به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

اشاره‌گرها و آرایه‌ها

اشاره‌گرها

- یک اشاره‌گر [1] متغیری است که آدرس یک متغیر را ذخیره می‌کند.
- اشاره‌گرها به کثرت در زبان سی استفاده می‌شوند. گاهی استفاده از اشاره‌گرها بدین دلیل است که برنامه‌نویسی را ساده‌تر می‌کنند و بدون آنها توصیف محاسبات پیچیده می‌شود و گاهی بدین دلیل است که با استفاده از آنها می‌توان برنامه‌های فشرده‌تر و کارامدتری نوشت.
- اشاره‌گرها و آرایه بسیار به یکدیگر شبیه هستند و در این قسمت به معرفی اشاره‌گرها و بررسی آنها در کنار آرایه‌ها خواهیم پرداخت.

[1] pointer

اشاره‌گرها

- یک سیستم کامپیوتری معمولاً دارای آرایه‌ای از مکان‌های حافظه است که به صورت پی‌درپی و شماره‌گذاری شده به دنبال یکدیگر قرار گرفته شده‌اند.
- مقدار یک متغیر از نوع `char` در یک بایت از این مکان‌های حافظه قرار می‌گیرد. مقدار یک متغیر از نوع `short` در دو بایت پی‌درپی در حافظه ذخیره می‌شود.
- یک اشاره‌گر یک متغیر ۴ یا ۸ بایتی (بسته به نوع ماشین) است که آدرس یک مکان حافظه را نگهداری می‌کند.
- بنابراین اگر `c` یک متغیر از نوع `char` باشد و `p` یک متغیر از نوع اشاره‌گر باشد، متغیر `p` می‌تواند آدرس متغیر `c` در حافظه را نگهداری کند.

اشاره‌گرها

- آدرس یک متغیر را می‌توانیم با عملگر امپرسند & دریافت کنیم، بنابراین عبارت `p = &c` آدرس متغیر `c` را دریافت می‌کند و این آدرس را در اشاره‌گر `p` ذخیره می‌کند. می‌گوییم اشاره‌گر `p` به متغیر `c` اشاره می‌کند.
- عملگر & تنها بر روی متغیرهایی که بر روی حافظه هستند عمل می‌کند و برروی عبارات و نمادهای ثابت و متغیرهای رجیستر نمی‌توان آن را اعمال کرد.
- عملگر ستاره * یک عملگر رفع ارجاع [1] است. وقتی این عملگر بر روی یک اشاره‌گر اعمال می‌شود، مقدار متغیری را که اشاره‌گر به آن اشاره می‌کند بازمی‌گرداند.
- فرض کنید `x` و `y` دو متغیر از نوع عدد صحیح هستند و `ip` یک اشاره‌گر از نوع عدد صحیح (متغیر از نوع اشاره‌گر به عدد صحیح) است.

[1] dereference

اشاره‌گرها

- در عبارات زیر نشان داده شده است چگونه از اشاره‌گرها استفاده می‌کنیم.

```
۱ int x = 1, y = 2, z[10];
۲ int *ip; /* ip is a pointer to int */
۳ ip = &x; /* ip now points to x */
۴ y = *ip; /* y is now 1 */
۵ *ip = 0; /* x is now 0 */
۶ ip = &z[0]; /* ip now points to z[0] */
```

شاره‌گرها

- در تعریف اشاره‌گر ip می‌گوییم *ip یک عدد صحیح است، یعنی مقداری که ip به آن اشاره می‌کند یک عدد صحیح است و بنابراین ip یک اشاره‌گر است به یک متغیر از نوع عدد صحیح.
- در عبارت زیر در واقع می‌گوییم *dp و تابع atof از نوع double هستند و تابع atof یک اشاره‌گر از نوع کاراکتر دریافت می‌کند. در واقع dp یک اشاره‌گر به یک عدد double است و تابع atof مقداری از نوع double بازمی‌گرداند.

```
1 double *dp, atof(char *);
```

- هر اشاره‌گر به یک متغیر از نوع معین اشاره می‌کند. البته یک استثنا برای اشاره‌گر به نوع void وجود دارد که در مورد آن بعدها بیشتر صحبت خواهیم کرد.

اشاره‌گرها

- فرض کنید `ip` اشاره‌گری است که به متغیر `x` اشاره می‌کند. در این‌صورت عبارت `*ip = *ip + 10` مقدار متغیر `x` را ۱۰ واحد افزایش می‌دهد. عبارت `y = *ip + 1` مقدار متغیری که `ip` به آن اشاره می‌کند را دریافت می‌کند و پس از افزودن یک واحد مقدار به دست آمده را در متغیر `y` ذخیره می‌کند. عبارت `*ip += 1` مقدار متغیری که `ip` به آن اشاره می‌کند را یک واحد افزایش می‌دهد. که معادل است با `*ip++` یا `(*ip)++`.

- پرانتزگذاری در عبارت `(*ip)++` ضروری است، زیرا وابستگی [1] عملگر `++` و `*` از راست به چپ است. بنابراین در عبارت `*ip++` ابتدا مقدار `ip` یک واحد افزایش می‌یابد (به خانه حافظه بعدی اشاره می‌کند) و سپس مقدار آن دریافت می‌شود.

- همچنین می‌توانیم مقدار یک اشاره‌گر (که یک آدرس است) را در یک اشاره‌گر دیگر ذخیره کنیم برای مثال اگر `ip` و `iq` دو اشاره‌گر باشند، می‌توانیم بنویسیم `iq = ip` که در اینصورت `iq` به متغیری اشاره خواهد کرد که `ip` نیز به آن اشاره می‌کند.

[1] associativity

اشاره‌گرها به عنوان پارامتر تابع

- از آنجایی که در زبان سی آرگومان‌ها با مقدار به توابع ارسال می‌شوند، یک تابع نمی‌تواند مقدار یک آرگومان را تغییر دهد.
- برای مثال فرض کنید در یک تابع می‌خواهیم مقدار دو آرگومان را توسط تابع swap جابجا کنیم. این جابجایی ممکن است در یک الگوریتم مرتب‌سازی برای جابجایی دو عنصر مورد استفاده قرار بگیرد.
- حال فرض کنید تابع swap را به صورت زیر بنویسیم.

```
۱ void swap(int x, int y)   /* WRONG */
۲ {
۳   int temp;
۴   temp = x;
۵   x = y;
۶   y = temp;
۷ }
```

اشاره‌گرها به عنوان پارامتر تابع

- چون فراخوانی توابع با مقدار است، تابع `swap` مقدار دو متغیر را جابجا نمی‌کند، بلکه تنها مقدار دو متغیر پارامتر خود را جابجا می‌کند که در مقدار آرگومان‌ها تأثیری نمی‌گذارد، زیرا مقدار آرگومان تابع `swap` در مقدار پارامترهای `x` و `y` کپی می‌شود.
- به عبارت دیگر با فراخوانی تابع `swap(a,b)` در واقع مقدار `a` در `x` و مقدار `b` در `y` کپی می‌شود و تغییر متغیرهای `x` و `y` در مقدار متغیرهای `a` و `b` بی تأثیر است.
- برای اینکه تابع بتواند مقدار آرگومان‌ها را تغییر دهد، باید اشاره‌گری از آرگومان‌ها را به تابع ارسال کنیم تا تابع با در دست داشتن آدرس آرگومان‌ها بتواند مقدار آنها را تغییر دهد پس تابع `swap` باید دو پارامتر اشاره‌گر دریافت کند.

اشاره‌گرها به عنوان پارامتر تابع

- تابع زیر مقدار آرگومان‌هایی که با تابع ارسال می‌شوند را جابجا می‌کند.

```
۱ void swap(int *px, int *py) /* interchange *px and *py */
۲ {
۳   int temp;
۴   temp = *px;
۵   *px = *py;
۶   *py = temp;
۷ }
```

- حال می‌توانیم تابع swap(&a,&b) را فراخوانی کنیم. در این صورت اشاره‌گر px آدرس a و اشاره‌گر py آدرس را نگهداری خواهد کرد و بنابراین می‌تواند با استفاده از آدرس آنها به مقدارشان دسترسی پیدا کرده و مقادیر آنها را جابجا کند.

اشاره‌گرها به عنوان پارامتر تابع

- پارامترهایی که از نوع اشاره‌گر هستند توابع را قادر می‌سازند که آرگومان‌هایی که به آنها ارسال می‌شوند را تغییر دهند.

اشاره‌گرها به عنوان پارامتر تابع

- فرض کنید می‌خواهیم تابعی به نام `getint` بنویسیم که در هر بار فراخوانی یک عدد صحیح را از ورودی استاندارد دریافت می‌کند.
- برای دریافت یک عدد ترتیب ارقام دریافت شده تا وقتی که ورودی یک رقم نباشد. این ارقام تبدیل به عدد می‌شوند و در اشاره‌گر ورودی ذخیره می‌شوند. در صورتی که در ورودی کاراکتر پایان فایل دریافت شود، تابع این کاراکتر را باز می‌گرداند.
- درواقع در طراحی این تابع می‌توانیم به صورتی دیگر عمل کنیم بدین صورت که تابع `getint` مقدار عدد دریافت شده از ورودی را بازگرداند، اما در این صورت چگونه به فراخوانی کنندهٔ `getint` بگوییم عدد دیگری وجود ندارد ؟ درواقع در این تابع نیاز داریم دو مقدار را به فراخوانی کنندهٔ تابع بازگردانیم : مقدار عدد خوانده شده و کاراکتر پایان فایل در صورتی که به پایان فایل برسیم.
- در چنین مواقعی معمولاً مقادیر کنترلی را در خروجی تابع باز می‌گردانیم و مقادیر مورد نیاز را در پارامترهای اشاره‌گر در ورودی ذخیره می‌کنیم.

اشاره‌گرها به عنوان پارامتر تابع

- بنابراین از تابع `getint` می‌توانیم به صورت زیر برای دریافت آرایه‌ای از اعداد استفاده کنیم.

```
۱ int n, array[SIZE], getint(int *);
۲ for (n = 0; n < SIZE && getint(&array[n]) != EOF; n++);
```

- در هر یک از عناصر `array[n]` عدد ورودی بعدی قرار می‌گیرد تا وقتی که یا به پایان رشتهٔ ورودی برسیم و یا ظرفیت آرایهٔ `array` تکمیل شود.

اشاره‌گرها به عنوان پارامتر تابع

- برنامهٔ دریافت اعداد صحیح به صورت زیر نوشته می‌شود.

```
۱  int getch (void);
۲  void ungetch (int);
۳  /* getint: get next integer from input into *pn */
۴  int
۵  getint (int *pn)
۶  {
۷    int c, sign;
۸    while (isspace (c = getch ()))  /* skip white space */
۹      ;
۱۰   if (!isdigit (c) && c != EOF && c != '+' && c != '-')
۱۱     {
۱۲       ungetch (c);  /* it is not a number */
۱۳       return 0;
۱۴     }
```

اشاره‌گرها به عنوان پارامتر تابع

```
۱۵    sign = (c == '-') ? -1 : 1;
۱۶    if (c == '+' || c == '-')
۱۷      c = getch ();
۱۸    for (*pn = 0; isdigit (c), c = getch ())
۱۹      *pn = 10 * *pn + (c - '0');
۲۰    *pn *= sign;
۲۱    if (c != EOF)
۲۲      ungetch (c);
۲۳    return c;
۲۴  }
```

اشاره‌گرها و آرایه‌ها

- اشاره‌گرها و آرایه‌ها به یکدیگر شباهت زیادی دارند.
- عملیاتی که برروی آرایه‌ها توسط عملگر زیرنویس انجام می‌دهیم، در اشاره‌گرها نیز امکان‌پذیر است.
- تعریف آرایهٔ `a[10]` را در نظر بگیرید. توسط این تعریف درواقع آرایه‌ای از ۱۰ بلوک حافظه را که هر کدام حاوی یک عدد صحیح خواهند بود در اختیار می‌گیریم. به هریک از عناصر آرایه می‌توانیم توسط عملگر زیرنویس به صورت `a[0]` ، `a[1]` ، ... و `a[9]` دسترسی پیدا کنیم.

اشاره‌گرها و آرایه‌ها

- حال اگر اشاره‌گر `pa*` را به صورت `int *pa` تعریف کنیم، می‌توانیم مقدار آن را برابر با آدرس عنصر اول آرایه با دستور `pa = &a[0]` قرار دهیم. بنابراین `pa` به اولین عنصر آرایه اشاره خواهد کرد.
- حال می‌توانیم مقدار متغیری که `pa` به آن اشاره می‌کند را توسط `x = *pa` دریافت کنیم. اگر `pa` به عنصر صفرم آرایه اشاره کند، `pa + 1` به عنصر اول و `pa + i` به عنصر `i` ام آرایه اشاره می‌کند. بنابراین `*(pa + 1)` مقدار `a[1]` و `*(pa + i)` مقدار `a[i]` را بازمی‌گردانند.
- عبارت `pa + 1` در واقع به سلول بعدی در حافظه اشاره می‌کند و به نوع متغیری که `pa` به آن اشاره می‌کند بستگی ندارد. پس اگر `pa` از نوع `char` باشد `pa + 1` در واقع یک بایت به جلو حرکت می‌کند و اگر `pa` از نوع `int` باشد، `pa + 1` در واقع چهار بایت در حافظه حرکت می‌کند.

اشاره‌گرها و آرایه‌ها

- نام یک آرایه درواقع یک اشاره‌گر است و یک متغیر از نوع آرایه درواقع اولین عنصر آرایه را نگهداری می‌کند.
- بنابراین به جای `pa = &a[0]` می‌توانیم بنویسیم `pa = a` زیرا نام آرایه برابراست با آدرس اولین عنصر آرایه.

اشاره‌گرها و آرایه‌ها

- درواقع به عناصر آرایه نیز می‌توانیم مانند اشاره‌گرها به عملگر * دسترسی پیدا کنیم. بنابراین برای دسترسی به عنصر i ام آرایه a می‌توانیم بنویسیم *(a+i) که معادل است با a[i].
- درواقع در زبان سی a[i] در هنگام ارزیابی به *(a+i) تبدیل می‌شود پس هر دو عبارت معادل هستند.
- همچنین &a[i] معادل است با a+i
- اگر pa یک اشاره‌گر باشد، عبارت *(pa+i) می‌تواند به صورت pa[i] نیز نوشته شود.

اشاره‌گرها و آرایه‌ها

- نتیجه اینکه عبارات نوشته شده به صورت نام آرایه و عملگر زیرنویس معادل هستند با عبارات نوشته شده به صورت نام اشاره‌گر و آفست از عنصر اول.
- با این حال، یک تفاوت بین اشاره‌گرها و آرایه‌ها وجود دارد. از آنجایی که اشاره‌گرها متغیر هستند می‌توانیم عملیاتی مانند متغیرها برروی آنها انجام دهیم برای مثال می‌توانیم بنویسیم `pa = a` یا `pa++` اما نام آرایه‌ها متغیر نیستند پس `a = pa` و `a++` عبارات نادرستی هستند.
- وقتی یک آرایه به یک تابع ارسال می‌شود، درواقع آدرس مکان اول آرایه به تابع ارسال می‌شود.

اشاره‌گرها و آرایه‌ها

- می‌خواهیم تابعی بنویسیم که طول یک رشته را محاسبه کند. با استفاده از اشاره‌گرها این تابع را به صورت زیر می‌نویسیم.

```
۱ /* strlen: return length of string s */
۲ int strlen(char *s)
۳ {
۴   int n;
۵   for (n = 0; *s != '\0', s++)
۶     n++;
۷   return n;
۸ }
```

اشاره‌گرها و آرایه‌ها

- در واقع در این تابع اشاره‌گر s یک کپی است از اشاره‌گر ارسال شده به تابع از طریق آرگومان ورودی تابع. پس اگر آرگومان ورودی تابع یک آرایه باشد، اشاره‌گر s یک کپی از آدرس آرایه را ذخیره می‌کند.
- در برنامه قبل عبارت s++ تأثیری در آرگومان ورودی تابع نمی‌گذارد.

اشاره‌گرها و آرایه‌ها

- این تابع را می‌توانیم به گونه‌های متفاوت فراخوانی کنیم.

```
1 strlen("hello, world"); /* string constant */
2 strlen(array); /* char array[100]; */
3 strlen(ptr); /* char *ptr; */
```

- در پارامتر ورودی تابع `char s[]` و `char *s` معادل یکدیگرند.

اشاره‌گرها و آرایه‌ها

- همچنین گاه ممکن است قسمتی از یک آرایه را به یک تابع ارسال کنیم. برای مثال می‌توانیم بنویسیم `f(&a[2])` و یا `f(a+2)` تا آرایه `a` را به شروع از عنصر `a[2]` به تابع `f` ارسال کنیم.
- تابع `f` می‌تواند به دو صورت `f(int arr[])` یا `f(int *arr)` تعریف شود.
- همچنین اگر بدانیم که عناصر قبل از عنصر `p[0]` در حافظه وجود دارند، می‌توانیم به آنها با اندیس‌های منفی به صورت `p[-1]` و `p[-2]` و ... دسترسی پیدا کنیم.

محاسبات آدرس‌ها

- اگر p یک اشاره‌گر باشد، آنگاه ++p یک واحد به p می‌افزاید تا به عنصر بعدی در حافظه اشاره کند و p+=i درواقع i واحد به آدرس p می‌افزاید.
- فرض کنید می‌خواهیم تابعی به نام alloc(n) بنویسیم که اشاره‌گری به ابتدای n عنصر در حافظه بازمی‌گرداند. فراخوانی کنندهٔ تابع alloc(n) می‌تواند با فراخوانی این تابع n مکان در حافظه را برای ذخیره یک رشته n حرفی در اختیار بگیرد.
- همچنین می‌خواهیم afree(p) را پیاده‌سازی کنیم که مکانی که اشاره‌گر p در حافظه اشغال کرده را آزاد کند. تا بتوانیم از آن مکان حافظه مجددا استفاده کنیم.
- در کتابخانه استاندارد دو تابع malloc و free بدین منظور پیاده‌سازی شده‌اند. در اینجا به مطالعه پیاده‌سازی ساده این دو تابع می‌پردازیم.

محاسبات آدرس‌ها

- فرض کنید بافری به نام `allocbuf` در اختیار داریم که تخصیص و آزادسازی حافظه را برروی آن بافر انجام می‌دهیم.
- این بافر را به صورت ایستا تعریف می‌کنیم چون نمی‌خواهیم برنامه‌های دیگر به صورت مستقیم به این بافر دسترسی داشته باشند.
- همچنین در این برنامه به یک اشاره‌گر به نام `allocp` نیاز داریم که به آدرس مکان در دسترس بعدی در بافر اشاره می‌کند.
- وقتی می‌خواهیم توسط تابع `alloc` تعداد `n` کاراکتر در حافظه را رزرو کنیم، باید ابتدا بررسی شود آیا `allocbuf` این مقدار مکان آزاد در اختیار دارد یا خیر. اگر مکان آزاد موجود بود آدرس فعلی `allocp` بازگردانده می‌شود و اشاره‌گر `allocp` به `allocp + n` اشاره می‌کند.

محاسبات آدرس‌ها

- اگر فضای مورد نیاز در بافر وجود نداشت تابع `alloc` مقدار صفر را بازمی‌گرداند.
- همچنین با فراخوانی `afree(p)` اشاره‌گر `allocp` به `p` اشاره خواهد کرد.

محاسبات آدرس‌ها

- این توابع به صورت زیر نوشته می‌شوند.

```
۱  #define ALLOCSIZE 10000   /* size of available space */
۲  static char allocbuf[ALLOCSIZE];  /* storage for alloc */
۳  static char *allocp = allocbuf;    /* next free position */
۴  char *
۵  alloc (int n)    /* return pointer to n characters */
۶  {
۷    if (allocbuf + ALLOCSIZE - allocp >= n)
۸      {     /* it fits */
۹        allocp += n;
۱۰       return allocp - n;^^I/* old p */
۱۱     }
۱۲   else    /* not enough room */
۱۳     return 0;
۱۴ }
```

محاسبات آدرس‌ها

```
۱۵ void
۱۶ afree (char *p)    /* free storage pointed to by p */
۱۷ {
۱۸   if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
۱۹     allocp = p;
۲۰ }
```

محاسبات آدرس‌ها

- در عبارت

```
1 static char *allocp = allocbuf
```

در واقع `allocp` به اولین مکان آزاد در بافر `allocbuf` اشاره می‌کند. می‌توانستیم این عبارت به صورت

```
1 static char  *allocp = &allocbuf[0]
```

نیز بنویسیم، زیرا نام آرایه‌ها درواقع معادل است با آدرس اولین عنصر آنها.

محاسبات آدرس‌ها

- در برنامه قبل توسط

```
1 if(allocbuf + ALLOCSIZE - allocp > = n)
```

بررسی می‌کنیم که فضای کافی برای n کاراکتر بر روی بافر وجود داشته. اگر فضای کافی برروی بافر وجود داشته باشد، مقدار جدید allocp تنظیم خواهد و اشاره‌گری به ابتدای حافظهٔ تخصیص داده شده بازگردانده خواهد شد. در غیر اینصورت باید سیگنالی به فراخوانی کننده بازگردانده شود مبنی بر اینکه فضای خالی در حافظه وجود ندارد. از عدد صفر برای این سیگنال استفاده می‌شود.

- انتساب اعداد به اشاره‌گرها ممکن نیست، اما امکان انتساب عدد صفر به یک اشاره‌گر وجود دارد. اشاره‌گری که مقدار آن برابر با صفر باشد، به جایی در حافظه اشاره نمی‌کند. نماد ثابت NULL برای اشاره‌گر صفر در کتابخانه استاندارد تعریف شده است.

محاسبات آدرس‌ها

- اشاره‌گرها را می‌توان با یکدیگر توسط عملگرهای رابطه‌ای مانند == ، =! ، > ، < ، => یا =< مقایسه کرد.
- برای مثال p < q مقدار درست را بازمی‌گرداند اگر p به مکانی در حافظه قبل از q اشاره کند به عبارت دیگر اگر آدرس حافظه‌ای که p نگه می‌دارد از آدرس حافظه‌ای که q نگه می‌دارد کوچکتر باشد، p < q است.
- جمع یک اشاره‌گر با یک عدد صحیح نیز وجود دارد. عبارت p + n مکان حافظه‌ای نشان می‌دهد که n واحد از مکان حافظهٔ اشاره‌گر p بیشتر باشد. این جمع به نوع متغیر p بستگی دارد. اگر p یک عدد صحیح باشد، هر واحد از n درواقع ۴ بایت است.
- تفریق اشاره‌گرها از یکدیگر نیز امکان پذیر است. برای مثال اگر p و q به دو مکان از حافظه اشاره کنند و داشته باشیم p < q ، آنگاه q - p + 1 تعداد عناصر بین آنها را مشخص می‌کند.

محاسبات آدرس‌ها

- برنامه زیر طول یک رشته را توسط محاسبات برروی اشاره‌گرها به دست می‌آورد.

```
۱ /* strlen: return length of string s */
۲ int
۳ strlen (char *s)
۴ {
۵   char *p = s;
۶   while (*p != '\0')
۷     p++;
۸   return p - s;
۹ }
```

محاسبات آدرس‌ها

- در کتابخانه استاندارد مقدار بازگشتی تابع `strlen` از نوع `size-t` است که یک عدد صحیح بدون علامت است.
- عملیات مجاز بر روی اشاره‌گرها عبارتند از : انتساب اشاره‌گرهای هم نوع به یکدیگر، افزودن یک عدد صحیح به یک اشاره‌گر، کاستن یک عدد صحیح از یک اشاره‌گر، مقایسهٔ دو اشاره‌گر که به یک آرایهٔ واحد اشاره می‌کنند، انتساب عدد صفر به اشاره‌گر و مقایسه یک اشاره‌گر با عدد صفر.
- عملیات دیگر مانند ضرب کردن دو اشاره‌گر مجاز نیست، زیرا این عملیات بی معنی است.

اشاره‌گر نوع کاراکتر

- یک رشته آرایه‌ای است از حروف (کاراکترها). ذخیره‌سازی بیتی رشته‌ها به نحوی است که کاراکتر آخر آنها `'\0'` است. بنابراین برای ذخیرهٔ یک رشته با طول `n` به `n + 1` مکان در حافظه نیاز است.
- یک تابع می‌تواند یک اشاره‌گر نوع کاراکتر دریافت کند. برای مثال تابع `printf` به عنوان پارامتر یک اشاره‌گر نوع کاراکتر دریافت می‌کند.
- یک رشته در حافظه در واقع آرایه‌ای از کاراکترهاست. بنابراین در برنامهٔ زیر یک اشاره‌گر به یک اشاره‌گر دیگر انتساب داده می‌شود.

```
1 char *pmessage;
2 pmessage = "hello" ;
```

اشاره‌گر نوع کاراکتر

- توجه کنید که عبارت قبل رشته را کپی نمی‌کند، بلکه تنها آدرس اشاره‌گر را تغییر می‌دهد.
- یک تفاوت در مقداردهی اولیه بین آرایه‌ها و اشاره‌گرها وجود دارد.
- دو دستور زیر را در نظر بگیرید :

```
1 char amessage[] = "now is the time"; /* an array */
2 char *pmessage = "now is the time"; /* a pointer */
```

- متغیر amessage یک آرایه است و در مقداردهی اولیه اندازهٔ آن توسط رشته داده شده تعیین می‌شود. محتوای این آرایه رشته‌ای است که در مقداردهی اولیه برابر با آن قرارگرفته شده است. اما متغیر pmessage یک اشاره‌گر است. در مقداردهی اولیه، رشته‌ای در مکانی بر روی حافظه قرار می‌گیرد و اشاره‌گر به مکان آن رشته اشاره می‌کند. در طول اجرای برنامه اشاره‌گر ممکن است به مکانی دیگر اشاره کند و رشته را از دست بدهد ولی آرایه نمی‌تواند به مکانی دیگر اشاره کند.

اشاره‌گر نوع کاراکتر

- می‌خواهیم تابعی بنویسیم که دو اشاره‌گر نوع کاراکتر را دریافت کند و محتوای رشته‌ای که توسط اشاره‌گر دوم مشخص شده است را در مکان حافظه‌ای که توسط رشته اول مشخص شده کپی کند.
- این تابع با نام `strcpy` در کتابخانه استاندارد پیاده سازی شده است. تابع `strcpy(s,t)` رشته `t` را در رشته `s` کپی می‌کند.
- توجه کنید که دستور `s = t` تنها آدرس اشاره‌گر `t` را در اشاره‌گر `s` کپی می‌کند.

اشاره‌گر نوع کاراکتر

- تابع کپی رشته به صورت زیر پیاده‌سازی می‌شود.

```
۱  /* strcpy: copy t to s; array subscript version */
۲  void
۳  strcpy (char *s, char *t)
۴  {
۵    int i;
۶    i = 0;
۷    while ((s[i] = t[i]) != '\0')
۸      i++;
۹  }
```

اشاره‌گر نوع کاراکتر

- تابع کپی رشته را می‌توان به صورت زیر نیز پیاده‌سازی کرد.

```
۱  /* strcpy: copy t to s; pointer version */
۲  void
۳  strcpy (char *s, char *t)
۴  {
۵    int i;
۶    i = 0;
۷    while ((*s = *t) != '\0')
۸      {
۹        s++;
۱۰       t++;
۱۱     }
۱۲ }
```

اشاره‌گر نوع کاراکتر

- از آنجایی که فراخوانی در زبان سی با مقدار است، مقدار آرگومان‌ها در پارامترهای تابع کپی می‌شوند، بنابراین تابع می‌تواند پارامترهای `s` و `t` را تغییر دهد بدون اینکه مقدار آرگومان‌های تابع (که اشاره‌گر هستند) تغییر کند. با این حال، چون تابع به مکان‌های حافظه از طریق پارامترهای ورودی دسترسی دارد، می‌تواند محتوای حافظه‌ای که توسط اشاره‌گرها مشخص شده است را تغییر دهد.

اشاره‌گر نوع کاراکتر

- تابع کپی رشته را می‌توان به طور فشرده‌تر نیز به صورت زیر پیاده‌سازی کرد.

```
۱ /* strcpy: copy t to s; pointer version 2 */
۲ void strcpy(char *s, char *t)
۳ {
۴   while ((*s++ = *t++) != '\0');
۵ }
```

اشاره‌گر نوع کاراکتر

- توجه کنید که مقدار '\0' درواقع صفر است و در حلقه نیازی به مقایسه مقدار عبارت شرطی با '\0' نداریم.
- تابع کپی رشته را می‌توان به طور فشرده‌تر نیز به صورت زیر پیاده‌سازی کرد.

```
1 /* strcpy: copy t to s; pointer version 3 */
2 void strcpy(char *s, char *t)
3 {
4   while (*s++ = *t++);
5 }
```

اشاره‌گر نوع کاراکتر

- تابع دیگری که در اینجا بررسی می‌کنیم، دو رشته را دریافت کرده، با یکدیگر مقایسه می‌کند. اگر رشته اول از رشته دوم از نظر لغوی کوچکتر بود یک عدد منفی بازگردانده می‌شود. اگر دو رشته برابر بودند، عدد صفر و اگر رشته اول از رشته دوم بزرگ‌تر بود، یک عدد مثبت بازگردانده می‌شود. مقداری که بازگردانده می‌شود، تفاضل اولین حرفی که در رشته اول در رشته دوم متفاوت است.

اشاره‌گر نوع کاراکتر

- تابع مقایسه دو رشته به صورت زیر نوشته می‌شود.

```
۱  /* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
۲  int
۳  strcmp (char *s, char *t)
۴  {
۵    int i;
۶    for (i = 0; s[i] == t[i]; i++)
۷      if (s[i] == '\0')
۸        return 0;
۹    return s[i] - t[i];
۱۰ }
```

اشاره‌گر نوع کاراکتر

- تابع مقایسه دو رشته را می‌توان به صورت زیر با استفاده از عملگرهای اشاره‌گرها نیز پیاده‌سازی کرد.

```
۱ /* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
۲ int
۳ strcmp (char *s, char *t)
۴ {
۵   for (; *s == *t; s++, t++)
۶     if (*s == '\0')
۷       return 0;
۸   return *s - *t;
۹ }
```

اشاره‌گر نوع کاراکتر

- عملگرهای ++ و -- هم به صورت پسوندی استفاده می‌شوند و هم به صورت پیشوندی. برای مثال عبارت `val = *p++` مقدار `val` را در مکان حافظهٔ `p` کپی می‌کند و سپس اشاره‌گر را یک واحد به جلو حرکت می‌دهد. عبارت `val = *--p` ابتدا اشاره‌گر را یک واحد به عقب حرکت می‌دهد و سپس مقداری که اشاره‌گر به آن اشاره می‌کند را در `val` ذخیره می‌کند.

اشاره‌گر به اشاره‌گر

- از آنجایی که اشاره‌گرها خود متغیر هستند، می‌توانند مانند بقیهٔ متغیرها در آرایه‌ها نگهداری شوند.
- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که مجموعه‌ای از رشته‌ها را به ترتیب الفبایی مرتب کند.
- در گذشته الگوریتم مرتب‌سازی را برای اعداد صحیح بررسی کردیم.
- برای نگهداری یک مجموعه از رشته‌ها به آرایه‌ای از اشاره‌گرها نیاز داریم. هر عنصر آرایه به یک رشته (یا یک خط) اشاره خواهد کرد. دو رشته را می‌توانیم با استفاده از تابع `stdcmp` مقایسه کنیم. وقتی می‌خواهیم جای دو رشته را در این آرایه عوض کنیم، کافی است اشاره‌گرها را جابجا کنیم. بنابراین در تعویض مکان دو رشته در آرایه، اشاره‌گرهای آنها با یکدیگر تعویض می‌کنیم، به محتوای رشته‌ها.

اشاره‌گر به اشاره‌گر

- می‌خواهیم برنامه‌ای بنویسیم که تعدادی رشته را هر یک در یک خط از ورودی دریافت کند، رشته‌ها را به ترتیب حروف الفبایی مرتب کند و در نهایت رشته‌های مرتب شده را چاپ کند.
- قبل از نوشتن تابع مرتب‌سازی ساختار کلی برنامه را بررسی می‌کنیم.

اشاره‌گر به اشاره‌گر

```
۱ #include <stdio.h>
۲ #include <string.h>
۳ #define MAXLINES 5000^^I^^I/* max #lines to be sorted */
۴ char *lineptr[MAXLINES];^^I/* pointers to text lines */
۵ int readlines (char *lineptr[], int nlines);
۶ void writelines (char *lineptr[], int nlines);
۷ void qsort (char *lineptr[], int left, int right);
```

اشاره‌گر به اشاره‌گر

```
 8  /* sort input lines */
 9  int main ()
10  {
11    int nlines;      /* number of input lines read */
12    if ((nlines = readlines (lineptr, MAXLINES)) >= 0)
13      {
14        qsort (lineptr, 0, nlines - 1);
15        writelines (lineptr, nlines);
16        return 0;
17      }
18    else
19      {
20        printf ("error: input too big to sort\n");
21        return 1;
22      }
23  }
```

اشاره‌گر به اشاره‌گر

```
۲۴ #define MAXLEN 1000      /* max length of any input line */
۲۵ int getline (char *, int);
۲۶ char *alloc (int);
۲۷ /* readlines: read input lines */
۲۸ int
۲۹ readlines (char *lineptr[], int maxlines)
۳۰ {
۳۱   int len, nlines;
۳۲   char *p, line[MAXLEN];
۳۳   nlines = 0;
۳۴    while ((len = getline (line, MAXLEN)) > 0)
۳۵     if (nlines >= maxlines || p = alloc (len) == NULL)
۳۶     return -1;
```

اشاره‌گر به اشاره‌گر

```
۳۷      else
۳۸        {
۳۹      line[len - 1] = '\0';     /* delete newline */
۴۰      strcpy (p, line);
۴۱      lineptr[nlines++] = p;
۴۲        }
۴۳    return nlines;
۴۴  }
۴۵  /* writelines: write output lines */
۴۶  void
۴۷  writelines (char *lineptr[], int nlines)
۴۸  {
۴۹     int i;
۵۰     for (i = 0; i < nlines; i++)
۵۱     printf ("%s\n", lineptr[i]);
۵۲  }
```

اشاره‌گر به اشاره‌گر

- در این برنامه آرایه به رشته‌ها را به صورت آرایه‌ای از اشاره‌گرها به صورت

```
1 char *lineptr [MAXLINES]
```

تعریف کردیم.

- بنابراین `lineptr` یک آرایه با `MAXLINES` عنصر است که هر عنصر آن یک اشاره‌گر از نوع `char` است. بنابراین `lineptr[i]` یک اشاره‌گر از نوع کاراکتر است و `*lineptr[i]` کاراکتری است که آن عنصر به آن اشاره می‌کند، که در واقع اولین کاراکتر رشته است.

اشاره‌گر به اشاره‌گر

- از آنجایی که `lineptr` نام یک آرایه است، می‌تواند مانند یک اشاره‌گر مورد استفاده قرار بگیرد، بنابراین تابع `writelines` می‌تواند به صورت زیر نوشته شود.

```
۱ /* writelines: write output lines */
۲ void
۳ writelines (char *lineptr[], int nlines)
۴ {
۵   while (nlines-- > 0)
۶     printf ("%s\n", *lineptr++);
۷ }
```

- در ابتدا `lineptr*` به اولین خط اشاره می‌کند. سپس اشاره‌گر به سمت جلو حرکت می‌کند و به عناصر بعدی اشاره می‌کند.

اشاره‌گر به اشاره‌گر

- حال که می‌توانیم ورودی و خروجی را کنترل کنیم، الگوریتم مرتب‌سازی را پیاده سازی کنیم.
- الگوریتم مرتب‌سازی سریع را که برای اعداد صحیح پیاده‌سازی کردیم، کمی تغییر می‌دهیم تا رشته‌ها را دریافت کند. برای مقایسهٔ دو رشته از تابع `strcmp` استفاده می‌کنیم.

اشاره‌گر به اشاره‌گر

- تابع مرتب‌سازی برای رشته‌ها به صورت زیر پیاده‌سازی می‌شود.

```
۱  /* qsort: sort v[left]...v[right] into increasing order */
۲  void
۳  qsort (char *v[], int left, int right)
۴  {
۵    int i, last;
۶    void swap (char *v[], int i, int j);
۷    if (left >= right)    /* do nothing if array contains */
۸      return;        /* fewer than two elements */
۹    swap (v, left, (left + right) / 2);
۱۰   last = left;
۱۱   for (i = left + 1; i <= right; i++)
```

اشاره‌گر به اشاره‌گر

```
۱۲      if (strcmp (v[i], v[left]) < 0)
۱۳        swap (v, ++last, i);
۱۴    swap (v, left, last);
۱۵    qsort (v, left, last - 1);
۱۶    qsort (v, last + 1, right);
۱۷  }
```

اشاره‌گر به اشاره‌گر

- همچنین برای جابجایی دو رشته در آرایه از تابع swap به صورت زیر استفاده می‌کنیم.

```
۱ /* swap: interchange v[i] and v[j] */
۲ void
۳ swap (char *v[], int i, int j)
۴ {
۵   char *temp;
۶   temp = v[i];
۷   v[i] = v[j];
۸   v[j] = temp;
۹ }
```

آرایه‌های چند بعدی

- در زبان سی امکان ایجاد آرایه‌های چند بعدی نیز وجود دارد، اگر آرایه‌های یک بعدی کاربرد بیشتری دارند.
- می‌خواهیم برنامه‌ای بنویسیم که توسط آن تعیین کنیم یک روز از یک ماه چندمین روز از سال است و همچنین یک روز معین از سال چه روزی از چه ماهی است.
- تابع `day-of-year` را برای تبدیل یک روز از یک ماه به یک روز از سال و تابع `month-year` را برای تبدیل یک روز از سال به یک روز از یک ماه تعریف می‌کنیم.
- از آنجایی که تابع دوم دو خروجی دارد، می‌توانیم خروجی‌ها را توسط اشاره‌گرها در ورودی از تابع دریافت کنیم. برای مثال ۶۰ امین روز سال ۱۹۸۸ برابر با ۲۹ فوریه است که آن را با فراخوانی تابع `month-day(1988,60, &m,&d)` به دست می‌آوریم. تابع `m` را برابر با ۲ و `d` را برابر با ۹ قرار خواهد داد.

آرایه‌های چند بعدی

- تعداد روزهای ماه در سال‌های کبیسه متفاوت است از تعداد روزها در سال‌های معمولی، بنابراین از یک جدول با ۲ سطر و ۱۲ ستون استفاده می‌کنیم که سطر اول تعداد روزها در سال معمولی و سطر دوم تعداد روزها در سال کبیسه را تعیین می‌کند. این جدول را به صورت یک آرایهٔ دو بعدی تعریف می‌کنیم.

آرایه‌های چند بعدی

- برنامهٔ تبدیل روز ماه به روز سال به صورت زیر نوشته می‌شود.

```
۱  static char daytab[2][13] = {
۲    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
۳    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
۴  };
۵  /* day_of_year: set day of year from month & day */
۶  int
۷  day_of_year (int year, int month, int day)
۸  {
۹    int i, leap;
۱۰   leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
۱۱   for (i = 1; i < month; i++)
۱۲     day += daytab[leap][i];
۱۳   return day;
۱۴ }
```

آرایه‌های چند بعدی

- برنامهٔ تبدیل روز سال به روز ماه به صورت زیر نوشته می‌شود.

```
۱  /* month_day: set month, day from day of year */
۲  void
۳  month_day (int year, int yearday, int *pmonth, int *pday)
۴  {
۵    int i, leap;
۶    leap = year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
۷    for (i = 1; yearday > daytab[leap][i]; i++)
۸      yearday -= daytab[leap][i];
۹    *pmonth = i;
۱۰   *pday = yearday;
۱۱ }
```

آرایه‌های چند بعدی

- دقت کنید که مقدار منطقی به دست آمده در متغیر `leap` یا صفر است و یا یک. بنابراین این مقدار را می‌توان به عنوان اندیس آرایهٔ دو بعدی `daytab` استفاده کرد.
- برای تعریف این آرایه از نوع `char` استفاده کردیم، زیر می‌خواهیم مقادیر کوچک را نگهداری کنیم که یک بایت برای آنها کافی است.
- یک آرایهٔ دو بعدی در واقع یک آرایه از آرایه‌ها است و بنابراین برای دسترسی به هر عنصر آن می‌نویسیم `daytab[i][j]`.
- در حافظه، مقادیر یک آرایهٔ دو بعدی سطر به سطر ذخیره می‌شوند، بنابراین دو مقدار `daytab[i][j]` و `daytab[i][j+1]` در کنار یکدیگر در حافظه قرار می‌گیرند.

آرایه‌های چند بعدی

- برای مقداردهی اولیه یک آرایه می‌توانیم از علامت آکولاد استفاده کنیم. مقادیری که در آکولاد قرار می‌گیرند به ترتیب عناصر آرایه را تشکیل می‌دهند. همچنین در یک آرایهٔ دو بعدی هر سطر درون آکولاد و همهٔ ستون‌ها در یک آکولاد بیرونی قرار می‌گیرند.
- در این برنامه اولین ستون جدول `daytab` را برابر با صفر قرار دادیم تا بتوانیم ماه‌ها را از ۱ تا ۱۲ شماره‌گذاری کنیم به جای ۰ تا ۱۱.
- وقتی می‌خواهیم یک آرایهٔ دو بعدی را به یک تابع ارسال کنیم، باید تعداد ستون‌های آرایه‌ها را در امضای تابع مشخص کنیم، زیرا آنچه به تابع ارسال می‌شود، اشاره‌گری است از آرایه‌ها و برای خواندن صحیح مقادیر از حافظه باید اندازهٔ آرایه‌ها مشخص باشد.
- بنابراین تابعی که آرایهٔ دو بعدی `daytab` را دریافت می‌کند می‌تواند به صورت `if(daytab[2][13])` یا `if(int daytab[][13])` یا `if(int (*daytab)[13])` تعریف شود.

آرایه‌های چند بعدی

- در تعریف آخر درواقع می‌گوییم پارامتر تابع اشاره‌گری است از آرایه‌های ۱۳ عنصری. پرانتزها ضروری هستند، زیرا اولویت براکت [] بیشتر از اولویت عملگر ستاره * است.
- درواقع `int *daytab[13]` یک آرایهٔ ۱۳ عنصری است از اشاره‌گرها به اعداد صحیح.
- در حالت کلی برای یک آرایهٔ چند بعدی تنها بعد اول را می‌توان بدون تعداد عناصر قید کرد و تعداد عناصر ابعاد دیگر باید دقیقا در تعاریف مشخص شوند.

مقداردهی اولیهٔ آرایه از اشاره‌گرها

- فرض کنید می‌خواهیم تابعی بنویسیم که با دریافت عدد یک ماه، اشاره‌گری به نام آن ماه بازگرداند. نام ماه‌ها را می‌توانیم در یک آرایه از رشته‌ها (اشاره‌گرهای کاراکتری) نگهداری کنیم و این آرایه می‌تواند در درون تابع قرار بگیرد، اما چون مقادیر آن آرایه باید توسط دیگر تابع در دسترس باشد، آرایه را به صورت ایستا تعریف می‌کنیم.

مقداردهی اولیهٔ آرایه از اشاره‌گرها

- این تابع را می‌توانیم به صورت زیر تعریف کنیم.

```
۱  /* month_name: return name of n-th month */
۲  char *
۳  month_name (int n)
۴  {
۵    static char *name[] = {
۶      "Illegal month",
۷      "January", "February", "March",
۸      "April", "May", "June",
۹      "July", "August", "September",
۱۰     "October", "November", "December"
۱۱   };
۱۲   return (n < 1 || n > 12) ? name[0] : name[n];
۱۳ }
```

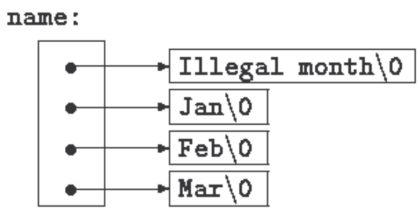
اشاره‌گرها و آرایه‌های چند بعدی

- یک آرایهٔ دو بعدی را به صورت `int a[10][20]` تعریف می‌کنیم و یک آرایه به اشاره‌گر را به صورت `int *b[10]` . در اینصورت `a[3][4]` و `b[3][4]` هر دوبه یک `int` اشاره می‌کنند، اما `a` یک آرایهٔ دو بعدی واقعی است که دقیقاً در آن ۲۰۰ مکان حافظه برای مقادیر صحیح در نظر گرفته شده است و برای به دست آوردن خانهٔ حافظه‌ای که در آن سطر `row` و ستون `col` قرار می‌گیرد. می‌توان از عبارت `20 * row + col` استفاده کرد.
- اما متغیر `b` تنها یک آرایهٔ ۱۰ عنصری از اشاره‌گرها را تعریف می‌کند. فرض کنید هر عنصر `b` به یک آرایهٔ ۲۰ عنصری اشاره کند در این صورت ۲۰۰ مکان حافظه برای نگهداری اعداد صحیح داریم و به علاوه ۱۰ مکان حافظه برای نگهداری آدرس اشاره‌گرهای هر سطر.
- روش دوم فضای بیشتری اشغال می‌کند اما مزیت آن این است که هر سطر از آرایهٔ دو بعدی می‌تواند طول متفاوتی از سطرهای دیگر داشته باشد.

اشاره‌گرها و آرایه‌های چند بعدی

- برای مثال آرایه‌ای از رشته‌ها را در نظر بگیرید. هر عنصری در این آرایه در واقع اشاره‌گری است که به رشته‌ها با طول‌های متفاوت اشاره می‌کند.

```
1 char * name[] = {"Illegal month" , "Jan" , "Feb" , "Mar"};
```



اشاره‌گرها و آرایه‌های چند بعدی

- اما آرایهٔ دو بعدی در حافظه به صورت زیر است.

```
1  char aname[][15] = {"Illegal month" , "Jan" , "Feb" , "Mar"};
```

aname:

Illegal month\0	Jan\0	Feb\0	Mar\0
0	15	30	45

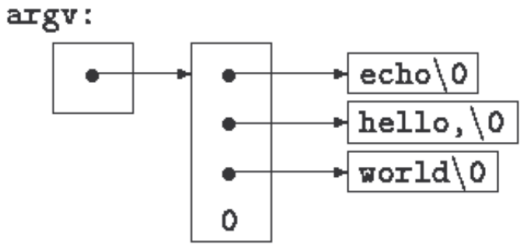
آرگومان‌های ورودی برنامه

- در محیطی که برنامهٔ سی در آن اجرا می‌شود، امکان دریافت آرگومان توسط برنامه وجود دارد. بدین ترتیب با اجرای یک برنامه تعدادی ورودی به برنامه ارسال می‌شوند. این ورودی‌ها توسط برنامه دریافت شده، در برنامه استفاده می‌شوند.
- وقتی تابع `main` فراخوانی می‌شود، معمولا با دو آرگومان فراخوانی انجام می‌شود. آرگومان اول تعداد آرگومان‌ها را مشخص می‌کند و `argc` (تعداد آرگومان) [1] نامیده می‌شود و آرگومان دوم اشاره‌گری است به یک آرایه از رشته‌های کاراکتری و شامل مقدار همهٔ آرگومان است و `argv` (وکتور آرگومان‌ها) [2] نامیده می‌شود.
- برای مثال برنامه `echo` را در نظر بگیرید. این برنامه یک ورودی می‌گیرد و مقدار ورودی را چاپ می‌کند. برای مثال `world` ، `echo hello` رشته `world` ، `hello` را چاپ می‌کند.

[1] argument count
[2] argument vector

آرگومان‌های ورودی برنامه

- مقدار argv[0] نام برنامه است، بنابراین مقدار argc حداقل ۱ است. اگر مقدار argc برابر با ۱ باشد، درواقع هیچ آرگومانی به برنامه ارسال نشده است. در مثال echo درواقع argc برابر با ۳ است. مقادیر argv[0] ، argv[1] و argv[2] به ترتیب "echo" ، "hello" و "world" می‌باشد. بنابراین مقدار اولین آرگومان در argv[1] و آخرین آرگومان در argv[argc-1] قرار می‌گیرد.



آرگومان‌های ورودی برنامه

- برنامهٔ echo به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  /* echo command-line arguments; 1st version */
۳  int main (int argc, char *argv[])
۴  {
۵    int i;
۶    for (i = 1; i < argc; i++)
۷      printf ("%s%s", argv[i], (i < argc - 1) ? " " : "");
۸    printf ("\n");
۹    return 0;
۱۰ }
```

آرگومان‌های ورودی برنامه

- از آنجایی که argv یک اشاره‌گر به آرایه‌ای از اشاره‌گرهاست می‌توانیم به جای استفاده از آن به صورت آرایه، از آن به صورت اشاره‌گر استفاده کنیم.

```
۱ /* echo command-line arguments; 2nd version */
۲ int main (int argc, char *argv[])
۳ {
۴   while (--argc > 0)
۵     printf ("%s%s", *++argv, (argc > 1) ? " " : "");
۶   printf ("\n");
۷   return 0;
۸ }
```

آرگومان‌های ورودی برنامه

- با هربار افزایش اشاره‌گر argv اشاره‌گر به مکان حافظه بعدی اشاره می‌کند و *argv یک آرگومان اشاره می‌کند.
- عبارت چاپ رشته را می‌توانیم به صورت زیر بنویسیم.

```
printf ((argc > 1) ? "%s " : "%s" , *++ argv)
```

- حال می‌خواهیم برنامه‌ای بنویسیم که ورودی‌های دریافت شده از آرگومان‌های برنامه را در یک متن که از ورودی استاندارد دریافت می‌شود، جستجو کند.

آرگومان‌های ورودی برنامه

- این برنامه جستجو که در یونیکس و لینوکس grep نامیده می‌شود به صورت زیر است.

```
۱  #include <stdio.h>
۲  #include <string.h>
۳  #define MAXLINE 1000
۴  int getline (char *line, int max);
۵  /* find: print lines that match pattern from 1st arg */
۶  int main (int argc, char *argv[])
۷  {
۸    char line[MAXLINE];
۹    int found = 0;
۱۰   if (argc != 2)
۱۱       printf ("Usage: find pattern\n");
۱۲   else
```

آرگومان‌های ورودی برنامه

```
۱۳      while (getline (line, MAXLINE) > 0)
۱۴      if (strstr (line, argv[1]) != NULL)
۱۵     {
۱۶     printf ("%s", line);
۱۷     found++;
۱۸     }
۱۹     return found;
۲۰  }
```

آرگومان‌های ورودی برنامه

- تابع `strstr` یک اشاره‌گر به اولین وقوع رشته `t` در رشته `s` بازمی‌گرداند و در صورتی که رشته پیدا نشود مقدار `NULL` را بازمی‌گرداند.
- حال فرض کنید می‌خواهیم این برنامه را تعمیم دهیم به طوری‌که در برخی مواقع همهٔ خطوط دریافت شده از کاربر را به جز خطوطی که شامل یک عبارت معین هستند را چاپ کند و در برخی مواقع شماره خطوط دریافت شده توسط کاربر را قبل از خط مورد نظر چاپ کند. در سیستم عامل یونیکس برای مشخص کردن آرگومان‌های ورودی، از یک حرف و یک خط تیره استفاده می‌شود. برای مثال می‌توانیم از آرگومان `x-` برای نشان دادن انتخاب به جز `(except)` استفاده کنیم و از آرگومان `n-` برای نشان دادن انتخاب شماره خط `(number)`.
- بنابراین `find -x -n pattern` بدین معناست که می‌خواهیم تمام خطوطی را که حاوی الگوی `pattern` نیستند را با شماره خط آنها چاپ کنیم. همچنین ممکن است بخواهیم این دستور را به صورت `find -nx pattern` وارد کنیم.

آرگومان‌های ورودی برنامه

- این برنامهٔ جستجو به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  #include <string.h>
۳  #define MAXLINE 1000
۴  int getline (char *line, int max);
۵  /* find: print lines that match pattern from 1st arg */
۶  int main (int argc, char *argv[])
۷  {
۸    char line[MAXLINE];
۹    long lineno = 0;
۱۰   int c, except = 0, number = 0, found = 0;
۱۱   while (--argc > 0 && (*++argv)[0] == '-')
```

آرگومان‌های ورودی برنامه

```
۱۲        while (c = *++argv[0])
۱۳            switch (c)
۱۴                {
۱۵                case 'x':
۱۶                     except = 1;
۱۷                     break;
۱۸                case 'n':
۱۹                     number = 1;
۲۰                     break;
۲۱                default:
۲۲                     printf ("find: illegal option %c\n", c);
۲۳                     argc = 0;
۲۴                     found = -1;
۲۵                     break;
۲۶                  }
```

آرگومان‌های ورودی برنامه

```
۲۷    if (argc != 1)
۲۸      printf ("Usage: find -x -n pattern\n");
۲۹    else
۳۰       while (getline (line, MAXLINE) > 0)
۳۱          {
۳۲               lineno++;
۳۳               if ((strstr (line, *argv) != NULL) != except)
۳۴                   {
۳۵                          if (number)
۳۶                                 printf ("%ld:", lineno);
۳۷                          printf ("%s", line);
۳۸                          found++;
۳۹                   }
۴۰       }
۴۱     return found;
۴۲  }
```

آرگومان‌های ورودی برنامه

- دقت کنید که در این برنامه `argv++*` اشاره‌گری است به یک رشته، بنابراین `[0] (argv++*)` اولین کاراکتر این رشته است. می‌توانستیم از `argv++**` نیز برای دریافت اولین کاراکتر استفاده کنیم، ولی روش اول خواناتر است.
- پرانتزگذاری در عبارت مذکور ضروری است زیرا عبارت `argv[0]++*` معادل است با `(argv[0])++*` که کاراکتری را که در اندیس ۱ قرار دارد را باز می‌گرداند.

اشاره‌گر به توابع

- در زبان سی گرچه نام توابع را نمی‌توان به عنوان متغیر استفاده کرد، ولی می‌توان اشاره‌گر به توابع تعریف کرد. چنین اشاره‌گرهایی را می‌توان به یکدیگر انتساب کرد یا در آرایه قرار داد و یا به تابع به عنوان آرگومان ارسال کرد و یا از تابع به عنوان خروجی بازگرداند.

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که در آن با دریافت آرگومان n- جستجو را بر روی اعداد به جای رشته‌ها انجام دهد.

- یک الگوریتم مرتب‌سازی از چند بخش تشکیل شده است. یک الگوریتم مرتب‌سازی شامل یک تابع مقایسه‌گر است که دو عنصر از یک آرایه را با یکدیگر مقایسه می‌کند (برای مثال رشته‌ها به صورت الفبایی مقایسه می‌شوند و اعداد صحیح یه صورت عددی). یک الگوریتم مرتب‌سازی همچنین تعیین می‌کند که عناصر به صورت صعودی مقایسه شوند یا به صورت نزولی. همچنین یک الگوریتم مرتب‌سازی روش مرتب کردن عناصر را مشخص می‌کند.

اشاره‌گر به توابع

- نوع مقایسهٔ دو عنصر و همچنین صعودی یا نزولی بودن الگوریتم مرتب‌سازی می‌تواند به عنوان آرگومان به تابع مرتب‌سازی ارسال شود.
- مقایسه الفبایی دو رشته نوع `strcmp` انجام می‌شود. می‌توانیم تابعی به نام `numcmp` بنویسیم که شبیه به `strcmp` عمل کند ولی برای مقایسهٔ دو مقدار عددی به کار می‌رود. بدین ترتیب اگر اشاره‌گری به تابع `strcmp` به تابع مرتب‌سازی `qsort` ارسال شود، مقادیر به صورت الفبایی مقایسه می‌شوند و در صورتی که `numcmp` به تابع `qsort` ارسال شود، مقادیر به صورت عددی مرتب می‌شوند.

اشاره‌گر به توابع

- بنابراین مرتب‌سازی در حالت کلی برای رشته‌ها و اعداد به صورت زیر انجام می‌شود.

```
۱  #include <stdio.h>
۲  #include <string.h>
۳  #define MAXLINES 5000^^I^^I/* max #lines to be sorted */
۴  char *lineptr[MAXLINES];^^I/* pointers to text lines */
۵  int readlines (char *lineptr[], int nlines);
۶  void writelines (char *lineptr[], int nlines);
۷  void qsort (void *lineptr[], int left, int right,
۸  int (*comp) (void *, void *));
۹  int numcmp (char *, char *);
۱۰ /* sort input lines */
۱۱ int main (int argc, char *argv[])
۱۲ {
۱۳   int nlines;      /* number of input lines read */
۱۴   int numeric = 0;    /* 1 if numeric sort */
```

اشاره‌گر به توابع

```
۱۶    if (argc > 1 && strcmp (argv[1], "-n") == 0)
۱۷      numeric = 1;
۱۸    if ((nlines = readlines (lineptr, MAXLINES)) >= 0)
۱۹      {
۲۰        qsort ((void **) lineptr, 0, nlines - 1,
۲۱         (int (*)(void *, void *)) (numeric ? numcmp : strcmp));
۲۲        writelines (lineptr, nlines);
۲۳        return 0;
۲۴      }
۲۵    else
۲۶      {
۲۷        printf ("input too big to sort\n");
۲۸        return 1;
۲۹      }
۳۰  }
```

اشاره‌گر به توابع

- بنابراین به یک تابع `qsort` نیاز داریم که هر نوع آرایه‌ای را بتوان پردازش کند و نه فقط رشته‌ها. در این حالت تابع `qsort` یک آرایه از اشاره‌گرها را دریافت می‌کند که این اشاره‌گرها می‌توانند به اعداد و یا رشته‌ها اشاره‌کنند. برای این که این تابع در حالت کلی عمل کند از اشاره‌گر `*void` استفاده می‌کنیم که یک اشاره‌گر بدون نوع است. هر نوع اشاره‌گری را می‌توان به اشاره‌گر نوع `*void` تبدیل کرد و همچنین اشاره‌گر نوع `*void` را می‌توان به هر نوع اشاره‌گری تبدیل کرد.

اشاره‌گر به توابع

- تابع مرتب‌سازی qsort در حالت کلی به صورت زیر نوشته می‌شود.

```
۱  /* qsort: sort v[left]...v[right] into increasing order */
۲  void
۳  qsort (void *v[], int left, int right, int (*comp) (void *, void *))
۴  {
۵    int i, last;
۶    void swap (void *v[], int, int);
۷    if (left >= right)   /* do nothing if array contains */
۸      return;      /* fewer than two elements */
۹    swap (v, left, (left + right) / 2);
۱۰   last = left;
۱۱   for (i = left + 1; i <= right; i++)
۱۲     if ((*comp) (v[i], v[left]) < 0)
۱۳       swap (v, ++last, i);
```

اشاره‌گر به توابع

```
14    swap (v, left, last);
15    qsort (v, left, last - 1, comp);
16    qsort (v, last + 1, right, comp);
17  }
```

اشاره‌گر به توابع

- چهارمین پارامتر تابع `qsort` یک اشاره‌گر به تابع به صورت `int (*comp)(void* , void*)` است که به تابعی اشاره می‌کند که دو ورودی از نوع `*void` و یک خروجی از نوع `int` دارد. هر دو تابع `strcmp` و `numcmp` چنین خروجی و ورودی‌هایی دارند و هر دو می‌توانند به تابع `qsort` ارسال شوند.
- مجموعهٔ ورودی‌ها و خروجی یک تابع امضای تابع نامیده می‌شود. در ارسال یک تابع به عنوان اشاره‌گر به یک تابع دیگر، امضای تابع ارسال شده با امضای اشاره‌گر به تابع در ورودی تابع دیگر باید یکسان باشند.
- از آنجایی که در مثال قبل `comp` یک اشاره‌گر است، پس به خود تابع توسط `comp*` دسترسی پیدا می‌کنیم و بنابراین می‌نویسیم `(*comp)(v[i] , v[left])`

اشاره‌گر به توابع

- تابع مقایسهٔ دو مقدار عددی به صورت زیر تعریف می‌شود.

```
۱  #include <stdlib.h>
۲  /* numcmp: compare s1 and s2 numerically */
۳  int
۴  numcmp (char *s1, char *s2)
۵  {
۶    double v1, v2;
۷    v1 = atof (s1);
۸    v2 = atof (s2);
۹    if (v1 < v2)
۱۰     return -1;
۱۱   else if (v1 > v2)
۱۲     return 1;
۱۳   else
۱۴     return 0;
۱۵ }
```

اشاره‌گر به توابع

- تابع جابجایی دو اشاره‌گر در حالت کلی به صورت زیر خواهد بود.

```
۱ void swap (void *v[],
۲                 {
۳                    void *temp; int i, int j;
۴         )temp = v[i];
۵   v[i] = v[j];
۶   v[j] = temp;
۷ }
```