به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

ساختارهای کنترلی

ساختارهای کنترلی

- ساختارهای کنترلی [1] در زبان‌های برنامه‌نویسی جهت اعمال تصمیمات بر روی ترتیب اجرای دستورات به کار می‌روند.
- یک دستور [2] در یک زبان برنامه‌نویسی با علامت نقطه ویرگول (;) پایان می‌یابد. یک دستور می‌تواند یک عملیات انتساب، فراخوانی یک تابع، تعریف [3] متغیر یا اعلام [4] متغیر یا تابع باشد.
- علامت آکولاد باز و بسته {} برای گروه بندی دستورات استفاده می‌شوند. یک گروه از دستورات را یک بلوک [5] می‌نامیم. برای مثال دستورات تعریف یک تابع در یک بلوک قرار می‌گیرند.
- دستورات متعلق به یک ساختار کنترلی نیز که در این قسمت توضیح خواهیم داد، در یک بلوک قرار می‌گیرند.

---

[1] control flow structures
[2] statement
[3] definition
[4] declaration
[5] block

دستورات شرطی

- دستورات شرطی اگر–وگرنه `if-else` برای بیان یک تصمیم در انتخاب دستورات به کار می‌رود.
- ساختار نحوی دستور شرطی `if-else` به صورت زیر است.

```
۱ if (condition) {
۲       statement-block1
۳ } else {
۴       statement-block2
۵ }
```

- قسمت `else` اختیاری است. شرط `condition` سنجیده می‌شود و در صورتی که مقدار آن درست بود بلوک دستورات `statement-block1` اجرا می‌شود و در صورتی که مقدار آن نادرست بود بلوک دستورات `statement-block2` اجرا می‌شود.
- در صورتی که بلوک شامل تنها یک دستور باشد، می‌توان علامت‌های آکولاد را حذف کرد.

دستورات شرطی

- شرط در دستور شرطی یک عبارت منطقی است، یعنی عبارتی که یک مقدار درست یا نادرست بازمی‌گرداند.
- در یک عبارت منطقی برای تشکیل یک عبارت منطقی از عملگرهای منطقی و رابطه‌ای استفاده می‌کنیم. برای مثال `(x >= 10 && x <= 20)` ، `(x < 10 || x > 20)` ، `(c == 'q')` ، `((x & y) != 0)` عبارات منطقی هستند.
- مقدار درست [1] در زبان سی برابر با مقدار غیر صفر است و مقدار نادرست [2] برابر با مقدار صفر.
- از آنجایی که در شرط `if` یک مقدار سنجیده می‌شود، به جای `if(expression != 0)` می‌توانیم بنویسیم `if(expression)` .

[1] true
[2] false

دستورات شرطی

- عبارت else همیشه متعلق به نزدیک‌ترین عبارت if است.
- برای مثال در قطعه برنامه زیر else متعلق به if دوم است.

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

- اگر بخواهیم در برنامه فوق else متعلق به if اول باشد، باید قطعه برنامه را به صورت زیر بنویسیم.

```
if (n > 0){
    if (a > b)
        z = a;
}
else
    z = b;
```

دستورات شرطی

- برنامهٔ زیر را در نظر بگیرید. گرچه برنامه‌نویس با دندانه‌گذاری [1] کد مقصود خود را بیان داشته و هدفش این بوده که `else` متعلق به `if` اول باشد، ولی کامپایلر برنامه را به نحوی دیگر اجرا می‌کند، زیرا `else` متعلق به نزدیک‌ترین `if` یعنی `if` دوم است.

```
۱ if (n > 0)
۲   for (i = 0; i < n; i++)
۳     if (s[i] > 0) {
۴       printf("...");
۵       return i;
۶     }
۷ else  /* WRONG */
۸   printf("error -- n is negative\n");
```

[1] indentation

دستورات شرطی

- در دستور شرطی `if-else` تنها دو حالت سنجیده می‌شود یعنی یا `condition` درست است و یا نادرست.
- اگر بخواهیم چندین شرط با بسنجیم می‌توانیم از دستورات `if else` استفاده کنیم.
- دستورات `if else` را می‌توانیم به صورت زیر بنویسیم.

```
if (condition1) {
      statement-block1
} else if (condition2) {
      statement-block2
}
...
else {
      statement-blockN
}
```

دستورات شرطی

- این شرط‌ها به ترتیب سنجیده می‌شوند. اگر `condition1` درست باشد بلوک دستورات `statement-block1` اجرا می‌شود. در غیراینصورت اگر `condition2` درست باشد، بلوک دستورات `statement-block2` اجرا می‌شود، و در صورتی که هیچ یک از شرط ها برقرار نبود، بلوک دستورات `statement-blockN` اجرا می‌شود.
- قسمت `else` آخر وقتی اجرا می‌شود که هیچ کدام از شرط‌ها درست نباشند. این قسمت اختیاری است و می‌توان آن را می‌توان حذف کرد.

دستورات شرطی

- فرض کنید v یک آرایهٔ مرتب شده از اعداد صحیح است. در برنامه زیر می‌خواهیم یک عنصر را در آرایه جستجو کنیم. فرض کنید مقدار این عنصر در x ذخیره می‌شود.
- برای این جستجو از الگوریتم جستجوی دودویی [1] استفاده می‌کنیم.
- در جستجوی دودویی ابتدا مقدار x را با عنصر وسط آرایه v مقایسه می‌کنیم. اگر مقدار x کمتر از مقدار عنصر وسط آرایه بود، جستجو را بر روی نیمه اول آرایه ادامه می‌دهیم. اگر مقدار x بیشتر از مقدار عنصر وسط آرایه بود، جستجو را بر روی نیمه دوم آرایه ادامه می‌دهیم. و اما اگر مقدار x برابر با عنصر وسط آرایه بود، مکان عنصر وسط را باز می‌گردانیم.

[1] binary search

دستورات شرطی

- الگوریتم جستجوی دودویی به صورت زیر است.

```
۱  /* binsearch: find x in v[0] <= v[1] <= ... <= v[n-1] */
۲  int binsearch (int x, int v[], int n)
۳  {
۴    int low, high, mid;
۵    low = 0;
۶    high = n - 1;
```

دستورات شرطی

```
 8    while (low <= high)
 9          {
10                mid = (low + high) / 2;
11                if (x < v[mid])
12                      high = mid + 1;
13                else if (x > v[mid])
14                      low = mid + 1;
15                else          /* found match */
16                      return mid;
17          }
18    return -1;          /* no match */
19  }
```

دستورات شرطی

- ساختار کنترلی switch یک ساختار شرطی چند گزینه‌ای است. در این ساختار شرطی چند شرط وجود دارد که در آن بررسی می‌شود کدام یک از شرط‌ها برقرار هستند. هر کدام از شرط‌ها که برقرار بود، دستورات متعلق به آن شرط اجرا می‌شوند.

- ساختار کنترلی switch به صورت زیر است.

```
switch (expression) {
    case const_expr1 : statement1
    case const_expr2 : statement2
    ...
    default : statementN
}
```

- عبارت expression یک عبارت تشکیل شده از متغیرها و ثابت‌ها ست و const_expr یک مقدار ثابت است. اگر مقدار عبارت expression برابر با const_expr1 بود، statements1 اجرا می‌شود، اگر مقدار عبارت expression برابر با const_expr2 بود، statements2 اجرا می‌شود و در غیراینصورت در حالت پیش‌فرض default دستور statementsN اجرا می‌شود.

دستورات شرطی

- در جایی که ممکن است استفاده از شرط `if` طولانی شود، از `switch` استفاده می‌کنیم.
- در ساختار کنترلی `switch-case` مقدار عبارت `switch` برابر با هر یک از مقادیر `case` ها باشد، دستورات متناظر آن `case` اجرا می‌شود.
- سپس دستورات متناظر با `case` های بعدی اجرا می‌شوند تا جایی که اجرا به دستور `break` برسد و متوقف شود.
- بنابراین برای متوقف کردن بررسی حالت‌ها از دستور `break` استفاده می‌کنیم.

دستورات شرطی

- مثال زیر را در نظر بگیرید. اگر متغیر grade برابر با 'A' باشد عبارت Excellent چاپ می‌شود و ارزیابی به پایان می‌رسد، در غیراینصورت اگر مقدار آن برابر با 'B' باشد، اجرا ادامه پیدا می‌کند، پس اگر مقدار متغیر 'B' یا 'C' باشد، عبارت Well done چاپ می‌شود و به همین ترتیب الی آخر.

```
switch(grade) {
case 'A':
  printf("Excellent!\n");
  break;
case 'B':
case 'C':
  printf("Well done\n");
  break;
```

دستورات شرطی

```
16  case 'D':
17    printf("You passed\n");
18    break;
19  case 'F':
20    printf("Better try again\n");
21    break;
22  default:
23    printf("Invalid grade\n");
24    break;
25  }
```

دستورات شرطی

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که رخداد هر یک از ارقام را در یک رشته و کاراکترهای خط فاصله، خط جدید و ستون جدید را بشمارد.

دستورات شرطی

- این برنامه را می‌توانیم به صورت زیر بنویسیم.

```
۱  #include <stdio.h>
۲  int main()  /* count digits, white space, others */
۳  {
۴    int c, i, nwhite, nother, ndigit[10];
۵    nwhite = nother = 0;
۶    for (i = 0; i < 10; i++)
۷        ndigit[i] = 0;
۸    while ((c = getchar ()) != EOF)
۹      {
۱۰     switch (c)
۱۱       {
۱۲           case '0':
۱۳           case '1':
۱۴           case '2':
۱۵           case '3':
```

دستورات شرطی

```
۱۶            case '4':
۱۷            case '5':
۱۸            case '6':
۱۹            case '7':
۲۰            case '8':
۲۱            case '9':
۲۲                 ndigit[c - '0']++;
۲۳                 break;
۲۴            case ' ':
۲۵            case '\n':
۲۶            case '\t':
۲۷                 nwhite++;
۲۸                 break;
۲۹            default:
۳۰                 nother++;
```

دستورات شرطی

```
۳۱                      break;
۳۲                }
۳۳            }
۳۴        printf ("digits =");
۳۵        for (i = 0; i < 10; i++)
۳۶            printf (" %d", ndigit[i]);
۳۷        printf (", white space = %d, other = %d\n", nwhite, nother);
۳۸        return 0;
۳۹    }
```

دستورات شرطی

- دستور توقف `break` باعث می‌شود کنترل برنامه از `switch` خارج شود. در ساختار `switch` همهٔ شرط‌ها بررسی می‌شوند، بنابراین اگر یک شرط برقرار باشد، همهٔ دستورات حالت‌های بعدی اجرا می‌شوند، مگر اینکه به طور صریح توسط دستور `break` در یکی از شاخه‌های `switch` از آن خارج شویم.
- مزیت این طراحی در زبان سی این است که به برنامه‌نویس اجازه می‌دهد برای چند حالت متفاوت یک دسته دستورات واحد اجرا کند. مشکل این طراحی این است که همیشه وقتی حالت‌ها متمایزند نیاز است بعد از هر حالت دستور `break` نوشته شود.
- توجه کنید با این‌که از نظر منطقی نیازی به قرار دادن دستور `break` بعد از دستورات شاخه `default` نیست، اما بهتر است دستور `break` قرار داده شود، زیرا ممکن است در آینده برنامه‌نویس دیگری تعدادی شاخه به انتهای `switch` اضافه کند و قرار دادن `break` پس از شاخه `default` مانع بروز خطاهای احتمالی می‌شود.
- دستور توقف `break` در ساختارهای حلقه تکرار مانند `while` و `for` و `do` نیز به کار می‌رود و باعث می‌شود کنترل برنامه از حلقه تکرار خارج شود قبل از اینکه حلقه به پایان برسد.

ساختارهای حلقه تکرار

- ساختارهای حلقه [1] برای تکرار بلوکی از دستورات به تعداد معین به کار می‌روند.
- چند نوع ساختار حلقه وجود دارد که عبارتند از حلقه `for` و `while` و `do`.
- ساختار نحوی `while` (مادامی‌که) به صورت زیر است.

```
while(condition) {
        statement-block
}
```

- مادامی‌که مقدار شرط `condition` صحیح است، دستورات `statement-block` اجرا می‌شوند. به عبارت دیگر پس از اتمام اجرای دستورات، دوباره شرط بررسی می‌شود و اگر شرط همچنان درست بود دستورات برای بار دیگر اجرا می‌شوند.

[1] loop structures

ساختارهای حلقه تکرار

- فرض کنید می‌خواهیم یک عدد دهدهی را به معادل دودویی آن تبدیل کنیم.

```
۱  int n;
۲  scanf("%d", &n);
۳  int binary=0;
۴  int r;
۵  int base = 1;
۶  while(n > 0) {
۷      r = n % 2;
۸      n = n / 2;
۹      binary = binary + base*r;
۱۰     base = base * 10;
۱۱ }
۱۲ printf("%d = %d in binary \n", n, binary);
```

ساختارهای حلقه تکرار

- ساختار for (برای) به صورت زیر است.

```
for (init-expr ; condition ; step-expr) {
        statement-block
}
```

- این ساختار معادل است با :

```
init-expr;
while(condition) {
          statement-block
          step-expr;
}
```

ساختارهای حلقه تکرار

- هر یک از سه عبارت در ساختار `for` می‌تواند حذف شوند. اگر `init-expr` حذف شوند، این عبارات می‌تواند قبل از حلقه قرار بگیرند. اگر `step-expr` حذف شود، این عبارت می‌تواند در درون حلقه قرار بگیرند. اگر `condition` حذف شود، شرط حلقه همیشه درست است و خاتمه نمی‌یابد مگر با دستور `break` یا `return`.
- عبارت `for(;;){...}` یک حلقه بینهایت است و فقط با `break` یا `return` خاتمه می‌یابد.
- معمولا وقتی از ساختار `for` استفاده می‌کنیم که نیاز به مقدار دهی اولیه باشد و نیاز به اعمال تغییرات در متغیرها برای تکرار بعدی در حلقه داشته باشیم.

ساختارهای حلقه تکرار

- در مثال زیر به اعمال تغییرات در تکرارهای حلقه و مقدار دهی اولیه نداریم، پس `while` مناسب‌تر است.

```
while ((c = getchar()) == ' ' || c == '\n' || c = '\t')
; /* skip white space characters */
```

- معمولا در هنگام انجام عملیات بر روی عناصر یک آرایه به یک حلقه به صورت `for(i = 0 ; i < n ; i++)` نیاز داریم.

ساختارهای حلقه تکرار

- می‌خواهیم برنامه‌ای بنویسیم که فاکتوریل یک عدد را محاسبه کند.

```
۱ int n;
۲ int fact = 1;
۳ scanf("%d", &n);
۴ for(int i=1; i<=n; i++) {
۵         fact = fact * i;
۶ }
```

ساختارهای حلقه تکرار

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که یک رشته را به عددی که محتوای رشته است تبدیل کنیم. در ابتدا باید همه کاراکترهای خط فاصله را نادیده بگیریم. سپس یک کاراکتر را به عنوان علامت (در صورت وجود) دریافت کنیم و سپس ارقام را به ترتیب از رشته بخوانیم و تبدیل به عدد کنیم. (در کتابخانه استاندارد تابع `strtol` برای تبدیل یک رشته به عدد صحیح طولانی وجود دارد.)

ساختارهای حلقه تکرار

- برنامه تبدیل رشته به عدد به صورت زیر نوشته می‌شود.

```
۱  #include <ctype.h>
۲  /* atoi: convert s to integer; version 2 */
۳  int atoi (char s[])
۴  {
۵    int i, n, sign;
۶    for (i = 0; isspace (s[i]); i++)  /* skip white space */
۷        ;
۸    sign = (s[i] == '-') ? -1 : 1;
۹    if (s[i] == '+' || s[i] == '-')  /* skip sign */
۱۰       i++;
۱۱   for (n = 0; isdigit (s[i]); i++)
۱۲     n = 10 * n + (s[i] - '0');
۱۳   return sign * n;
۱۴ }
```

ساختارهای حلقه تکرار

- می‌خواهیم یک آرایه‌ای از اعداد صحیح را توسط الگوریتم مرتب‌سازی شل [1] مرتب کنیم. این روش مرتب‌سازی در سال ۱۹۵۹ توسط شل ابداع شد. در این روش مرتب‌سازی ابتدا عناصر با فاصله دور مقایسه می‌شوند. پس در گام‌های ابتدایی عناصر با فاصله مرتب می‌شوند و در گام‌های انتهایی عناصر نزدیک به هم مرتب می‌شوند. الگوریتم مرتب‌سازی شل به صورت زیر است.

[1] Shell sort

ساختارهای حلقه تکرار

- سه حلقه تودرتو در این مثال وجود دارند. در حلقهٔ بیرونی فاصلهٔ بین عناصر کنترل می‌شود، و در هر بار این فاصله نصف می‌شود. حلقه میانی بررسی عناصر آرایه را کنترل می‌کند و در حلقه درونی جفت عناصر با فاصله معین را مقایسه و در صورت نیاز آنها را جابجا می‌کند.

```
۱  /* shellsort: sort v[0]...v[n-1] into increasing order */
۲  void shellsort (int v[], int n)
۳  {
۴    int gap, i, j, temp;
۵    for (gap = n / 2; gap > 0; gap /= 2)
۶        for (i = gap; i < n; i++)
۷            for (j = i - gap; j >= 0 && v[j] > v[j + gap]; j -= gap)
۸            {
۹                temp = v[j];
۱۰               v[j] = v[j + gap];
۱۱               v[j + gap] = temp;
۱۲           }
۱۳ }
```

ساختارهای حلقه تکرار

- یکی از عملگرها در زبان سی عملگر ویرگول (`,`) است. در حلقه `for` می‌توان با استفاده از این عملگر چند عبارت مقداردهی اولیه یا چند عبارت اعمال گام قرار داد.

ساختارهای حلقه تکرار

- در مثال زیر وارون یک رشته محاسبه می‌شود.

```
۱  #include <string.h>
۲  /* reverse: reverse string s in place */
۳  void reverse (char s[])
۴  {
۵    int c, i, j;
۶    for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
۷      {
۸        c = s[i];
۹        s[i] = s[j];
۱۰       s[j] = c;
۱۱     }
۱۲ }
```

ساختارهای حلقه تکرار

- دقت کنید که در حلقه `for` عبارات جدا شده با ویرگول از چپ به راست اجرا می‌شوند، ولی عبارات جدا شده توسط ویرگول در آرگومان‌های تابع و تعریف و اعلام متغیرها توسط عملگر ویرگول از چپ به راست اجرا نمی‌شوند، زیرا این ویرگول‌ها عملگر ویرگول نیستند.

ساختارهای حلقه تکرار

- یکی دیگر از ساختارهای حلقه `do-while` است. این ساختار به صورت زیر نوشته می‌شود.

```
do {
      statement-block
} while(condition);
```

- در این ساختار دستورات `statement-block` اجرا می‌شوند، و پس از آن شرط حلقه بررسی می‌شود. این روند ادامه پیدا می‌کند تا وقتی که شرط حلقه برابر با صفر (مقدار نادرست) شود. از این ساختار معمولاً کمتر از `for` و `while` استفاده می‌شود.

ساختارهای حلقه تکرار

- برنامه‌ای بنویسید که اعدادی را از ورودی دریافت کند و تا وقتی که عدد ورودی مخالف صفر است، اعداد را با یکدیگر جمع کند.

```
double number, sum = 0;
do {
    printf("Enter a number: ");
    scanf("%f", &number);
    sum += number;
} while(number != 0.0);
printf("Sum = %.2f", sum);
```

- در اینجا از ساختار `do-while` استفاده می‌کنیم، زیرا لازم است حداقل یک عدد از ورودی دریافت شود.

ساختارهای حلقه تکرار

- در برنامه زیر یک عدد صحیح به رشته تبدیل می‌شود.

```
۱  /* itoa: convert n to characters in s */
۲  void itoa (int n, char s[])
۳  {
۴    int i, sign;
۵    if ((sign = n) < 0)   /* record sign */
۶       n = -n;     /* make n positive */
۷    i = 0;
۸    do {          /* generate digits in reverse order */
۹            s[i++] = n % 10 + '0';     /* get next digit */
۱۰   } while ((n /= 10) > 0);     /* delete it */
۱۱   if (sign < 0)
۱۲      s[i++] = '-';
۱۳   s[i] = '\0';
۱۴   reverse (s);
۱۵ }
```

ساختارهای حلقه تکرار

- در اینجا از ساختار `do-while` استفاده می‌کنیم زیرا لازم است حداقل یک کاراکتر در آرایهٔ `s` باشد، حتی اگر `n` برابر با صفر باشد.

توقف و ادامه

- گاهی نیاز داریم از یک حلقه خارج شویم، قبل از اینکه حلقه به پایان رسیده باشد. دستور break برای خروج از حلقه به کار برده می‌شود.
- در تابع زیر، کاراکترهای خط فاصله و خط جدید و ستون جدید از انتهای یک رشته حذف می‌شوند و این کار ادامه پیدا می‌کند تا جایی‌که یا به ابتدای رشته برسیم یا به یک کاراکتر معمولی برخورد کنیم.

```
۱  /* trim: remove trailing blanks, tabs, newlines */
۲  int trim(char s[])
۳  {
۴    int n;
۵    for (n = strlen(s) - 1; n >= 0; n--)
۶        if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
۷            break;
۸    s[n + 1] = '\0';
۹    return n;
۱۰ }
```

توقف و ادامه

- در برنامهٔ زیر مجموع حداکثر تعداد ۱۰ عدد مثبت دریافت شده از ورودی محاسبه می‌شود، و در صورتی که یکی از اعداد دریافت شده منفی باشد، عملیات خاتمه می‌یابد.

```
int i;
double number, sum = 0.0;
for (i = 1; i <= 10; ++i) {
      printf("Enter number %d: ", i);
      scanf("%f", &number);
      // if the user enters a negative number, break the loop
      if (number < 0.0) {
         break;
      }
      sum += number; // sum = sum + number;
}
printf("Sum = %.2f", sum);
```

توقف و ادامه

- دستور ادامه یا `continue` وقتی به کار می‌رود که می‌خواهیم ادامه دستورات در بدنه حلقه را ادامه ندهیم و اجرای حلقه را با تکرار بعدی ادامه دهیم.
- در حلقه `while` و `do-while` پس از اجرای دستور `continue` شرط حلقه بررسی می‌شود. ما در حلقه `for` پس از اجرای دستور `continue` ابتدا دستور افزایش حلقه اجرا می‌شود و پس از آن شرط حلقه بررسی می‌شود.
- دستور `break` هم در حلقه‌ها استفاده می‌شود و هم در `switch` ، اما دستور `continue` تنها در حلقه‌ها استفاده می‌شود.

توقف و ادامه

- در برنامهٔ زیر مجموع دقیقا ۱۰ عدد مثبت دریافت شده از ورودی محاسبه می‌شود، و در صورتی که یکی از اعداد دریافت شده منفی باشد، آن عدد به مجموع اضافه نمی‌شود و به جای آن یک عدد دیگر دریافت می‌شود.

```
۱  int i = 1;
۲  double number, sum = 0.0;
۳  while (i <= 10) {
۴        printf("Enter number %d: ", i);
۵        scanf("%f", &number);
۶        // if the user enters a negative number, continue the loop
۷        if (number < 0.0) {
۸           continue;
۹        }
۱۰       sum += number; // sum = sum + number;
۱۱       i++;
۱۲ }
۱۳ printf("Sum = %.2f", sum);
```

توقف و ادامه

- در مثال زیر تنها عناصر غیر منفی در حلقه پردازش می‌شوند.

```
۱ for (i = 0; i < n; i++)
۲     if (a[i] < 0)    /* skip negative elements */
۳         continue
۴     ... /* do positive elements */
```