌گردد و محتواي دو بايت حافظة مزبور به‌صورت يك رشتة دوبايتي نمايش داده مي‌شود. به همين دليل است كه در فرمت چاپ مقدار متغير مزبور، از فرمت "%s" كه براي رشته است استفاده شده است. حال اگر براي مثال محتواي حافظة مربوط به متغير x به صورت 4 3 2 1 باشد، موقع نوشتن به صورت رشتة "4 3 2 1" چاپ مي‌شود.

در اينجا به اختصار يادآور مي‌شويم كه فرمتهاي "%d", "%f", "%c", "%s" به ترتيب براي متغيرهاي از نوع مقادير صحيح، اعشار، كاراكتر و رشته به كار می‌روند.

نوع void

داده از نوع void (تهي) در استاندارد ريچي وجود نداشت و بعد به استاندارد ANSI افزوده شد. از اين نوع داده هدف مهمي مورد نظر است و آن در مورد معرفي توابعي است كه مقداري را برنمي‌گردانند، بلكه فقط عمل خاصي را انجام مي‌دهند.

ϖ **مثال 7_2** تابعي به نام FF1 كه داراي دو آرگون x و y است به صورت زير تعريف شده است.

```
void FF1(x , y)
int x , y ;
   {
        --------------
        --------------
        --------------
   }
```

قرار گرفتن void در جلوي نام تابع مزبور به‌اين دليل است كه اين تابع چيزي را برنمي‌گرداند.

ϖ

پيش‌پردازنده

پيش‌پردازنده را مي‌توان برنامة جداگانه‌اي در نظر گرفت كه قبل از كامپايلر واقعي اجرا مي‌گردد. هنگامي كه برنامه‌ای را كامپايل مي‌كنيد، پيش‌پردازنده به طور خودكار اجرا مي‌گردد. تمام فرامين پيش‌پردازنده با علامت " # " شروع مي‌گردند كه بايد اولين كاراكتر خط باشد. وظيفة اصلي و مهم پيش‌پردازنده آن است كه فايل درخواستي را آماده سازد و وارد برنامه كند. برخلاف دستورهای C كه به سميكولون ختم مي‌شوند، پايان جمله‌های آن با خط جديد مشخص مي‌گردد. دو دستور متداول از پيش‌پردازنده‌ها را در ادامه بررسي مي‌كنيم.

فرمان #include

فرمان #include موجب مي‌گردد كه كامپايلر همزمان با فايلي كه ترجمه مي‌كند، يك متن را نيز از فايل ديگر بخواند. اين عمل شما را قادر مي‌سازد كه بتوانيد قبل از شروع ترجمه، محتواي يك فايل را در فايل ديگر بريزيد (درج كنيد)، ضمن اينكه فايل اوليه تغيير نمي‌يابد. اين عمل به ويژه در حالتي كه بيش از يك فايل مبنا، اطلاعاتي يكسان را سهيم شوند و به كار ببرند مفيد است. با اين كار به جاي اينكه اطلاعات مشترك دو فايل را به صورت دوبله در هر دو منظور كنيد، آن را فقط در يك فايل قرار مي‌دهيد، سپس هر موقع آن اطلاعات در فايل دوم مورد نياز باشد، به طريق مذكور آن را براي استفادة فايل دوم نيز آماده مي‌كنيد.

تعريف فرمان #include به دو شكل زير است.

شكل اول	**شكل دوم**
# include < filename >	# include "filename"

در شكل اول، پيش‌پردازنده فقط محل خاصي را كه با عامل مشخص شده است نگاه مي‌كند. اين محل جايي است كه include fileهاي سيستم، مانند header files براي كتابخانة زمان اجرا يا Runtime Library نگهداري مي‌شوند. اگر شكل دوم به كار رود، پيش‌پردازنده فهرست يا دايركتوري را كه شامل فايل مبناست نگاه مي‌كند. اگر فايل include file را در آنجا پيدا نكند، پس از آن مشابه

شكل اول، محل خاص مورد نظر را جستجو مي‌كند. اسامي include fileها بر حسب قرارداد، به پسوند "h" ختم مي‌شوند.
حال ببينيم وقتي كه پيش‌پردازنده با فرمان include<stdio.h> # مواجه مي‌شود، چه پيش مي‌آيد؟ پيش‌پردازنده فهرست تعريف شده در سيستم را براي فايلي به نام stdio.h جستجو مي‌كند. سپس فرمان include# را با محتواي فايل جايگزين مي‌كند.
براي اينكه بدانيد فرمان include# چگونه كار مي‌كند، فرض كنيد كه فايلي به نام file1.h داريد كه محتواي آن فقط دو دستور زير است.

```
int st-no ;
char name ;
```

سپس در فايل مبنا، فرمان include# را به صورت زير به كار مي‌بريد.

```
#include "file1.h"
main()
  {
        ----------
        ----------
  }
```

حال وقتي كه برنامة مزبور را ترجمه مي‌كنيد، پيش‌پردازنده به جاي فرمان include# محتواي فايل مشخص شده را قرار مي‌دهد. بنابراين فايل مبنا به صورت زير درمي‌آيد.

```
int st-no ;
char name ;
main()
  {
      ------------
      ------------
  }
```

فرمان define#

همان طور كه مي‌توان با توصيف يا اعلان متغير، اسمي را به يك محل از حافظه وابسته كرد و به آن محل با آن نام (كه همان متغير مورد نظر است) مراجعه كرد، به همان طريق مي‌توان اسمي را به يك مقدار ثابت وابسته كرد و آن را با همان اسم كه ثابت سمبوليكي ناميده مي‌شود مشخص و هنگام نياز به آن مراجعه كرد. اين گونه متغيرها را كه معمولاً با حروف بزرگ معرفي مي‌شوند ثابتهاي سمبوليكي يا symbolic constants گويند و اين عمل با دستور define# انجام مي‌گيرد.
براي مثال با دستور define book 15 # مي‌توان در هر جاي برنامه به جاي 15 از book استفاده كرد. بنابراين دو دستور زير هم‌ارزند.

```
k = 12 + 15 ;
k = 12 + book ;
```

هر دو دستور مقدار 27 (27 = 15+ 12) را به متغير k نسبت مي‌دهند.
براساس دستور define مقدارbook در حافظه به صورت مقدار ثابت 15 است كه در طول برنامه تغيير نمي‌كند. انتخاب نام براي مقادير ثابت چند فايدة مهم دارد.
اول آنكه به بعضي مقادير ثابت مي‌توان اسم با معني اختصاص داد. مثلاً مي‌توان عدد معروف «پي» را كه تا 4 رقم اعشار معادل 3.1415 است، در آغاز به صورت

```
#define Pi 3.1415
```

تعريف كرد و سپس در سرتاسر برنامه، به جاي عدد مزبور در تمام محاسبات وابسته به آن Pi را به كار برد.
دوم آنكه اگر مجبور باشيم در برنامه‌اى يك مقدار ثابت طولاني نامأنوس را چندين بار به كار بريم، ساده‌تر آن خواهد بود كه با دستور define# نامى مناسب براي آن انتخاب كنيم و به جاي ثابت مزبور از آن نام استفاده كنيم. مثلاً اگر مجبور باشيم همان عدد پي را با 6 رقم اعشار در قسمتهاي متعددي از برنامه به كار بريم، اين عمل هم پردردسر و هم اشتباه‌زا خواهد بود. لذا بهتر است، مثل حالت قبل، يك اسم براي آن انتخاب كنيم.
سوم آنكه ممكن است طبيعت مقدار ثابت طوري باشد كه در زمانهاي مختلف تغيير كند، مثل نرخ ماليات، يا اجرت ساعت كار و يا درصدي كه به عنوان سود در بانكها به پس‌اندازهاي پولي يا سرمايه‌گذاري تعلق مي‌گيرد كه هميشه ثابت نيست و ممكن است برحسب مقررات، قوانين و ساير شرايط مقدار آن عوض شود. در چنين مواردي بايد در سرتاسر برنامه مقدار ثابت موردنظر را عوض كنيم. درحالي كه اگر با دستور define# اسمي براي آن انتخاب كرده باشيم كافي است فقط در

همان يك دستور تغيير مورد نظر را اعمال كنيم. مثلاً اگر براساس قوانين، نرخ جديد ماليات بر درآمد برابر 2 درصد باشد، كافي است به آن قبلاً نام TAX را اختصاص داده باشيم و با دستور #define TAX 0.02 مقدار جديد را جايگزين قبلي كنيم و ديگر نيازي نيست كه در داخل برنامه تغييراتي انجام دهيم. از مثال بالا مشخص مي‌گردد كه يك ثابت سمبوليكي نامي است كه جايگزين دنباله‌اي از كاراكترها مي‌گردد. كاراكترها ممكن است يك ثابت عددي، يك ثابت كاراكتري، و يك ثابت رشته‌اي باشند. بنابراين يك ثابت سمبوليكي اجازه مي‌دهد كه در يك برنامه به جاي يك مقدار ثابت (عددي، حرفي يا رشته‌اي) يك اسم قرار گيرد. وقتي كه برنامه ترجمه مي‌گردد، در هر محلي از برنامه كه ثابت سمبوليكي قرار گرفته باشد، دنبالة كاراكترهاي متناظر آن جايگزين مي‌گردد. ثابتهاي سمبوليكي معمولاً در آغاز برنامه تعريف مي‌گردند و شكل آنها به صورت زير است.

#define name text

كه در آن name معرف نام سمبوليك است كه معمولاً با حروف بزرگ نوشته مي‌شود. text نيز دنباله‌اي از كاراكترها را كه بايد به نام سمبوليك اختصاص داده شود معرفي مي‌نمايد. توجه داشته باشيد كه text به سميكولون ختم نمي‌گردد، زيرا تعريف ثابت سمبوليك دستور واقعي C نيست.

ϖ **مثال 8_2** در زير نمونه‌هاي ديگري از ثابتهاي سمبوليكي نشان داده شده است.

```
#define        Temp     524
#define        Pi       3.1415
#define        true     1
#define        Name     " payam noor "
#define        f(x)      x + 1
```

خصيصة #define كه براي تعريف ثابتهاي سمبوليك به كار می‌رود يكي از چندين خصيصه‌اي است كه در پيش‌پردازنده وجود دارد.

ϖ

خودآزمایی 2

1. برنامه‌اي بنويسيد كه مساحت دايره‌اي به شعاع R را محاسبه كند و نمايش دهد.
2. برنامه‌اي بنويسيد كه كاراكتري را از ورودي بخواند و كد اسكي آن را چاپ كند.
3. برنامه‌اي بنويسيد كه مجموع 100 جملة اول سري زير را محاسبه و چاپ كند.

y = 1 + 1/2 + 1/3 + 1/4 + ...

4. برنامه‌اي بنويسيد كه ضرايب معادله‌های درجه دوم را بخواند، جوابهاي آن را به دست آورد و چاپ كند. اگر معادله ريشة حقيقي نداشته باشد، پيغام مناسب نمايش دهد.
5. خروجي برنامة زير چيست ؟

```
main()
  {
        unsigned char ch ;
        ch = -12 ;
        printf ("%d" , ch) ;
  }
```

6. خروجي برنامة زيرچيست ؟

```
main ()
  {
        unsigned char c ;
        c = 100 * 4 ;
        printf ("%d" , c) ;
  }
```


شبکه آموزشی- پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
madsg.com
مادسیج
IRan Education & Research NETwork
(IRERNET)