

به نام خدا

مبانی برنامه نویسی

آرش شفیعی



ساختارهای کنترلی

ساختارهای کنترلی

- ساختارهای کنترلی¹ در زبان‌های برنامه‌نویسی جهت اعمال تصمیمات بر روی ترتیب اجرای دستورات به کار می‌روند.
- یک دستور² در یک زبان برنامه‌نویسی با علامت نقطه ویرگول (;) پایان می‌یابد. یک دستور می‌تواند یک عملیات انتساب، فراخوانی یک تابع، تعریف³ متغیر یا اعلام⁴ متغیر یا تابع باشد.
- علامت آکولاد باز و بسته { } برای گروه بندی دستورات استفاده می‌شوند. یک گروه از دستورات را یک بلوک⁵ 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.

- شرط در دستور شرطی یک عبارت منطقی است، یعنی عبارتی که یک مقدار درست یا نادرست باز می‌گرداند.
- در یک عبارت منطقی برای تشکیل یک عبارت منطقی از عملگرهای منطقی و رابطه‌ای استفاده می‌کنیم.
برای مثال $(x \geq 10 \ \&\& \ x \leq 20)$ ، $(x < 10 \ || \ x > 20)$ ، $(c == 'q')$ ، $(x \ \& \ y) \neq 0$ عبارات منطقی هستند.
- مقدار درست¹ در زبان سی برابر با مقدار غیر صفر است و مقدار نادرست² 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.

```
۱ if (n > 0)
۲     if (a > b)
۳         z = a;
۴     else
۵         z = b;
```

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

```
۱ if (n > 0){
۲     if (a > b)
۳         z = a;
۴ }
۵ else
۶     z = b;
```

- برنامه زیر را در نظر بگیرید. گرچه برنامه‌نویس با دندان‌گذاری¹ 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 else if را می‌توانیم به صورت زیر بنویسیم.

```
۱ if (condition1) {  
۲     statement-block1  
۳ } else if (condition2) {  
۴     statement-block2  
۵ }  
۶ ...  
۷ else {  
۸     statement-blockN  
۹ }
```

- 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 شرطها 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.

- فرض کنید v یک آرایه 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 دودویی¹ 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

- 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;
```

```
۸  while (low <= high)
۹      {
۱۰         mid = (low + high) / 2;
۱۱         if (x < v[mid])
۱۲             high = mid + 1;
۱۳         else if (x > v[mid])
۱۴             low = mid + 1;
۱۵         else          /* found match */
۱۶             return mid;
۱۷     }
۱۸     return -1;          /* no match */
۱۹ }
```

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.

```
۱ switch (expression) {  
۲     case const_expr1 : statement1  
۳     case const_expr2 : statement2  
۴     ...  
۵     default : statementN  
۶ }
```

- عبارت expression یک عبارت تشکیل شده از متغیرها و ثابت‌ها است 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.

- 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.

- 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.

```
۱ switch(grade) {  
۲ case 'A':  
۳     printf("Excellent!\n");  
۴     break;  
۵ case 'B':  
۶ case 'C':  
۷     printf("Well done\n");  
۸     break;
```

```
۱۶ case 'D':  
۱۷     printf("You passed\n");  
۱۸     break;  
۱۹ case 'F':  
۲۰     printf("Better try again\n");  
۲۱     break;  
۲۲ default:  
۲۳     printf("Invalid grade\n");  
۲۴     break;  
۲۵ }
```

- 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.

- 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':
```

```
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++;
```

```
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 همه شرط‌ها بررسی می‌شوند، بنابراین اگر یک شرط برقرار باشد، همه 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 این که 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 (مادامی که) به صورت زیر است.

```
۱ while(condition) {  
۲     statement-block  
۳ }
```

- مادامی که 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

- 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 (به ازای) به صورت زیر است.

```
۱ for (init-expr ; condition ; step-expr) {  
۲     statement-block  
۳ }
```

- این ساختار معادل است با :

```
۱ init-expr;  
۲ while(condition) {  
۳     statement-block  
۴     step-expr;  
۵ }
```


- 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.
- معمولاً 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.

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

```
۱ while ((c = getchar()) == ' ' || c == '\n' || c == '\t')  
۲ ; /* 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.

- 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 }
```

- 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 }
```

- می‌خواهیم یک آرایه‌ای از اعداد صحیح را توسط الگوریتم مرتب‌سازی شل¹ 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 حلقه بیرونی فاصله 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 }
```

- 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.

- 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 }
```

- 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.

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

```
۱ do {  
۲     statement-block  
۳ } 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.

```
۱ double number, sum = 0;
۲ do {
۳     printf("Enter a number: ");
۴     scanf("%f", &number);
۵     sum += number;
۶ } while(number != 0.0);
۷ 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 }
```

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

- 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 برنامه 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.

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

- 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 جایی که 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 }
```

- دستور ادامه یا `continue` وقتی به کار می رود که می خواهیم 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 می شود.
- در حلقه `for` پس از اجرای دستور `continue` ابتدا دستور افزایش حلقه اجرا می شود و پس از آن شرط حلقه بررسی می شود.
- دستور `break` هم در حلقه ها استفاده می شود و هم در `switch` ، اما دستور `continue` تنها در حلقه ها استفاده می شود.

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

```
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);
```

- 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 */
```

- برنامه‌ای بنویسید که عدد n را دریافت کند و زوج و فرد بودن آن را تعیین کند.

```

۱ int main() {
۲     int n;
۳     printf("Enter a number: ");
۴     scanf("%d", &n);
۵     if ( n%2 == 0)
۶         printf("%d is even.\n", n);
۷     else
۸         printf("%d is odd.\n", n);
۹ }
```

- برنامه‌ای بنویسید که عدد n را دریافت کند و مجموع $1 + 2 + \dots + n$ را بازگرداند.

```

۱  int main() {
۲      int n;
۳      int sum = 0;
۴      printf("Enter a number: ");
۵      scanf("%d", &n);
۶      if (n < 1) {
۷          printf("Invalid number.\n");
۸          return 0;
۹      }
۱۰     for(int i=1; i<=n; i++)
۱۱         sum += i;
۱۲     printf("1 + ... + %d = %d\n", n, sum);
۱۳     return 0;
۱۴ }
```

- برنامه‌ای بنویسید که عدد n را دریافت کند و $n!$ را بازگرداند.

```

۱  int main() {
۲      int n;
۳      int fact = 1;
۴      printf("Enter a number: ");
۵      scanf("%d", &n);
۶      if (n < 0) {
۷          printf("Invalid number.\n");
۸          return 0;
۹      }
۱۰     for(int i=1; i<=n; i++)
۱۱         fact *= i;
۱۲     printf("%d! = %d\n", n, fact);
۱۳     return 0;
۱۴ }
```

- برنامه‌ای بنویسید که عدد n را دریافت کرده و عدد فیبوناچی n ام را بازگرداند.

```
۱ int main() {  
۲     int n;  
۳     printf("Enter a number: ");  
۴     scanf("%d", &n);  
۵     if (n < 0) {  
۶         printf("Invalid number.\n");  
۷         return 0;  
۸     }  
۹     if (n==0 || n==1) {  
۱۰        printf("Fibonacci %d is %d.\n", n, n);  
۱۱        return 0;  
۱۲    }
```

- برنامه‌ای بنویسید که عدد n را دریافت کرده و عدد فیبوناچی n ام را بازگرداند.

```

۱۳     int prev = 0, curr = 1, next;
۱۴     for(int i=2; i<=n; i++) {
۱۵         next = prev + curr;
۱۶         prev = curr;
۱۷         curr = next;
۱۸     }
۱۹     printf("Fibonacci %d is %d.\n", n, next);
۲۰     return 0;
۲۱ }
```

- برنامه‌ای بنویسید که انتگرال $f(x) = x^2$ را به طور تقریبی از ۰ تا n محاسبه کند.

```

۱  int main() {
۲      int n;
۳      printf("Enter a number: ");
۴      scanf("%d", &n);
۵      if (n < 0) {
۶          printf("Invalid number.\n");
۷          return 0;
۸      }
۹      float integral = 0;
۱۰     float delta = 0.001;
۱۱     for (float i = 0; i <= n; i=i+delta)
۱۲         integral += i * i * delta;
۱۳     printf("Integral of x^2 from 0 to %d is %d.\n", n, integral);
۱۴     return 0;
۱۵ }

```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، و ارقام آن را از سمت راست به چپ چاپ کند.

```

۱ int main() {
۲     int n, r;
۳     printf("Enter a number: ");
۴     scanf("%d", &n);
۵     if (n < 0) {
۶         printf("Invalid number.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، و ارقام آن را از سمت راست به چپ چاپ کند.

```

۹  while(n > 0) {
۱۰     r = n % 10;
۱۱     printf("%d ", r);
۱۲     n = n/10;
۱۳ }
۱۴ printf("\n");
۱۵ return 0;
۱۶ }
```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، و مقلوب آن را محاسبه کند.

```

۱ int main() {
۲     int num, invert = 0, n, r;
۳     printf("Enter a number: ");
۴     scanf("%d", &num);
۵     if (num < 0) {
۶         printf("Invalid number.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، و مقلوب آن را محاسبه کند.

```

۹      n = num;
۱۰     while(n > 0) {
۱۱         r = n % 10;
۱۲         invert = (10*invert) + r;
۱۳         n = n/10;
۱۴     }
۱۵     printf("Inversion of %d is %d\n", num, invert);
۱۶     return 0;
۱۷ }
```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، آن را به مبنای ۲ تبدیل کند.

```

۱ int main() {
۲     int num, n, r, binary = 0, base = 1;
۳     printf("Enter a number: ");
۴     scanf("%d", &num);
۵     if (num < 0) {
۶         printf("Invalid number.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که عدد n را دریافت کرده، آن را به مبنای ۲ تبدیل کند.

```
۹    n = num;
۱۰    while(n > 0) {
۱۱        r = n % 2;
۱۲        n = n / 2;
۱۳        binary += base * r;
۱۴        base *= 10;
۱۵    }
۱۶    printf("%d = %d in binary \n", num, binary);
۱۷    return 0;
۱۸ }
```

- برنامه‌ای بنویسید که یک عدد دودویی را دریافت کرده، به مبنای ۱۰ تبدیل کند.

```
۱ int main() {
۲     int num, n, r, decimal = 0, base = 1;
۳     printf("Enter a number: ");
۴     scanf("%d", &num);
۵     if (num < 0) {
۶         printf("Invalid number.\n");
۷         return 0;
۸     }
```

- برنامه‌ای بنویسید که یک عدد دودویی را دریافت کرده، به مبنای ۱۰ تبدیل کند.

```
۹   n = num;
۱۰   while(n > 0) {
۱۱       r = n % 10;
۱۲       n = n / 10;
۱۳       decimal += base * r;
۱۴       base *= 2;
۱۵   }
۱۶   printf("%d = %d in decimal \n", num, decimal);
۱۷   return 0;
۱۸ }
```

- برنامه‌ای بنویسید که دو عدد را دریافت کرده، بزرگترین مقسوم علیه مشترک آنها را محاسبه کند.

```

۱ int main() {
۲     int n, m, k, r;
۳     printf("Enter two numbers: ");
۴     scanf("%d %d", &n, &m);
۵     if (n < 1 || m < 1) {
۶         printf("Invalid numbers.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که دو عدد را دریافت کرده، بزرگترین مقسوم علیه مشترک آنها را محاسبه کند.

```

۹   if (n > m) {
۱۰       k = m;
۱۱       m = n;
۱۲       n = k;
۱۳   }
۱۴   printf("Greatest Common Divisor of %d and %d is ", m, n);
۱۵   while ((r = m%n) != 0) {
۱۶       m = n;
۱۷       n = r;
۱۸   }
۱۹   printf("%d .\n", n);
۲۰   return 0;
۲۱ }

```

- برنامه‌ای بنویسید که دو عدد را دریافت کرده، کوچک‌ترین مضرب مشترک آنها را محاسبه کند.

```

۱ int main() {
۲     int n, m, k, d, lcm;
۳     printf("Enter two numbers: ");
۴     scanf("%d %d", &n, &m);
۵     if (n < 1 || m < 1) {
۶         printf("Invalid numbers.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که دو عدد را دریافت کرده، کوچک‌ترین مضرب مشترک آنها را محاسبه کند.

```
۹    if (n > m) {
۱۰        k = m;
۱۱        m = n;
۱۲        n = k;
۱۳    }
۱۴    d=0;
۱۵    do {
۱۶        lcm = ++d * m;
۱۷    } while(lcm % n != 0);
۱۸    printf("Least Common Multiple of %d and %d is %d .\n", m, n, lcm);
۱۹    return 0;
۲۰ }
```

- برنامه‌ای بنویسید که یک عدد را دریافت کرده، بررسی کند آیا عدد دریافت شده اول است یا خیر.

```

۱ int main() {
۲     int n;
۳     printf("Enter a number: ");
۴     scanf("%d", &n);
۵     if (n == 1) {
۶         printf("1 is not prime nor composite.\n");
۷         return 0;
۸     }

```

- برنامه‌ای بنویسید که یک عدد را دریافت کرده، بررسی کند آیا عدد دریافت شده اول است یا خیر.

```
۹   for(int i=2; i <= n/2; i++) {  
۱۰       if (n%i == 0) {  
۱۱           printf("%d is composite.\n", n);  
۱۲           return 0;  
۱۳       }  
۱۴   }  
۱۵   printf("%d is prime.\n", n);  
۱۶   return 0;  
۱۷ }
```

- برنامه‌ای بنویسید که مجموع سری $1 + 1/2 + 1/4 + 1/8 + \dots$ را محاسبه کند.

```

۱ int main() {
۲     float sum = 0, term = 1, threshold = 0.00001;
۳     while (term > threshold) {
۴         sum += term;
۵         term /= 2;
۶     }
۷     printf("1+ 1/2 + 1/4 + 1/8 + ... = %.2f", sum);
۸     return 0;
۹ }
```

- برنامه‌ای بنویسید که یک آرایه از اعداد و یک عدد را دریافت کرده و عدد مورد نظر را در آرایه جستجو کند.

```

۱  int main() {
۲      int n, x, flag = 0;
۳      printf("Enter a number: ");
۴      scanf("%d", &n);
۵      if (n < 1) {
۶          printf("Invalid number.\n");
۷          return 0;
۸      }
۹      int array[n];
۱۰     for (int i=0; i<n; i++) {
۱۱         printf("Enter array[%d] : ", i);
۱۲         scanf("%d", &array[i]);
۱۳     }

```

- برنامه‌ای بنویسید که یک آرایه از اعداد و یک عدد را دریافت کرده و عدد مورد نظر را در آرایه جستجو کند.

```

۱۴ printf("Enter a number to search: ");
۱۵ scanf("%d", &x);
۱۶ for(int i=0; i<n; i++)
۱۷     if (array[i] == x) {
۱۸         printf("array[%d] = %d \n", i, x);
۱۹         flag = 1;
۲۰     }
۲۱     if (!flag)
۲۲         printf("could not find %d in array\n", x);
۲۳     return 0;
۲۴ }
```

- برنامه‌ای بنویسید که یک آرایه از اعداد را دریافت کرده و به ترتیب صعودی مرتب کند.

```

۱  int main() {
۲      int n, k;
۳      printf("Enter a number: ");
۴      scanf("%d", &n);
۵      if (n < 1) {
۶          printf("Invalid number.\n");
۷          return 0;
۸      }
۹      int array[n];
۱۰     for (int i=0; i<n; i++) {
۱۱         printf("Enter array[%d] : ", i);
۱۲         scanf("%d", &array[i]);
۱۳     }

```

- برنامه‌ای بنویسید که یک آرایه از اعداد را دریافت کرده و به ترتیب صعودی مرتب کند.

```

۱۴     for(int i=0; i<n-1; i++) {
۱۵         for(int j=i+1; j<n; j++) {
۱۶             if (array[i] > array[j]) {
۱۷                 k = array[i];
۱۸                 array[i] = array[j];
۱۹                 array[j] = k;
۲۰             }
۲۱         }
۲۲     }
۲۳     for(int i=0; i<n; i++)
۲۴         printf("array[%d] = %d \n", i, array[i]);
۲۵     return 0;
۲۶ }
```
