به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

عبارات و نوع‌های داده‌ای

نوع‌های داده‌ای

- یک متغیر [1] 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 [2] برای انجام عملیات محاسباتی و منطقی برروی متغیرها استفاده می‌شود.
- یک عبارت [3] با ترکیب مقادیر عددی و غیرعددی، متغیرها و ثابت‌ها [4] به عنوان عملوند [5] 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.

---

[1] variable
[2] operator
[3] expression
[4] constants
[5] operand

نوع‌های داده‌ای

- نام متغیرها می‌تواند از حروف الفبا و ارقام و زیرخط [1] (_) 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 معمولا از حروف کوچک برای نام متغیرها استفاده می‌کنند و نام نمادهای ثابت معمولا 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.

[1] underscore

نوع‌های داده‌ای

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

[1] character
[2] integer
[3] floation point
[4] double-precision float point

نوع‌های داده‌ای

- 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.

نوع‌های داده‌ای

- کاراکترها را با دو علامت آپوستروف [1] 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 طوری‌که hh کد حرف در مبنای شانزده است. برای مثال `'\x41'` معادل عدد ۶۵ در مبنای ۱۰ است که حرف `'A'` را نمایش می‌دهد.

[1] 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 :

```
int lower, upper;
int i = 0;
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 :

```
const double e = 2.7182;
```

نوع‌های داده‌ای

- آرایه برای نگهداری مجموعه‌ای از مقادیر که همهٔ 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.

```
1 int numbers[10];
2
3 float marks[15];
```

نوع‌های داده‌ای

- مکان هر یک از عناصر در آرایه با یک عدد صحیح مشخص می‌شود که اندیس [1] نامیده می‌شود. اولین اندیس آرایه صفر و آخرین اندیس آرایه یک واحد کمتر از اندازهٔ آرایه است.
- عناصر آرایه را می‌توانیم یک به یک با استفاده از علامت کروشه [1] 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;
```

[1] index
[1] brackets

نوع‌های داده‌ای

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

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

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

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

نوع‌های داده‌ای

- یک ثابت رشته [1] 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 برابراست با کاراکتر `'\0'` که کاراکتر تهی نامیده می‌شود.
- متغیر `name` شامل ۱۰ کاراکتر است که در آن رشتهٔ سه حرفی `"Ali"` قرار گرفته است.

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

[1] string constant

نوع‌های داده‌ای

- اگر طول رشته در مقداردهی اولیه تعیین نشود، طول آن برابر با طول رشتهٔ اولیه به علاوه یک خواهد بود، زیر 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' .

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

نوع‌های داده‌ای

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

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

- همچنین دو رشته با درکنار یکدیگر قرارگرفتن، به یکدیگر الحاق می‌شوند. برای مثال "hello, "world" برابراست با "hello, world" . برای تقسیم یک رشته طولانی در چند خط می‌توانیم از این تکنیک استفاده کنیم.

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

نوع‌های داده‌ای

- 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.

نوع‌های داده‌ای

- 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[MAX];
۵
۶ float area = PI * r * r;
```

نوع‌های داده‌ای

- نوع دادهٔ شمارشی [1] 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 دادهٔ شمارشی، یک ثابت شمارشی [2] نامیده می‌شود که در واقع یک عدد صحیح است.

[1] enumeration type
[2] enumeration constant

نوع‌های داده‌ای

- در مثال زیر مقدار عددی Ok 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 Ok.

```
enum message msg = Ok;
```

نوع‌های داده‌ای

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

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

نوع‌های داده‌ای

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

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

char ch = TAB;
```

عملگرها

- در یک عبارت می‌توان از عملگرهای حسابی [1] 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.

```
1 int x, y=2, z=3, w=4;
2 x = y*-z+w; // x = (y * (-z) ) + w
3 if (x % 2 == 0)
4     printf("%d is even.\n", x);
5 else
6     printf("%d is odd.\n", x);
7 // -2 is even.
```

[1] arithmetic operators

عملگرها

- عملگرهای رابطه‌ای [1] < ، >= ، > ، <= ، == 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، کوچکتر یا مساوی، مساوی، نامساوی باشد.
- برای مثال مقدار متغیر c درکد زیر برابر با صفر یا نادرست است.

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

[1] relational operators

عملگرها

- 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);
```

عملگرها

- 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);
```

عملگرها

- عملگرهای منطقی [2] 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 همهٔ مقایسه‌ها اعمال می‌شود.

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

[2] logical operator

عملگرها

- 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)`.

عملگرها

- 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);
```

عملگرها

- 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 z && y 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 تغییر می‌کاهد.
- این دو عملگر به دو صورت می‌توانند استفاده شوند : به صورت پیشوند [1] (یعنی قبل از متغیر n++) و به صورت پسوند [2] (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.

[1] prefix
[2] 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 چراکه 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 چراکه ابتدا یک واحد به `n` افزوده می‌شود و سپس `n` با مقدار جدید در عبارت استفاده می‌شود.
- این عملگرها تنها برروی 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.

عملگرهای افزایش و کاهش *

- در مثال زیر که همهٔ 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 }
```

عملگرهای افزایش و کاهش *

- اگر عملگر ++ وجود نداشت، مجبور بودیم بدنهٔ 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')   /* 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.

عملگرهای بیتی

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

[1] binary bitwise operators
[2] conjunction
[3] disjunction
[4] exclusive disjunction
[5] left shift
[6] 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 [1] مکمل با ~ نشان داده می‌شود.

[1] unary bitwise operator

عملگرهای بیتی

- خروجی عملگر بیتی عطف «و» & بین دو عملوند x و y عددی است که مقدار بیت i ام آن یک است اگر بیت i ام x و y هر دو یک باشد.
- خروجی عملگر بیتی فصل «یا» | بین دو عملوند x و y عددی است که مقدار بیت i ام آن یک است اگر بیت i ام x یا y یک باشد.
- خروجی عملگر بیتی فصل انحصاری «یای انحصاری» ^ بین دو عملوند x و y عددی است که مقدار بیت i ام آن یک است اگر بیت i ام فقط یکی از عملوندهای x یا y یک باشد.

عملگرهای بیتی

- عملگر انتقال به چپ << بیت‌های عملوند اول خود را به تعداد عدد عملوند دوم به چپ انتقال می‌دهد. برای مثال `x << n` مقدار `x` را `n` واحد به سمت چپ انتقال می‌دهد و مقدار `n` بیت سمت راست (کم ارزش) را برابر با صفر قرار می‌دهد. این عملیات معادل ضرب عدد `x` در ۲^n است.
- عملگر انتقال به راست >> بیت‌های عملوند اول خود را به تعداد عدد عملوند دوم به راست انتقال می‌دهد. برای مثال `x >> n` مقدار `x` را `n` واحد به سمت راست انتقال می‌دهد و مقدار `n` بیت سمت راست (کم ارزش) را حذف می‌کند. این عملیات معادل تقسیم عدد `x` بر ۲^n است.
- عملگر یگانی مکمل ~ بیت‌های صفر را به یک و بیت‌های یک را به صفر تبدیل می‌کند.

عملگرهای بیتی

- برای مثال :

```
x = 5; // x = 101
y = 6; // y = 110
z = x & y; // z = 101 & 110 = 100 = 4
w = x | y; // w = 101 | 110 = 111 = 7
u = x ^ y; // u = 101 ^ 110 = 011 = 3
s = x << 3; // s = 101 << 3 = 101000 = 40
t = y >> 1; // t = 110 >> 1 = 11 = 3
n = ~x + 1 // n = 11111010 + 1 =  -6 + 1 = -5
```

عملگرهای بیتی

- فرض کنید در یک برنامه، n عددی است دو بایتی که قطع و وصل بودن (صفر و یک بودن) ۱۶ پین را تعیین می‌کند. حال می‌خواهیم به غیر از ۷ پین اول از سمت راست، بقیهٔ پین‌ها را قطع کنیم.
- در عبارت `n = n & 0x007F` ، عدد 0x007F معادل عدد دودویی 0000000001111111 است. بنابراین همهٔ بیت‌های عدد n بعد از ۷ بیت سمت راست برابر با صفر قرار می‌گیرند.
- حال فرض کنید می‌خواهیم ۳ پین اول از سمت راست را در صورتی که وصل نیستند، وصل کنیم.
- عبارت `n = n | SET_ON` همهٔ بیت‌هایی که در n صفر هستند و در SET_ON یک هستند را به یک تبدیل می‌کند. حال برای تبدیل به یک کردن سه بیت اول قرار می‌دهیم: `SET_ON = 0b111` .
- 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.
- بنابراین عملگرهای منطقی بر روی متغیرهای منطقی (متغیرهایی که مقدار آنها درست یا نادرست است) اعمال می‌شوند، در صورتی که عملگرهای بیتی بر روی اعداد صحیح اعمال می‌شوند و روی بیت‌های آنها یک به یک عمل می‌کنند.

عملگرهای انتساب

- عبارت `i = i + 2` را می‌توانیم به صورت `i += 2` نیز بنویسیم. عملگر `+=` یک عملگر انتساب [1] نامیده می‌شود.
- همهٔ عملگرهای `op=` در زبان سی تعریف شده‌اند به طوری‌که مقدار `op` می‌تواند `+` ، `-` ، `*` ، `/` ، `%` ، `>>` ، `<<` ، `&` ، `|` ، `^` باشد
- برای مثال `x *= y+1` معادل است با `x = x * (y+1)`.

[1] assignment operator

عملگرهای انتساب

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

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

عملگرهای انتساب

- 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 [yypv[p3+p4] + yypv[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، خوانندهٔ 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 دو عبارت نباید یکسان باشند.

- 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. برای مثال برنامه زیر مقدار محاسبه شده برای x را با y مقایسه می‌کند.

```
1 if ( (x = i + j) == y)
2       ...
```

عبارات شرطی

- 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)  */
```

عبارات شرطی

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

عبارات شرطی

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

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

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

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

تبدیل نوع

- وقتی یک عملگر دارای دو عملوند از نوع‌های مختلف است، نوع عملوند با اندازهٔ کوچک‌تر (عملوندی که فضای کمتری در حافظه اشغال می‌کند) به نوع عملوند با اندازهٔ 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 اخطار صادر می‌کند. برای مثال در انتساب یک عدد صحیح به یک کاراکتر ممکن است اطلاعات از بین برود.
- طول `char` از `int` کمتر است، پس تبدیل کاراکتر به عدد صحیح می‌تواند بدون خطا انجام شود.

تبدیل نوع

- 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

- یک نوع را می‌توان به صورت صریح نیز به یک نوع دیگر تبدیل کرد. در عبارت `(type-name)expression` نوع عبارت به طور صریح تبدیل می‌شود. به این عملگر، عملگر تبدیل نوع [1] می‌گوییم.
- در مثال زیر به طور صریح یک عدد اعشاری به صحیح تبدیل می‌شود:

```
۱ float f = 2.5;
۲ int n = (int)f;
```

[1] cast operator

تبدیل نوع *

- اگر یک رشته شامل عدد صحیح باشد، نمی‌توان با عملگر تبدیل نوع آن را به عدد صحیح تبدیل کرد، اما می‌توان برنامه‌ای به صورت زیر نوشت که محتوای یک رشته به معادل عددی آن تبدیل می‌کند.

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

تبدیل نوع *

- در مثال قبل `s[i]-'0'` مقدار عددی کاراکتر را مشخص می‌کند. معادل اسکی کاراکتر `'0'` برابراست با ۴۸ و معادل اسکی کاراکترهای `'1'` ، `'2'` و ... به ترتیب برابراست با ۴۹ ، ۵۰ و ... . بنابراین تفاضل محاسبه شده معادل عددی یک کاراکتر را تعیین می‌کند.
- یک عبارت منطقی در صورتی که درست باشد مقدار یک و در غیراینصورت مقدار صفر را بازمی‌گرداند. بنابراین مقدار متغیر `d` 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 عبارات منطقی هر مقدار غیر صفر معادل درست [1] و مقدار صفر برابر با نادرست [2] است.

[1] true
[2] false

اولویت عملگرها

- در جدول زیر قوانین اولویت ذکر شده‌اند. عملگرهایی که در سطرهای بالاتر 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

اولویت عملگرها

- برای مثال در عبارت `if((x & MASK) == 0)` اگر از پرانتزگذاری استفاده نکنیم، معنای عبارت متفاوت خواهد بود، زیرا اولویت == از اولویت & بیشتر است.
- همچنین باید توجه داشت که استاندارد زبان سی مشخص نمی‌کند که در فراخوانی تابع 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 `power(2,n)` و بنابراین بهتر است برنامه‌نویس سی از نوشتن چنین عبارت‌هایی پرهیز کند.

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