

به نام خدا

مبانی برنامه نویسی

آرش شفیعی



## عبارات و نوع‌های داده‌ای

## نوع‌های داده‌ای

- یک متغیر<sup>1</sup> نامی است برای یک مکان در حافظه. دادهٔ ذخیره شده در یک مکان حافظه متناظر با یک متغیر، در طول یک برنامه می‌تواند تغییر داده شود.
- در تعریف یک متغیر نوع متغیر تعیین می‌شود و می‌توان یک مقدار اولیه نیز برای آن تعیین کرد. از عملگرها<sup>2</sup> برای انجام عملیات محاسباتی و منطقی بر روی متغیرها استفاده می‌شود.
- یک عبارت<sup>3</sup> با ترکیب مقادیر عددی و غیر عددی، متغیرها و ثابت‌ها<sup>4</sup> به عنوان عملوند<sup>5</sup> و همچنین عملگرها مقادیر مورد نیاز را محاسبه می‌کند.
- نوع یک متغیر مشخص می‌کند چه مقادیری می‌تواند داشته باشد و چگونه عملگر بر روی آن عملیات انجام می‌دهد. در این قسمت به معرفی نوع‌های داده‌ای می‌پردازیم.

---

<sup>1</sup> variable

<sup>2</sup> operator

<sup>3</sup> expression

<sup>4</sup> constants

<sup>5</sup> operand

- نام متغیرها می‌تواند از حروف الفبا و ارقام و زیرخط <sup>1</sup> ( \_ ) تشکیل شده باشد ولی اولین کاراکتر نمی‌تواند یک رقم باشد.
- حروف بزرگ و کوچک الفبا با یکدیگر متفاوت‌اند، بنابراین  $x$  و  $X$  دو متغیر متفاوت هستند. برنامه نویسان سی معمولاً از حروف کوچک برای نام متغیرها استفاده می‌کنند و نام نمادهای ثابت معمولاً با حروف بزرگ نوشته می‌شود.
- از کلمات کلیدی مانند `if` ، `else` ، `int` نمی‌توان به عنوان نام متغیرها استفاده کرد.
- بهتر است برای متغیرها از اسامی معنی‌دار استفاده شود تا برنامه خوانایی بیشتری داشته باشد.

---

<sup>1</sup> underscore

# نوع‌های داده‌ای

- نوع داده‌های اصلی در زبان سی عبارتند از :
- نوع کاراکتر <sup>1</sup> char که یک بایت است.
- نوع عدد صحیح <sup>2</sup> int که اندازه آن به ماشین بستگی دارد ولی معمولاً ۴ بایت است.
- نوع عدد اعشاری <sup>3</sup> float یا ممیز شناور که ۴ بایت است.
- نوع اعشاری با دقت دو برابر <sup>4</sup> double که ۸ بایت است.

---

<sup>1</sup> character

<sup>2</sup> integer

<sup>3</sup> floation point

<sup>4</sup> double-precision float point

- همچنین می‌توان از کلمات کلیدی توصیفی short و long قبل از تعریف متغیرها استفاده کرد. برای مثال متغیر از نوع int مقدار ۴ بایت را اشغال می‌کند و متغیر از نوع short int مقدار ۲ بایت. همچنین متغیر از نوع long int مقدار ۸ بایت را اشغال می‌کند.
- از کلمات توصیفی signed و unsigned نیز می‌توان برای تعیین علامت‌دار بودن یا بدون علامت بودن متغیرها استفاده کرد.
- وقتی یک متغیر علامت‌دار است یک بیت آن برای علامت آن در نظر گرفته می‌شود. برای مثال unsigned short int دو بایت است و مقادیر ۰ تا ۶۵۵۳۵ را خود نگه‌می‌دارد، اما signed short int مقادیر ۳۲۷۶۷- تا ۳۲۷۶۸ را در خود نگه‌می‌دارد.

## نوع‌های داده‌ای

- یک عدد صحیح را توسط ارقام نشان می‌دهیم. عدد 1234 یک عدد صحیح ثابت است. برای عددهای ثابت صحیح بزرگ از کاراکتر L در پایان عدد استفاده می‌کنیم. برای مثال عدد 123456789L یک عدد ثابت صحیح بزرگ (long int) است.
- اعداد اعشاری را با ممیز اعشاری (برای مثال به صورت 4.123) یا به صورت نمایش علمی (برای مثال به صورت  $1e-2$ ) نشان می‌دهیم.
- برای نمایش اعداد در مبنای ۱۶ از پیشوند 0x استفاده می‌کنیم. برای مثال 0x1F عدد صحیح 1F در مبنای ۱۶ است که معادل عدد ۳۱ در مبنای ۱۰ است.

- کاراکترها را با دو علامت آپوستروف<sup>1</sup> در ابتدا و انتها نشان می‌دهیم. برای مثال 'a' یک کاراکتر شامل حرف a است که مقداری است که در یک بایت ذخیره می‌شود و معادل عددی آن ۹۷ است. مقدار عددی کاراکتر 'A' برابر است با ۶۵.
- برخی کاراکترهای خاص مانند کاراکتر خط جدید '\n' و کاراکتر ستون جدید '\t' نیز وجود دارند. یک کاراکتر را می‌توان با کد اسکی آن نیز به صورت '\xhh' نمایش داد به طوری که hh کد حرف در مبنای شانزده است. برای مثال '\x41' معادل عدد ۶۵ در مبنای ۱۰ است که حرف 'A' را نمایش می‌دهد.

---

<sup>1</sup> apostrophe or single quote



## نوع‌های داده‌ای

- یک متغیر توسط نوع داده‌ای آن تعریف می‌شود. در تعریف متغیر ابتدا نوع و سپس نام متغیر و سپس به طور اختیاری مقدار اولیه آن مشخص می‌شود. برای مثال :

---

```
۱ int lower, upper;  
۲ int i = 0;  
۳ float width = 4.5, height = 15.9;
```

---

- با استفاده از کلمه کلیدی `const` می‌توان یک ثابت تعریف کرد. مقدار یک ثابت در طول اجرای برنامه غیر قابل تغییر است. برای مثال :

---

```
۱ const double e = 2.7182;
```

---

- یک ثابت مانند یک متغیر است پس می‌توان آن را به یک تابع ارسال کرد.

- آرایه برای نگهداری مجموعه‌ای از مقادیر که همه آنها نوع یکسانی دارند به کار می‌رود. هر یک از مقادیر آرایه یک عنصر نامیده می‌شود.
- بنابراین بر خلاف یک متغیر معمولی که یک خانه از حافظه را نامگذاری می‌کند، یک متغیر از نوع آرایه مجموعه‌ای از چند خانه در حافظه را نامگذاری می‌کند.
- برای تعریف یک آرایه نوع عناصر آن، نام آرایه، و تعداد عناصر به صورت زیر تعیین می‌شود.

---

```
۱ int numbers[10];  
۲  
۳ float marks[15];
```

---

## نوع‌های داده‌ای

- مکان هر یک از عناصر در آرایه با یک عدد صحیح مشخص می‌شود که اندیس<sup>1</sup> نامیده می‌شود. اولین اندیس آرایه صفر و آخرین اندیس آرایه یک واحد کمتر از اندازه آرایه است.
- عناصر آرایه را می‌توانیم یک به یک با استفاده از علامت کروشه<sup>1</sup> به صورت زیر مقدار دهی کنیم.

---

```
۱ float marks[5];  
۲ marks[0] = 19;  
۳ marks[1] = 17;  
۴ marks[2] = 15.5;  
۵ marks[3] = 16;  
۶ marks[4] = 17.5;
```

---

---

<sup>1</sup> index

<sup>1</sup> brackets

- عناصر آرایه را می‌توان به صورت زیر نیز مقداردهی اولیه کرد.

---

```
\ float marks[5] = {19, 17, 15.5, 16, 17.5};
```

---

- اگر اندازه یک آرایه در مقداردهی اولیه مشخص نشود، اندازه آرایه با استفاده از تعداد مقادیر اولیه تعیین می‌شود.

---

```
\ float marks[] = {19, 17, 15.5, 16, 17.5};
```

---

- یک ثابت رشته <sup>1</sup> دنباله‌ای است از کاراکترها که با علامت نقل قول تعریف می‌شود. برای مثال "I am a string" یک رشته است.
- یک رشته به صورت آرایه‌ای از کاراکترها تعریف می‌شود. برای مشخص شدن انتهای یک رشته، آخرین کاراکتر یک رشته به طور قراردادی برابر است با کاراکتر '\0' که کاراکتر تهی نامیده می‌شود.
- متغیر name شامل ۱۰ کاراکتر است که در آن رشته سه حرفی "Ali" قرار گرفته است.

---

```
\ char name[10] = "Ali";
```

---

---

<sup>1</sup> string constant

- اگر طول رشته در مقداردهی اولیه تعیین نشود، طول آن برابر با طول رشته اولیه به علاوه یک خواهد بود، زیرا کاراکتر آخر کاراکتر تهی در نظر گرفته می‌شود.
- متغیر name یک آرایه کاراکتری با طول ۴ است و مقدار کاراکتر آخر آن برابر است با '\0'.

---

```
\ char name[] = "Ali";
```

---

- برای استفاده از علامت نقل قول در یک رشته از "\" استفاده می‌کنیم.

---

```
1 char text[] = "One said : \"Truth shall set you free\".";
```

---

- همچنین دو رشته با در کنار یکدیگر قرار گرفتن، به یکدیگر الحاق می‌شوند. برای مثال "hello, world" برابر است با "hello, world". برای تقسیم یک رشته طولانی در چند خط می‌توانیم از این تکنیک استفاده کنیم.

---

```
1 char title[] = "hello, "  
2               "world";
```

---

- برای به دست آوردن طول یک رشته می‌توانیم از تابع زیر استفاده کنیم.

```
۱  /* strlen: return length of s */
۲  int strlen (char s[])
۳  {
۴      int i;
۵      while (s[i] != '\0')
۶          ++i;
۷      return i;
۸  }
```

- تابع strlen در کتابخانه <string.h> تعریف شده است.



– یک نماد ثابت را با `#define` می‌توان تعریف کرد. نماد ثابت در زمان کامپایل جایگزین مقدار ثابت می‌شود. برای مثال می‌نویسیم:

---

```
۱ #define MAX 1000
۲ #define PI 3.1415
۳
۴ char line[MAXLIN];
۵
۶ float area = PI * r * r;
```

---

- نوع داده‌ی شمارشی<sup>1</sup> برای نگهداری مجموعه‌ای از اعداد ثابت تحت عنوان یک نام تعیین شده به کار می‌رود.
- برای مثال نوع داده‌ای شمارشی `boolean` در زیر تعریف شده است و مقدار آن می‌تواند `No` باشد که معادل عدد صحیح صفر است و یا `Yes` باشد که معادل عدد صحیح یک است.

---

```
۱ enum boolean {No, Yes};
```

---

- هر یک از اعضای ثابت در نوع داده‌ی شمارشی، یک ثابت شمارشی<sup>2</sup> نامیده می‌شود که در واقع یک عدد صحیح است.

---

<sup>1</sup> enumeration type

<sup>2</sup> enumeration constant

- در مثال زیر مقدار عددی 0k صفر، مقدار Error یک، و مقدار Invalid برابر است با دو.

---

```
\ enum messages {Ok, Error, Invalid};
```

---

- مقدار متغیر msg از نوع message برابر است با 0k.

---

```
\ enum message msg = 0k;
```

---

- برای ثابت‌های شمارشی می‌توان مقدار نیز تعیین کرد. در مثال زیر مقدار ثابت شمارشی JAN برابر است با یک.

---

```
۱ enum months
۲ { JAN = 1, FEB, MAR, APR, MAY, JUN,
۳   JUL, AUG, SEP, OCT, NOV, DEC
۴ };
۵ /* FEB = 2, MAR = 3, etc. */
```

---

- همچنین برای نامگذاری کاراکترها یا اعداد (جهت سهولت استفاده می‌توان از نوع داده شمارشی استفاده کرد. برای مثال کاراکترهای ویژه در زیر نامگذاری شده‌اند.

```
۱ enum escapes
۲ { BELL = '\a', BACKSPACE = '\b', TAB = '\t',
۳   NEWLINE = '\n', VTAB = '\v', RETURN = '\r'
۴ };
۵
۶ char ch = TAB;
```

- در یک عبارت می‌توان از عملگرهای حسابی<sup>1</sup> مانند + ، - ، \* ، / برای جمع، تفریق، ضرب، و تقسیم، استفاده کرد. برای به دست آوردن باقیمانده از عملگر % استفاده می‌شود.
- عملگرهای یگانی + و - برای تعیین مثبت و منفی بودن اعداد به کار می‌روند.
- عملگرهای یگانی + و - بالاترین اولویت را دارند و پس از آنها \* ، / ، % هم اولویت بوده و در درجهٔ دوم اولویت قرار دارند و در نهایت + و - اولویت سوم قرار می‌گیرند.

---

```

۱ int x, y=2, z=3, w=4;
۲ x = y*-z+w; // x = (y * (-z) ) + w
۳ if (x % 2 == 0)
۴     printf("%d is even.\n", x);
۵ else
۶     printf("%d is even.\n", x);
۷ // -2 is even.
    
```

---



---

<sup>1</sup> arithmetic operators

- عملگرهای رابطه‌ای<sup>1</sup> > ، >= ، < ، <= ، == و != برای مقایسه دو مقدار به کار می‌روند.
- عملگر > مقدار درست را بازمی‌گرداند اگر عملوند اول از عملوند دوم بزرگتر باشد. به همین ترتیب عملگرهای بعدی مقدار درست را بازمی‌گردانند اگر عملوند اول آنها از عملوند دوم بزرگتر یا مساوی، کوچکتر یا مساوی، مساوی، نامساوی باشد.
- برای مثال مقدار متغیر c در کد زیر برابر با صفر یا نادرست است.

```
۱ int x=2, y=5;  
۲ int c;  
۳ c = x > y;  
۴ // c = 0
```

---

<sup>1</sup> relational operators

- همچنین می‌توانیم بنویسیم:

---

```

۱ c = x > y;
۲ if (c)
۳     printf("%d is greater than %d", x, y);
۴ if (x > y)
۵     printf("%d is greater than %d", x, y);

```

---



- عملگرهای رابطه‌ای نسبت به عملگرهای حسابی اولویت کمتری دارند.
- بنابراین در برنامه زیر ابتدا مقدار  $y*z$  محاسبه می‌شود و سپس با  $x$  مقایسه می‌شود.

---

```
۱ if (x < y*z)
۲     printf("%d is less than %d \n", x, y*z);
```

---

- عملگرهای منطقی<sup>2</sup> عطف && و فصل || و نقیض ! نیز در عبارات منطقی به کار می‌روند.
- یک عبارت منطقی عبارتی است که از متغیرهای منطقی و عملگرهای منطقی تشکیل شده و مقدار آن درست یا نادرست است.
- اولویت عملگرهای منطقی از عملگرهای حسابی و رابطه‌ای کمتر است.
- در عبارت زیر عملگر && در پایان همه مقایسه‌ها اعمال می‌شود.

---

```
۱ if (x*y < z && y > x)
۲     printf ("...");
```

---

---

<sup>2</sup> logical operator

- هر عبارت منطقی یا رابطه‌ای دارای مقدار یک است اگر درست باشد و مقدار آن برابر صفر است اگر نادرست باشد.
- عملگر نقیض ! مقدار صفر را به یک و مقدار یک را به صفر تبدیل می‌کند.
- بنابراین به جای `if(valid == 0)` می‌نویسیم `if(!valid)`.

- برای مثال برای مشخص کردن کبیسه بودن سال می‌توانیم از برنامه زیر استفاده کنیم.

```
۱ if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
۲     printf ("%d is a leap year\n", year);
۳ else
۴     printf ("%d is not a leap year\n", year);
```

- عملگر انتساب = برای مقداردهی یک متغیر به کار می‌رود و اولویت آن از عملگرهای حسابی، مقایسه‌ای، و منطقی کمتر است.
- در عبارت زیر ابتدا مقدار `y && z` محاسبه می‌شود، سپس این مقدار در متغیر `x` ذخیره می‌شود. اگر مقدار `x` درست بود، عبارت مربوطه چاپ می‌شود.

---

```

۱ y = 1;
۲ z = 0;
۳ if (x = y && z)
۴     printf("x is true. \n");

```

---

- در عبارت `(c = getchar()) != '\n'` به پرانتزگذاری نیاز داریم، زیرا اولویت `!=` بالاتر از عملگر `=` است.

## عملگرهای افزایش و کاهش

- در زبان سی دو عملگر وجود دارد که در زبان ریاضی وجود ندارند. عملگر افزایش ++ که یک واحد به مقدار یک متغیر می‌افزاید و عملگر کاهش -- که یک واحد از مقدار یک متغیر می‌کاهد.
- این دو عملگر به دو صورت می‌توانند استفاده شوند : به صورت پیشوند <sup>1</sup> (یعنی قبل از متغیر ++n) و به صورت پسوند <sup>2</sup> (یعنی بعد از متغیر n++).
- در هر دو شکل عملگر افزایش یک واحد به مقدار متغیر می‌افزاید با این تفاوت که در حالت پیشوند افزایش قبل از استفاده از متغیر اعمال می‌شود و در حالت پسوند افزایش بعد از استفاده متغیر اعمال می‌شود.

---

<sup>1</sup> prefix

<sup>2</sup> postfix

## عملگرهای افزایش و کاهش

- برای مثال اگر مقدار  $n$  برابر با ۵ باشد، آنگاه پس از عبارت  $x = n++$  مقدار  $x$  برابر با ۵ خواهد بود چرا که ابتدا  $n$  با مقدار قبلی در عبارت استفاده می‌شود و سپس یک واحد به آن افزوده می‌شود. اما پس از عبارت  $x = ++n$  مقدار  $x$  برابر با ۶ خواهد بود چرا که ابتدا یک واحد به  $n$  افزوده می‌شود و سپس  $n$  با مقدار جدید در عبارت استفاده می‌شود.
- این عملگرها تنها بر روی تک متغیر اعمال می‌شوند، بنابراین  $++(i+j)$  یک عبارت غیر معتبر و بی معنا است.
- در برخی موارد هدف تنها افزایش یا کاهش یک متغیر است و در این موارد پیشوند و پسوند تفاوتی ندارند. اما وقتی این عملگرها در یک عبارت استفاده شوند، تفاوت آنها مشهود است. همچنین با استفاده از این عملگرها می‌توان برنامه را مختصرتر نوشت.

- در مثال زیر که همه حروف c را از رشته s حذف می‌کند از عملگر افزایش استفاده شده است.

```
۱ /* squeeze: delete all c from s */  
۲ void  
۳ squeeze (char s[], int c)  
۴ {  
۵     int i, j;  
۶     for (i = j = 0; s[i] != '\0'; i++)  
۷         if (s[i] != c)  
۸             s[j++] = s[i];  
۹     s[j] = '\0';  
۱۰ }
```



- اگر عملگر ++ وجود نداشت، مجبور بودیم بدنه شرط را به صورت زیر بنویسیم.

---

```
۱  if (s[i] != c ) {  
۲      s[j] = s[i];  
۳      j = j + 1;  
۴  }
```

---

## عملگرهای افزایش و کاهش

- حال تابع strcat را در نظر بگیرید که رشته t را در انتها رشته s الحاق می‌کند.

```
۱  /* strcat: concatenate t to end of s;  
۲   * s must be big enough */  
۳  void  
۴  strcat (char s[], char t[])  
۵  {  
۶      int i, j;  
۷      i = j = 0;  
۸      while (s[i] != '\0')      /* find end of s */  
۹          i++;  
۱۰     while ((s[i++] = t[j++]) != '\0')^I/* copy t */  
۱۱         ;  
۱۲ }
```

- پس از هر بار کپی کردن یک کاراکتر از رشته t به رشته s ، مقدار اندیس i و j یک واحد افزایش پیدا می‌کند.

- در زبان سی تعدادی عملگر بیتی وجود دارد که بر روی عملوندهای صحیح مانند char ، short ، int و long اعمال می‌شوند. عملگرهای بیتی دوگانی<sup>1</sup> بر روی بیت‌های دو عملوند دریافتی خود عملیات بیتی شامل عطف<sup>2</sup> و فصل<sup>3</sup> و فصل انحصاری<sup>4</sup> و انتقال به چپ<sup>5</sup> و انتقال به راست<sup>6</sup> انجام می‌دهند.

---

<sup>1</sup> binary bitwise operators

<sup>2</sup> conjunction

<sup>3</sup> disjunction

<sup>4</sup> exclusive disjunction

<sup>5</sup> left shift

<sup>6</sup> right shift

## عملگرهای بیتی

- عملگر بیتی عطف یا AND با & ، عملگر بیتی فصل یا OR با | ، عملگر بیتی فصل انحصاری یا XOR با ^ ، عملگر انتقال به چپ با << و عملگر انتقال به راست با >> نشان داده می‌شوند.
- همچنین عملگر یگانی<sup>1</sup> مکمل با ~ نشان داده می‌شود.
- برای مثال :

```
۱ int x = 5; // binary 101
۲ int y = 6; // binary 110
۳ int z = x & y; // 101 & 110 = 100 // z = 4
۴ int w = x | y; // 101 | 110 = 111 // w = 7
۵ int n = x << 3 ; 101 << 3 = 101000 // n = 40
۶ int m = ~x + 1 // m = -5
```

---

<sup>1</sup> unary bitwise operator

- عبارت  $x = x \mid \text{SET\_ON}$  همه بیت‌هایی که در  $x$  صفر هستند و در  $\text{SET\_ON}$  یک هستند را به یک تبدیل می‌کند.
- توجه داشته باشد که عملگر منطقی  $\&\&$  با عملگر بیتی  $\&$  متفاوت است. اگر  $x$  برابر با ۱ و  $y$  برابر با ۲ باشد، مقدار  $x \& y$  برابر با صفر است، در حالی که مقدار  $x \&\& y$  برابر با مقدار درست یا یک است.

- عملگر انتقال  $\ll$  و  $\gg$  انتقال به چپ و راست انجام می‌دهند. در سمت چپ عملگر متغیری قرار می‌گیرد که عملیات انتقال برای آن انجام می‌شود و در سمت راست عملگر تعداد انتقال‌ها قرار می‌گیرد.
- برای مثال  $2 \ll x$  مقدار  $x$  را دو واحد به سمت چپ انتقال می‌دهد و مقدار دو بیت کم ارزش را برابر با صفر قرار می‌دهد. این عملیات معادل ضرب عدد در ۴ است.
- عملگر یگانی بیتی مکمل  $\sim$  بیت‌های صفر را به یک و بیت‌های یک را به صفر تبدیل می‌کند.

- عبارت  $i = i + 2$  را می‌توانیم به صورت  $i += 2$  نیز بنویسیم. عملگر  $+=$  یک عملگر انتساب<sup>1</sup> نامیده می‌شود.
- همه عملگرهای  $op=$  در زبان سی تعریف شده‌اند به طوری که مقدار  $op$  می‌تواند  $+$ ،  $-$ ،  $*$ ،  $/$ ،  $\%$ ،  $<<$ ،  $>>$ ،  $\&$ ،  $|$ ،  $\wedge$  باشد
- برای مثال  $x *= y+1$  معادل است با  $x = x * (y+1)$ .

---

<sup>1</sup> assignment operator

- در برنامه زیر تعداد بیت‌های یک در متغیر x شمرده می‌شود.

```
۱  /* bitcount: count 1 bits in x */  
۲  int  
۳  bitcount (unsigned x)  
۴  {  
۵      int b;  
۶      for (b = 0; x != 0; x >>= 1)  
۷          if (x & 01)  
۸              b++;  
۹      return b;  
۱۰ }
```



- علاوه بر این که با استفاده از عملگرهای انتساب برنامه را می‌توان به صورت مختصر نوشت، در برخی موارد باعث خوانایی بیشتر برنامه نیز می‌شود. برای مثال عبارت  
`yyval [yyvsp[p3+p4] + yyvp[p1]] += 2` را در نظر بگیرید. علاوه بر این که با استفاده از عملگر `+=` برنامه کوتاه‌تر می‌شود، اگر عبارت را به صورت عادی با استفاده از عملگر تساوی می‌نوشتیم، خواننده برنامه مجبور بود بررسی کند آیا در سمت چپ و راست عملگر تساوی دو عبارت یکسان‌اند یا خیر و اگر دو عبارت یکسان نبودند نمی‌توانست مطمئن باشد که آیا برنامه‌نویس خطایی در نوشتن برنامه انجام داده یا اینکه در عبارت یکسان نیستند.
- یک عبارت انتساب دارای یک مقدار است. مقدار یک عبارت، برابر است با مقدار سمت چپ عبارت انتساب بعد از عملیات و نوع آن برابر با نوع متغیر سمت چپ عبارت.

– قطعه برنامه زیر را در نظر بگیرید.

```
۱  if ( a > b )  
۲      z = a;  
۳  else  
۴      z = b;
```

– این قطعه برنامه ماکزیمم بین دو متغیر را محاسبه می‌کند. این عبارت را به طور مختصرتر می‌توان با استفاده از عملگر  $?:$  به صورت زیر نوشت.

```
۱  z = (a > b) ? a : b;      /* z = max(a,b) */
```

- در واقع مقدار عبارت  $\text{expr2} : \text{expr1} ? \text{cond}$  برابر است با  $\text{expr1}$  اگر  $\text{cond}$  درست باشد و برابر است با  $\text{expr2}$  اگر  $\text{cond}$  نادرست باشد.
- اگر نوع  $\text{expr1}$  و  $\text{expr2}$  متفاوت باشد، نوع عبارت شرطی برابر با نوعی است که وسیع‌تر باشد. برای مثال نوع عبارت  $i : f ? (n > 0)$  برابر است با  $\text{float}$  جایی که  $f$  یک  $\text{float}$  است و  $i$  یک  $\text{int}$ .

- در برنامه زیر کاراکتر خط جدید هر ۱۰ خط یک بار چاپ می‌شود و همچنین وقتی آخرین عنصر آرایه چاپ می‌شود.

---

```
۱ for (i = 0; i < n; i++)  
۲     printf("%6d%c", a[i], (i%10==9 || i==n-1) ? '\n' : ' ');
```

---

- در مثال زیر اگر یک عنصر وجود داشت می‌خواهیم از کلمه مفرد استفاده کنیم و اگر چند عنصر وجود داشت از کلمه جمع.

---

```
۱ printf("You have %d item%s.\n", n, n==1 ? "" : "s");
```

---

- وقتی یک عملگر دارای چند عملوند از نوع‌های مختلف است، نوع عملوند با اندازه کوچک‌تر به نوع عملوند با اندازه بزرگ‌تر تبدیل می‌شود. بدین ترتیب هیچ اطلاعاتی را بین نمی‌رود. برای مثال در عبارت  $f + i$  که جمع یک عدد اعشاری و یک عدد صحیح است، عدد صحیح به اعشاری تبدیل می‌شود.
- در جایی که تبدیل نوع ممکن است به از دست رفتن اطلاعات منجر شود. کامپایلر پیام خطا صادر می‌کند. برای مثال در انتساب یک عدد صحیح طولانی به یک عدد صحیح ممکن است اطلاعات از بین برود.
- طول `char` از `int` کمتر است، پس می‌تواند بدون خطا تبدیل کاراکتر به عدد صحیح انجام شود.

- در مثال زیر یک رشته به معادل عددی آن تبدیل می‌شود.

```
۱  /* atoi: convert s to integer */
۲  int
۳  atoi (char s[])
۴  {
۵      int i, n;
۶      n = 0;
۷      for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
۸          n = 10 * n + (s[i] - '0');
۹      return n;
۱۰ }
```

## تبدیل نوع

- در مثال قبل `s[i] - '0'` مقدار عددی کاراکتر را مشخص می‌کند. معادل اسکی کاراکتر `'0'` برابر است با ۴۸ و معادل اسکی کاراکترهای `'1'` ، `'2'` و ... به ترتیب برابر است با ۴۹ ، ۵۰ و ... بنابراین تفاضل محاسبه شده معادل عددی یک کاراکتر را تعیین می‌کند.
- به عنوان یک مثال دیگر، در برنامه زیر یک حروف بزرگ به حروف کوچک تبدیل می‌شود.

```
۱  /* lower: convert c to lower case; ASCII only */
۲  int
۳  lower (int c)
۴  {
۵      if (c >= 'A' && c <= 'Z')
۶          return c + 'a' - 'A';
۷      else
۸          return c;
۹  }
```

– در کتابخانه `<ctype.h>` بسیاری از توابع کاربردی برای تبدیل کاراکترها مانند توابع `tolower` یا `isdigit` پیاده‌سازی شده‌اند.



- یک عبارت منطقی در صورتی که درست باشد مقدار یک و در غیراینصورت مقدار صفر را بازمی‌گرداند.  
بنابراین مقدار متغیر در عبارت `d = c >= '0' && c <= '9'` برابر با یک است اگر `c` یک رقم باشد و در غیراینصورت مقدار آن برابر با صفر است.
- در هنگام استفاده از اعداد در عبارت منطقی هر مقدار غیر صفر درست<sup>1</sup> است و مقدار صفر برابر با نادرست<sup>2</sup> است.

---

<sup>1</sup> true

<sup>2</sup> false

- در یک عبارت اگر یکی از عملوندها long double باشد بقیه عملوندها نیز به این نوع تبدیل می‌شوند در غیراینصورت اگر یکی از عملوندها double باشد بقیه عملوندها به double تبدیل می‌شوند. در غیراینصورت اگر یکی از عملوندها float باشد، بقیه عملوندها به float تبدیل می‌شوند. در غیراینصورت اگر یکی از عملوندها int باشد، بقیه عملوندها به int تبدیل می‌شوند.
- اگر در یک عملیات انتساب یک متغیر از نوع char را برابر با یک متغیر از نوع int قرار دهیم، بیت‌های پرارزش‌تر از بین می‌روند.
- در تبدیل float به int قسمت اعشاری از بین می‌رود. در تبدیل double به float رقم‌های اعشار با تقریب کاهش می‌یابند.
- تبدیل نوع در ارسال آرگومان‌ها به پارامترها نیز صورت می‌گیرد اگر نوع‌ها همخوانی نداشته باشند.

- یک نوع را می‌توان به صورت صریح نیز به یک نوع دیگر تبدیل کرد. در عبارت `expression (type name)` نوع عبارت به طور صریح تبدیل می‌شود. به این پرانتزگذاری و تعیین نوع عملگر تبدیل نوع<sup>1</sup> می‌گوییم.
- برای مثال برای تبدیل عدد `n` به `double` در تابع `sqrt` که جذر یک عدد را محاسبه می‌کند می‌نویسیم `.sqrt((double)n)`

---

<sup>1</sup> cast operator

## اولویت عملگرها

- در جدول زیر قوانین اولویت ذکر شده‌اند. عملگرهایی که در سطرها بالتر قرار دارند، اولویت بالاتری دارند. عملگرهایی که در یک سطر هستند اولویت یکسان دارند.

Operators	Associativity
() [] -> .	left to right
! ~ ++ -- + - * (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %= &= ^=  = <<= >>=	right to left
,	left to right

- برای مثال در عبارت ( 0 == (x & MASK) ) if اگر از پرانتز گذاری استفاده نکنیم، معنای عبارت متفاوت خواهد بود.

- همچنین باید توجه داشت که استاندارد زبان سی مشخص نمی‌کند که در یک عبارت کدام یک از پارامترها زودتر محاسبه می‌شوند، بنابراین در پیاده‌سازی‌های متفاوت زبان سی و کامپایلرهای متفاوت نتیجه یک عبارت متفاوت باشد.
- برای مثال در عبارت زیر مشخص نیست آیا ابتدا  $++n$  محاسبه می‌شود و یا مقدار  $\text{power}(2,n)$  و بنابراین بهتر است برنامه‌نویس سی از نوشتن چنین عبارت‌هایی پرهیز کند.

---

```
\ printf("%d %d \n" , ++n, power(2,n));    /* WRONG */
```

---