به نام خدا

مبانی برنامه‌نویسی

آرش شفیعی

توابع و ساختارهای برنامه

توابع

- توسط توابع می‌توان محاسبات طولانی را به قسمت‌های کوچکتر تقسیم کرد. این تقسیم‌بندی کمک می‌کند که یک برنامه خوانایی بیشتری داشته باشد، علاوه بر این که وقتی یک محاسبات معین توسط یک واحد مشخص تعریف شود، از آن واحد محاسباتی می‌توان در برنامه‌های دیگر نیز استفاده کرد.
- یک واحد محاسباتی توسط یک نام مشخص می‌شود. واحد محاسباتی برای انجام محاسبات تعدادی ورودی دریافت می‌کند و تعدادی خروجی باز می‌گرداند. چنین واحد محاسباتی در زبان سی یک تابع نامیده می‌شود.
- یک تابع یک واحد محاسباتی است که توسط یک نام مشخص می‌شود و صفر یا تعدادی ورودی دریافت کرده و صفر یا یک خروجی باز می‌گرداند.
- یک تابع تشکیل شده است از یک نوع داده بازگشتی، نام تابع، متغیرهای ورودی که پارامتر نامیده می‌شوند و بدنهٔ تابع.

توابع

- فرض کنید می‌خواهیم تابعی بنویسیم که مقدار x^y را محاسبه می‌کند. در واقع این تابع دو متغیر `x` و `y` را دریافت می‌کند و مقداری را از تابع `power(x,y)` بازمی‌گرداند.

توابع

- این تابع به صورت زیر تعریف می‌شود.

```
/* power: raise base to n-th power; n >= 0 */
int power (int base, int n)
{
  int i, p;
  p = 1;
  for (i = 1; i <= n; ++i)
    p = p * base;
  return p;
}
```

توابع

- سپس این تابع را به صورت زیر فراخوانی می‌کنیم.

```
۱ #include <stdio.h>
۲ /* test power function */
۳ int main ()
۴ {
۵   int i;
۶   for (i = 0; i < 10; ++i)
۷     printf ("%d %d %d\n", i, power (2, i), power (-3, i));
۸   return 0;
۹ }
```

توابع

- برنامهٔ سی از تعداد زیادی تابع تشکیل شده است که هر یک وظیفه مشخصی دارند. این توابع می‌توانند در یک یا چند فایل مختلف قرار بگیرند. هر فایل به طور جداگانه کامپایل می‌شود و فایل‌های کامپایل شده توسط لینکر به یکدیگر متصل می‌شوند. در بسیاری از مواقع توابع مورد نیاز توسط افراد دیگر نوشته شده‌اند و فایل‌های کامپایل شده برای استفاده در اختیار ما قرار می‌گیرند. مجموعه توابعی که برای یک کاربرد خاص تعریف شده و برای استفاده در اختیار دیگر برنامه‌ها قرار می‌گیرند، یک کتابخانه نامیده می‌شود.

توابع

- یک تابع را می‌توانیم قبل از تعریف کردن [1] اعلام کنیم [2].
- برای اعلام تابع باید نوع خروجی، نام تابع و پارامترهای ورودی و نوع آنها مشخص شود. به مجموعهٔ نوع، نام، و پارامترهای ورودی تابع، پروتوتایپ تابع گفته می‌شود، پس در اعلام تابع تنها پروتوتایپ تابع [3] مشخص می‌شود. این اعلام جهت اطلاع کامپایلر است از اینکه چنین تابعی وجود دارد.
- یک تابع پس از اعلام باید تعریف شود. در تعریف تابع علاوه بر مشخص کردن نام و نوع ورودی‌ها و خروجی تابع باید مجموعه دستورات تابع در یک بلوک تعریف شوند. تعریف و اعلام یک تابع باید با یکدیگر همخوانی داشته باشند.
- در اعلام تابع، الزامی به ذکر نام پارامترها وجود ندارد. بنابراین می‌توانیم بنویسیم:

```
int power (int, int);
```

[1] definition
[2] declaration
[3] function prototype

توابع

- اگر یک تابع، قبل از فراخوانی تعریف شده باشد، نیازی به اعلام تابع نداریم. اما اگر تابع بعد از فراخوانی تعریف شده باشد یا در یک فایل دیگر قرارگرفته باشد، باید قبل از فراخوانی آن را اعلام کنیم.

```c
#include <stdio.h>
int power (int m, int n);
/* test power function */
int main ()
{
  int i;
  for (i = 0; i < 10; ++i)
    printf ("%d %d %d\n", i, power (2, i), power (-3, i));
  return 0;
}
```

توابع

- یک تابع در یک فایل تعریف می‌شود و نمی‌توان یک تابع را به دو قسمت تقسیم کرد. وقتی یک تابع در یک فایل جداگانه از تابع فراخوانی کننده قرار می‌گیرند، برای استفاده از آن باید فایل‌های آبجکت ساخته شده به یکدیگر پیوند داده شوند.
- تابع `power` در مثال قبل دارای دو پارامتر ورودی از نوع عدد صحیح و یک مقدار بازگشتی از نوع عدد صحیح است.
- متغیرهایی که در تعریف تابع به کار می‌روند را پارامتر و متغیرهایی که در فراخوانی تابع به تابع ارسال می‌شوند را آرگومان می‌نامیم.
- کلمهٔ کلیدی `return` برای بازگرداندن یک مقدار از تابع به کار می‌رود.

توابع

- یک تابع می‌تواند هر نوعی از جمله عدد صحیح، اعشاری، و کاراکتر برگرداند.
- نوع `void` برای یک متغیر به معنی نوع تهی است. تابعی که هیچ متغیری بازنمی‌گرداند، مقدار بازگشتی آن از نوع `void` است.
- یک استثنا در نوع بازگشتی یک تابع وجود دارد و آن این است که نوع بازگشتی تابع نمی‌تواند آرایه باشد.

توابع

- تابع `main` هم مانند توابع دیگر یک مقدار باز می‌گرداند. مقدار صفر نشان دهنده این است که برنامه بدون خطا متوقف شده است.
- مقداری که توسط تابع `main` بازگردانده می‌شود به محیط اجرا کنندهٔ آن بازگردانده می‌شود. محیط اجرا کننده می‌تواند تصمیم بگیرد که با مقدار بازگردانده شده چه عملیاتی انجام دهد.
- می‌توانیم از تابع `main` مقادیر غیر صفر نیز بازگردانیم، که به معنای خاتمه برنامه با خطا است.
- برای مثال می‌توانیم برنامه دیگری بنویسیم که یک برنامه را راه‌اندازی کند و در صورتی که مقدار بازگردانده شده غیر صفر بود، عملیاتی برای مدیریت خطا انجام دهد.

توابع *

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که در ورودی تعدادی خط را دریافت کرده و اگر در یک خط یک رشته داده شده وجود داشت، آن خط را در خروجی چاپ کند.
- پس الگوریتم این برنامه بدین صورت است : تا وقتی که خط بعدی وجود دارد، خط بعدی را از ورودی استاندارد بخوان و اگر آن خط شامل رشتهٔ داده شده است، آن خط را چاپ کن.

توابع *

- گرچه همهٔ دستورات را می‌توانیم در بدنه تابع اصلی `main` قرار دهیم، راه بهتری این است که برنامه را طوری ساختاربندی کنیم که دستورات بدنهٔ اصلی کم و برنامه اصلی خوانا باشد. طراحی بهتر این است جزئیات را در توابع قرار دهیم و در بدنهٔ اصلی تنها بنویسیم چه‌کاری باید انجام نشود و به این که چگونه باید انجام شود.
- برای دریافت خط بعدی قبلا تابع `getline` را نوشتیم. برای چاپ یک جمله نیز از تابع `printf` استفاده می‌کنیم. تنها تابع مورد نیاز، تابعی است که یک رشته (یک خط) را دریافت کند و یک زیررشته (یک الگو) را در آن جستجو کند.
- می‌توانیم تابعی بنویسیم که به عنوان ورودی اول یک جمله و به عنوان ورودی دوم یک الگو را دریافت کرده و در جمله، الگوی مورد نظر را جستجو کند. این تابع در صورتی که الگوی ورودی را پیدا کرد اندیس کاراکتر اول الگوی یافت شده را در جمله باز می‌گرداند و در غیر اینصورت مقدار ۱- را باز می‌گرداند.
- در کتابخانهٔ استاندارد زبان سی تابعی به نام `strstr` وجود دارد که عملیات مشابه انجام می‌دهد.

توابع *

- برنامه جستجوی رشته در ورودی استاندارد به صورت زیر نوشته می‌شود.

```
۱ #include <stdio.h>
۲ #define MAXLINE 1000      /* maximum input line length */
۳ int _getline (char line[], int max);
۴ int strindex (char source[], char searchfor[]);
۵ char pattern[] = "ould";    /* pattern to search for */
```

توابع *

```
۶  /* find all lines matching pattern */
۷  int main ()
۸  {
۹    char line[MAXLINE];
۱۰   int found = 0;
۱۱   while (_getline (line, MAXLINE) > 0)
۱۲     if (strindex (line, pattern) >= 0)
۱۳       {
۱۴     printf ("%s", line);
۱۵     found++;
۱۶       }
۱۷   return found;
۱۸ }
```

توابع *

```
۲۰ /* getline: get line into s, return length */
۲۱ int _getline (char s[], int lim)
۲۲ {
۲۳   int c, i;
۲۴   i = 0;
۲۵   while (--lim > 0 && (c = getchar ()) != EOF && c != '\n')
۲۶     s[i++] = c;
۲۷   if (c == '\n')
۲۸      s[i++] = c;
۲۹   s[i] = '\0';
۳۰   return i;
۳۱ }
```

توابع *

```
۳۳ /* strindex: return index of t in s, -1 if none */
۳۴ int strindex (char s[], char t[])
۳۵ {
۳۶   int i, j, k;
۳۷   for (i = 0; s[i] != '\0'; i++)
۳۸     {
۳۹         for (j = i, k = 0; t[k] != '\0' && s[j] == t[k]; j++, k++)
۴۰         ;
۴۱       if (k > 0 && t[k] == '\0')
۴۲          return i;
۴۳     }
۴۴   return -1;
۴۵ }
```

توابع

- همانطور که دیدیم یک تابع توسط نوع خروجی، نام تابع، پارامترهای ورودی (متغیرهای ورودی) و نوع آنها که توسط کاما از یکدیگر جدا می‌شوند، و بدنه تابع تعریف می‌شود.
- بدنهٔ تابع می‌تواند تهی باشد، بنابراین کوچک‌ترین تابع به صورت `void f() {}` تعریف می‌شود. اگر نوع خروجی تابع تعیین نشود، نوع پیش فرض `int` است، ولی در کامپایلرهای امروزی پیام هشدار صادر می‌شود و نوع خروجی تابع باید مشخص شود.
- توابع با یکدیگر توسط پارامترها ارتباط برقرار می‌کنند و یک تابع توسط `return` یک مقدار باز می‌گرداند. وقتی یک تابع عبارت `expression` را باز می‌گرداند، نوع مقدار بازگشتی به نوع بازگشتی تابع تبدیل می‌شود.
- دستور `return` بدون تعیین عبارت بازگردانده شده، باعث خروج از تابع بدون بازگرداندن مقدار می‌شود.
- همچنین تابعی که یک تابع دیگر را فراخوانی می‌کند، می‌تواند مقدار بازگشتی تابع فراخوانی شده را نادیده بگیرد.

توابع

- توابع را می‌توانیم در فایل‌های جداگانه قرار دهیم و در هنگام کامپایل باید همهٔ فایل‌های مورد نیاز را مشخص کنیم. برای مثال می‌توانیم تابع محاسبهٔ توان را در یک فایل جداگانه به نام `power.c` قرا دهیم.
- در هنگام کامپایل توسط `gcc` برنامه را به صورت `gcc main.c power.c` کامپایل می‌کنیم.
- همچنین اگر فایل `power.c` قبلاً کامپایل شده بود و می‌خواستیم آنها را به برنامه اصلی پیوند دهیم، می‌توانستیم به صورت `gcc main.c power.o` نیز برنامه را کامپایل کنیم.

توابع *

- در برنامه‌ای که نوشتیم همهٔ توابع را در یک فایل قرار دادیم. می‌توانیم توابع را در فایل‌های جداگانه قرار دهیم و در هنگام کامپایل باید همهٔ فایل‌های مورد نیاز را مشخص کنیم. برای مثال برنامه قبل را می‌توانستیم به صورت `gcc main.c getline.c strindex.c` کامپایل کنیم.
- همچنین اگر فایل‌های `getline.c` و `strindex.c` قبلاً کامپایل شده بودند و می‌خواستیم آنها را به برنامه اصلی پیوند دهیم، می‌توانستیم به صورت `gcc main.c getline.o strindex.o` نیز برنامه را کامپایل کنیم.

فراخوانی با مقدار

- در زبان سی فراخوانی توابع به طور پیش فرض با مقدار است، بدین معنی که مقدار آرگومان‌ها در مقدار پارامترها کپی می‌شوند و تنها مقدار آرگومان‌ها در دسترس است نه مکان حافظه و آدرس آنها بنابراین متغیرهای ارسال شده به عنوان آرگومان در بدنه تابع در دسترس نیستند.
- فراخوانی با مقدار [1] درکنار فراخوانی با ارجاع [2] دو نوع فراخوانی در زبان سی هستند. فراخوانی به طور پیش‌فرض با مقدار است.
- ارسال متغیرها با نوع‌های معمولی با مقدار است، اما وقتی آرایه‌ها به توابع ارسال می‌شوند، ارسال با ارجاع است.

[1] call by value
[2] call by reference

فراخوانی با مقدار

- در مثال زیر، مقدار n در تابع کاهش می‌یابد ولی از آنجایی که n یک کپی از آرگومان دوم ارسال شده به تابع است، مقدار آرگومان دوم تغییر نخواهد کرد.

```
۱  /* power: raise base to n-th power; n >= 0; version 2 */
۲  int power (int base, int n)
۳  {
۴    int p;
۵    for (p = 1; n > 0; --n)
۶      p = p * base;
۷    return p;
۸  }
۹  int main () {
۱۰   int b = 2, n = 10;
۱۱   int x = power(b, n);
۱۲   // the value of n does not change after the call
۱۳ }
```

فراخوانی با مقدار

- اما تابع زیر مقدار ورودی تابع را تغییر نمی‌دهد.

```
۱ void change(int n) {
۲     n*=2;
۳ }
۴ int main () {
۵     int x = 4;
۶     change(x);
۷     printf("x = %d\n", x);
۸ }
```

فراخوانی با ارجاع

- وقتی نیاز داشته باشیم مقدار یک آرگومان را تغییر دهیم باید از فراخوانی با ارجاع استفاده کنیم. در فراخوانی با ارجاع پارامترها به صورت اشاره‌گر تعریف می‌شوند.
- وقتی یک آرایه به عنوان آرگومان به یک تابع ارسال می‌شود، فراخوانی با ارجاع است و نام آرایه به آدرس اولین عنصر آرایه که به عنوان آرگومان ارسال شده است اشاره می‌کند. عناصر یک آرایه از آرگومان به پارامتر کپی نمی‌شوند.

فراخوانی با ارجاع

- بنابراین تابع زیر مقدار عناصر آرایه را تغییر می‌دهد.

```
۱  void change(int arr[], int n) {
۲      for (int i=0; i<n; i++)
۳          arr[i]*=2;
۴  }
۵  int main () {
۶      int x[] = {1,2,3,4};
۷      change(x, 4);
۸      for (int i=0; i<4; i++)
۹          printf("x[%d] = %d\n", i, x[i]);
۱۰ }
```

توابع

- می‌خواهیم تابعی بنویسیم که یک رشته را در یک رشتهٔ دیگر کپی کند.

توابع

- این برنامه را به صورت زیر می‌توانیم بنویسیم.

```
۱  #include <stdio.h>
۲  #define MAXLEN 1000 /* maximum string length */
۳  void copy (char to[], char from[]);
۴  int main () {
۵    char target[MAXLEN];
۶    char source[MAXLEN] = "hello";
۷    copy (target, source);
۸    printf ("%s\n", target);
۹    return 0;
۱۰ }
```

توابع

```
/* copy: copy 'from' into 'to'; assume string 'to' is big enough */
void copy(char to[], char from[])
{
  int i;
  i = 0;
  while ((to[i] = from[i]) != '\0')
    ++i;
}
```

حوزهٔ تعریف

- در برنامهٔ قبل متغیرهای `source` و `target` را در تابع `main` تعریف کردیم. این متغیرها فقط در تابع `main` تعریف شده‌اند و بیرون از تابع دسترسی به آنها امکان پذیر نیست.
- متغیرهایی که در یک تابع تعریف می‌شوند را متغیرهای محلی [1] تابع می‌نامیم. به عبارت دیگر این متغیرها تنها در حوزهٔ تابع [2] تعریف شده‌اند.
- یک متغیر محلی با فراخوانی تابع تعریف می‌شود و در حافظه قرار می‌گیرد و به محض اتمام اجرای تابع از حافظه حذف می‌شود و مقدار خود را از دست می‌دهد و دسترسی به آن امکان‌پذیر نیست. بنابراین در ابتدای تابع نیاز به مقداردهی اولیه این متغیرهای محلی داریم.

[1] local variables
[2] scope

حوزهٔ تعریف

- یک متغیر را می‌توانیم بیرون از توابع نیز تعریف کنیم. چنین متغیرهایی را متغیر عمومی [1] می‌نامیم. به متغیرهای عمومی متغیرهای خارجی [2] نیز گفته می‌شود.
- یک متغیر عمومی با شروع برنامه تعریف می‌شود و با اتمام برنامه از بین می‌رود بنابراین مقدار خود را در طول برنامه نگه‌می‌دارد.
- وقتی یک متغیر خارج از یک تابع تعریف شده باشد، می‌توانیم با استفاده از کلمهٔ کلیدی `extern` آن را اعلام کنیم.

[1] global variable
[2] external variable

حوزهٔ تعریف

- در مثال زیر از متغیرهای عمومی به جای متغیرهای محلی استفاده شده است.

```
۱  #include <stdio.h>
۲  #define MAXLEN 1000 /* maximum string length */
۳  char target[MAXLEN];
۴  char source[MAXLEN] = "hello";
۵  void copy ();
۶  int main () {
۷    extern char target[];
۸    extern char source[];
۹    copy ();
۱۰   printf ("%s\n", target);
۱۱   return 0;
۱۲ }
```

حوزهٔ تعریف

```
۱۳ /* copy: specialized version */
۱۴ void copy()
۱۵ {
۱۶   int i = 0;
۱۷   extern char target[], source[];
۱۸   while ((target[i] = source[i]) != '\0')
۱۹       ++i;
۲۰ }
```

حوزهٔ تعریف

- اگر تعریف متغیر قبل از تابع استفاده‌کننده از آن صورت گرفته باشد، نیازی به اعلام متغیر با کلمه `extern` نداریم. همچنین اگر یک برنامه از چند فایل تشکیل شده باشد و یک متغیر در یک فایل تعریف شده باشد و بخواهیم در یک فایل دیگر از آن استفاده کنیم، نیاز داریم با استفاده از کلمهٔ `extern` آن را اعلام کنیم.
- توابعی مانند `printf` که از آنها استفاده کردیم، در فایل‌های دیگر تعریف شده‌اند و بنابراین در ابتدای برنامه فایل `stdio.h` را ضمیمه کردیم که در آن اعلام تابع `printf` وجود دارد. این توابع ورودی و خروجی توسط توسعه دهندگان زبان سی در این فایل تعریف شده‌اند.

حوزهٔ تعریف

- وقتی یک متغیر را تعریف می‌کنیم، در واقع متغیر باید ساخته شود و بر روی حافظه قرار بگیرد. وقتی یک متغیر را اعلام می‌کنیم در واقع به کامپایلر می‌گوییم آن متغیر قبلا تعریف شده و اکنون می‌خواهیم از آن استفاده کنیم.

- تا آنجایی که امکان دارد بهتر است از متغیرهای عمومی و خارجی استفاده نکنیم. دلیل اول این است که استفاده زیاد از متغیرهای خارجی از خوانایی برنامه می‌کاهد. دلیل دوم این است که گاهی ممکن است توابع مختلف مقدار یک متغیر عمومی را به نحوی تغییر دهند که برنامه نویس نسبت به آن آگاه نباشد و این امر باعث ایجاد برنامه‌ای شود که از نظر منطقی دچار مشکل شود. سومین دلیل این است که ممکن است چند تابع در یک اجرای همزمان (در برنامه نویسی همروند) به طور همزمان یک متغیر را تغییر دهند که باعث ایجاد اشکال در اجرای برنامه شود. چهارمین دلیل این است که توابعی که پارامتر دریافت می‌کنند به طور عمومی تعریف می‌شوند و در همهٔ برنامه‌ها قابل استفاده هستند چون به صورت یک تابع مستقل عمل می‌کنند. بنابراین ترجیح می‌دهیم برنامه قبلی را با متغیرهای محلی پیاده‌سازی کنیم و نه متغیرهای عمومی.

توابع *

- توابع می‌توانند هر نوع مقداری را بازگردانند. برای مثال فرض کنید می‌خواهیم تابعی بنویسیم که یک رشته را به عدد اعشاری تبدیل کند. در اینصورت تابع مورد نظر باید مقدار `double` بازگرداند.

توابع *

- تابع تبدیل رشته به عدد اعشاری به صورت زیر نوشته می‌شود.

```
۱  #include <ctype.h>
۲  /* atof: convert string s to double */
۳  double atof (char s[])
۴  {
۵    double val, power;
۶    int i, sign;
۷    for (i = 0; isspace (s[i]); i++)   /* skip white space */
۸      ;
۹    sign = (s[i] == '-') ? -1 : 1;
۱۰   if (s[i] == '+' || s[i] == '-')
۱۱     i++;
```

توابع *

```
13    for (val = 0.0; isdigit (s[i]); i++)
14      val = 10.0 * val + (s[i] - '0');
15    if (s[i] == '.')
16      i++;
17    for (power = 1.0; isdigit (s[i]); i++)
18      {
19        val = 10.0 * val + (s[i] - '0');
20        power *= 10;
21      }
22    return sign * val / power;
23  }
```

توابع *

- یک تابع را می‌توان در کنار تعریف متغیرها اعلام کرد. مزیت اعلام یک تابع این است که استفاده کنندهٔ آن خواهد دانست چگونه از تابع استفاده کند.

توابع *

- در برنامه زیر تابع atof اعلام شده است.

```
۱  #include <stdio.h>
۲  #define MAXLINE 100
۳  /* rudimentary calculator */
۴  int main ()
۵  {
۶    double sum, atof (char[]);
۷    char line[MAXLINE];
۸    int getline (char line[], int max);
۹    sum = 0;
۱۰   while (_getline (line, MAXLINE) > 0)
۱۱     printf ("\t%g\n", sum += atof (line));
۱۲   return 0;
۱۳ }
```

توابع *

- الزامی به اعلام توابع وجود ندارد و توابع را می‌توانیم تنها تعریف کنیم. در اینصورت در هنگام فراخوانی، کامپایلر به دنبال تعریف تابع می‌گردد.
- با فرض اینکه تابع atof برای تبدیل رشته به اعشاری تعریف شده است، می‌توانیم تابعی تعریف کنیم که یک رشته را به یک عدد صحیح تبدیل می‌کند.

```
۱ /* atoi: convert string s to integer using atof */
۲ int atoi (char s[])
۳ {
۴   double atof (char s[]);
۵   return (int) atof (s);
۶ }
```

توابع *

- توابع می‌توانند توسط متغیرهای عمومی و متغیرهای خارجی نیز با یکدیگر اطلاعات به اشتراک بگذارند. متغیرهای محلی یک تابع در هنگام فراخوانی تعریف شده و در هنگام اتمام اجرای تابع از بین می‌روند، ولی متغیرهای عمومی مقدار خود را نگه می‌دارند.

توابع *

- فرض کنید می‌خواهیم یک ماشین حساب بنویسیم که عملیات جمع و تفریق و ضرب و تقسیم انجام دهد. از آنجایی که محاسبهٔ عبارت‌های ریاضی که به صورت پسوندی [1] بیان می‌شوند برای کامپیوتر آسان‌تر است، به جای دریافت عبارت‌ها به صورت میانوندی [2] از نشانه گذاری پسوندی استفاده می‌کنیم.
- برای مثال عبارت میانوندی (1-2)*(4+5) در نشانه گذاری پسوندی به صورت 12-45+* بیان می‌شود. نشانه گذاری پسوندی غیر مبهم است پس به پرانتز گذاری نیاز ندارد.

[1] postfix
[2] infix

توابع *

- پیاده‌سازی این ماشین حساب به یک پشته نیاز دارد. هر عملوند در یک پشته ذخیره می‌شود. وقتی به یک عملگر می‌رسیم، دو عملوند را از پشته خارج می‌کنیم و عملگر مورد نظر را بر روی آنها اعمال می‌کنیم. نتیجه محاسبات مجدداً در پشته ذخیره می‌شود.

- الگوریتم این ماشین حساب بدین صورت است.

۱. تا وقتی که مقدار بعدی در رشته عملگر یا عملوند است.
۲. اگر به یک عدد رسیدی
۳. عدد را در پشته وارد کن
۴. در غیراینصورت اگر به یک عملگر رسیدی
۵. دو عملوند را از پشته خارج کن و عملگر را بر روی آنها اعمال کن و نتیجه را در پشته ذخیره کن.
۶. در غیراینصورت اگر به کاراکتر خط جدید رسیدی
۷. مقدار نهایی را از پشته خارج کن و چاپ کن.
۸. در غیراینصورت پیام خطا چاپ کن.

توابع *

- برای هر یک از عملیات پشته یک تابع تعریف می‌کنیم. این توابع نیاز دارند مکان آخر پشته را به یکدیگر منتقل کنند. آخرین مکان پشته را می‌توانیم به صورت یک متغیر عمومی تعریف کنیم.
- این برنامه را در چند فایل می‌نویسیم. در یک فایل بدنه اصلی برنامه تعریف می‌شود، در فایل دیگر توابع مربوط به پشته تعریف می‌شوند و در فایل دیگر توابع مورد نیاز برای دریافت ورودی.

توابع *

- بدنه اصلی برنامه ماشین حساب به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  #include <stdlib.h>        /* for atof() */
۳  #define MAXOP 100   /* max size of operand or operator */
۴  #define NUMBER '0'   /* signal that a number was found */
۵  int getop (char[]);
۶  void push (double);
۷  double pop (void);
۸  /* reverse Polish calculator */
۹  int main ()
۱۰ {
۱۱   int type;
۱۲   double op2;
۱۳    char s[MAXOP];
۱۴     while ((type = getop (s)) != EOF)
```

توابع *

```
۱۶      {
۱۷    switch (type)
۱۸       {
۱۹           case NUMBER:
۲۰                push (atof (s));
۲۱                break;
۲۲           case '+':
۲۳                push (pop () + pop ());
۲۴                break;
۲۵           case '*':
۲۶                push (pop () * pop ());
۲۷                break;
۲۸           case '-':
۲۹                op2 = pop ();
۳۰                push (pop () - op2);
۳۱                break;
```

```
۳۲            case '/':
۳۳                 op2 = pop ();
۳۴                 if (op2 != 0.0)
۳۵                      push (pop () / op2);
۳۶                 else
۳۷                      printf ("error: zero divisor\n");
۳۸                 break;
۳۹            case '\n':
۴۰                 printf ("\t%.8g\n", pop ());
۴۱                 break;
۴۲            default:
۴۳                 printf ("error: unknown command %s\n", s);
۴۴                 break;
۴۵            }
۴۶       }
۴۷    return 0;
۴۸ }
```

توابع *

- پشته مورد نیاز در برنامه ماشین حساب به صورت زیر نوشته می‌شود.

```
۱  #include <stdio.h>
۲  #define MAXVAL 100    /* maximum depth of val stack */
۳  int sp = 0;    /* next free stack position */
۴  double val[MAXVAL];    /* value stack */
۵  /* push: push f onto value stack */
۶  void push (double f)
۷  {
۸    if (sp < MAXVAL)
۹         val[sp++] = f;
۱۰   else
۱۱        printf ("error: stack full, can't push %g\n", f);
۱۲ }
```

توابع *

```
۱۴ /* pop: pop and return top value from stack */
۱۵ double pop (void)
۱۶ {
۱۷   if (sp > 0)
۱۸         return val[--sp];
۱۹   else
۲۰     {
۲۱         printf ("error: stack empty\n");
۲۲         return 0.0;
۲۳     }
۲۴ }
```

توابع *

- توابع مورد نیاز برای دریافت عملوندها به صورت زیر تعریف می‌شود.

```
۱ #include <stdio.h>
۲ #include <ctype.h>
۳ #define NUMBER '0'     /* signal that a number was found */
۴ int getch (void);
۵ void ungetch (int);
۶ /* getop: get next character or numeric operand */
۷ int getop (char s[])
۸   int i, c;
۹   while ((s[0] = c = getch ()) == ' ' || c == '\t')
```

```
11      ;
12    s[1] = '\0';
13    if (!isdigit (c) && c != '.')
14      return c;       /* not a number */
15    i = 0;
16    if (isdigit (c))    /* collect integer part */
17      while (isdigit (s[++i] = c = getch ()))
18        ;
19    if (c == '.')     /* collect fraction part */
20      while (isdigit (s[++i] = c = getch ()))
21        ;
22    s[i] = '\0';
23    if (c != EOF)
24      ungetch (c);
25    return NUMBER;
26  }
```

توابع *

- در این توابع نیاز به یک تابع برای دریافت یک کاراکتر داریم که آن را با نام `getch` تعریف می‌کنیم. وقتی می‌خواهیم یک عدد دریافت کنیم نیاز داریم کاراکترها به ترتیب دریافت کنیم تا زمانی که کاراکتر بعدی یک رقم نباشد. از آنجایی که طول یک عدد مشخص نیست باید یکی یکی کاراکترها را دریافت کنیم و در نهایت کاراکتری دریافت خواهیم کرد که رقم نیست. در اینصورت نیاز داریم این کاراکتر خوانده شده را بازگردانیم. تابع `ungetch` برای بازگرداندن یک کاراکتر استفاده می‌شود.

توابع *

- توابع دریافت کاراکتر به صورت زیر تعریف می‌شوند.

```
۱ #include <stdio.h>
۲ #define BUFSIZE 100
۳ char buf[BUFSIZE];  /* buffer for ungetch */
۴ int bufp = 0;  /* next free position in buf */
۵ int getch (void)  /* get a (possibly pushed-back) character */
۶ {
۷   return (bufp > 0) ? buf[--bufp] : getchar ();
۸ }
```

توابع *

```
۱۰ void ungetch (int c)  /* push character back on input */
۱۱ {
۱۲   if (bufp >= BUFSIZE)
۱۳     printf ("ungetch: too many characters\n");
۱۴   else
۱۵     buf[bufp++] = c;
۱۶ }
```

برنامه در چند فایل *

- معمولاً برنامه‌ها در زبان سی در چند فایل ذخیره می‌شوند و فایل‌ها به طور جداگانه کامپایل شده و به یکدیگر پیوند داده می‌شوند.
- حوزهٔ تعریف [1] یک نام، قسمتی از برنامه است که در آن نام تعریف شده است. برای مثال حوزهٔ تعریف یک متغیر که در ابتدای یک تابع تعریف شده است، بلوک دستورات آن تابع است. دو متغیر محلی در دو تابع مختلف دو متغیر کاملا متفاوت هستند. حوزهٔ تعریف یک متغیر عمومی در ابتدای یک فایل، همهٔ آن فایل است که شامل توابع تعریف شده در آن فایل می‌شود. همچنین یک تابع که در یک فایل تعریف و اعلام شده است در همهٔ آن فایل قابل استفاده است و بنابراین حوزهٔ تعریف آن همه آن فایل است.

[1] scope

برنامه در چند فایل *

- اگر تابع `f` بعد از تابع `g` تعریف شده باشد، تابع `f` در تابع `g` تابع استفاده نیست مگر اینکه `f` قبل از `g` اعلام شده باشد.
- اگر یک متغیر در یک فایل دیگر تعریف شده باشد، توسط کلیدواژه `extern` می‌توان آن متغیر را در فایل جاری اعلام کرد. در این صورت فایل دیگر باید توسط `include` به فایل جاری معرفی شود. یک متغیر یا یک تابع فقط یک بار می‌تواند تعریف شود. ولی می‌توان در چند مکان اعلام شود.

برنامه در چند فایل *

- معمولاً توابع در یک برنامه سی در یک فایل سرتیتر [1] با پسوند `h`. اعلام می‌شوند و در یک فایل سورس با پسوند `c`. تعریف می‌شوند.
- برای مثال در برنامهٔ ماشین حساب می‌توان همهٔ توابع را در یک فایل `calc.h` اعلام کرد و توابع مربوط به پشته را در فایل `stack.h` ، توابع دریافت عملوندها را در فایل `getop.c` ،توابع مربوط به دریافت کاراکتر را در فایل `getch.c` و تابع بدنه اصلی را در فایل `main.c` ذخیره کرد. برای اینکه هر یک از فایل‌های سورس که به توابع فایل‌های سورس دیگر نیاز دارند بتوانند کامپایل شوند، نیاز داریم در همهٔ فایل‌ها `calc.h` را اضافه کنیم. همچنین در فایل `main.c` نیاز به اعلام توابع داریم که برای اعلام آنها `calc.h` را به فایل اصلی اضافه می‌کنیم. وقتی یک متغیر در یک فایل تعریف شود، آن متغیر می‌تواند توسط کلیدواژهٔ `extern` به عنوان متغیر خارجی در یک فایل دیگر استفاده شود.

[1] header file

متغیرهای ایستا *

- اگر بخواهیم یک متغیر به طور خصوصی برای یک فایل تعریف شود و در فایل‌های دیگر قابل استفاده نباشد، آن متغیر باید به صورت ایستا با کلیدواژه `static` تعریف شود.
- یک متغیر عمومی ایستا در یک فایل در فایل‌های دیگر قابل استفاده نیست.
- همینطور می‌توان یک تابع را به طور ایستا تعریف کرد. تابع ایستا در یک فایل در فایل‌های دیگر قابل استفاده نیست.
- اگر یک متغیر به صورت ایستا در یک تابع تعریف شود، پس از اتمام فراخوانی تابع مقدار آن از بین نمی‌رود. در واقع متغیرهای ایستا حتی اگر محلی باشند، پس از تعریف تا اتمام برنامه در حافظه باقی می‌مانند.

متغیرهای رجیستر *

- متغیرهای پر استفاده معمولاً با استفاده از کلیدواژهٔ `register` تعریف می‌شوند. متغیرهای رجیستر در واقع برروی رجیستر پردازنده قرار می‌گیرند و نه برروی حافظه. بدین ترتیب سرعت دسترسی به آنها بسیار بیشتر است. از آنجایی که تعداد رجیسترهای پردازنده محدود است، در استفاده از متغیرهای رجیستر نیز محدودیت وجود دارد.

بلوک‌ها

- در زبان سی، یک متغیر می‌تواند در یک بلوک تعریف شود. این بلوک می‌تواند هرگونه بلوکی باشد مثلاً در بلوک متعلق به یک دستور `if` یا `for` می‌توان متغیر تعریف کرد. در این صورت متغیر فقط در آن بلوک تعریف شده و خارج از بلوک قابل دسترسی نیست.
- در مثال زیر متغیر `i` در بلوک `if` تعریف شده است و خارج از بلوک `if` قابل استفاده نیست.

```
۱ if (n > 0) {
۲   int i;  /* declare a new i */
۳   for (i = 0; i < n; i++)
۴     ...
۵ }
```

بلوک‌ها

- یک متغیر می‌تواند به صورت عمومی تعریف شود و سپس در یک بلوک مجدداً با همان نام تعریف شود. در این صورت، در بلوک متغیر محلی استفاده می‌شود و در خارج از بلوک متغیر عمومی.
- در مثال زیر متغیر x در داخل تابع از نوع double است و مقدار متفاوتی از متغیر عمومی x دارد.

```
۱ int x;
۲ int y;
۳ void f(double x) {
۴   double y;
۵ }
```

بلوک‌ها

- متغیر پس از تعریف باید مقداردهی اولیه شوند، در غیراینصورت مقدار آنها معتبر نخواهد بود.
- مقداردهی اولیه با عملگر تساوی انجام می‌شود. برای مثال `int low = 0` متغیر `low` را مقداردهی اولیه می‌کند.
- آرایه‌ها نیز می‌توانند با تعیین مقادیر عناصر آرایه مقداردهی اولیه شوند. برای مثال
`int days[] = { 31,24 }` یک آرایه با دو عنصر با مقادیر 31 و 24 می‌سازد.
- مقداردهی اولیه رشته‌ها می‌تواند به دو صورت انجام شود. یک رشته آرایه‌ای از کاراکترهاست بنابراین می‌توانیم بنویسیم
`char str[] = {'h' , 'i'}` و همچنین یک رشته می‌تواند به صورت
`char str = "hi"` مقداردهی اولیه شود.

توابع بازگشتی

- توابع در زبان سی می‌توانند به صورت بازگشتی نیز فراخوانی شوند. یک تابع می‌تواند خود را فراخوانی کند که به این فراخوانی بازگشتی گفته می‌شود. توابعی که فراخوانی بازگشتی دارند را توابع بازگشتی [1] می‌نامیم.
  توابع می‌توانند خود را به صورت مستقیم یا غیر مستقیم فراخوانی کنند.

[1] recursive function

توابع بازگشتی

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که فاکتوریل عدد `n` را محاسبه کند.
- فاکتوریل عدد `n` برابر است با `n` ضرب در فاکتوریل عدد `n-1`

توابع بازگشتی

- تابع فاکتوریل بازگشتی را می‌توانیم به صورت زیر بنویسیم.

```c
#include <stdio.h>
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n-1);
}
int main() {
  int x = 5;
  int fact = factorial(x);
  printf("factorial(%d) = %d\n", x, fact);
  return 0;
}
```

توابع بازگشتی *

- فرض کنید می‌خواهیم برنامه‌ای بنویسیم که یک عدد صحیح را به یک رشته تبدیل کند. روش اول این است که ارقام با ارزش پایین‌تر به ترتیب از عدد جدا شده و رشته از راست به چپ ساخته شود. روش دوم این است که یک فراخوانی بازگشتی استفاده کنیم.
- وقتی می‌خواهیم عدد `Ax` را به رشته تبدیل کنیم به طوری‌که `A` یک عدد و `x` یک رقم است در واقع باید ابتدا `A` را به رشته تبدیل کرده و سپس `x` را به صورت کاراکتر چاپ کنیم.

توابع بازگشتی *

- این برنامه را می‌توانیم به صورت زیر بنویسیم.

```
۱  #include <stdio.h>
۲  /* printd: print n in decimal */
۳  void
۴  printd (int n)
۵  {
۶    if (n < 0)
۷      {
۸        putchar ('-');
۹        n = -n;
۱۰     }
۱۱   if (n / 10)
۱۲     printd (n / 10);
۱۳   putchar (n % 10 + '0');
۱۴ }
```

توابع بازگشتی *

- یک مثال دیگر برای یک تابع بازگشتی الگوریتم مرتب‌سازی سریع است. با استفاده از این الگوریتم مرتب‌سازی، یک عنصر از آرایه انتخاب می‌شود و عناصر کوچک‌تر از آن به سمت چپ و عناصر بزرگ‌تر از آن به سمت راست عنصر انتخاب شده انتقال داده می‌شوند. این عملیات به صورت بازگشتی برای زیر آرایه‌ها تکرار می‌شود تا کل آرایه مرتب شود.

توابع بازگشتی *

- الگوریتم مرتب‌سازی سریع به صورت زیر است.

```
۱  /* qsort: sort v[left]...v[right] into increasing order */
۲  void
۳  qsort (int v[], int left, int right)
۴  {
۵    int i, last;
۶    void swap (int 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);  /* move partition elem */
۱۰   last = left;  /* to v[0] */
۱۱   for (i = left + 1; i <= right; i++)  /* partition */
۱۲     if (v[i] < v[left])
۱۳       swap (v, ++last, i);
```

توابع بازگشتی *

```
14    swap (v, left, last);  /* restore partition elem */
15    qsort (v, left, last - 1);
16    qsort (v, last + 1, right);
17  }
```

توابع بازگشتی *

- عملیات جابجایی دو عنصر آرایه به صورت زیر انجام می‌شود.

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

توابع بازگشتی *

- در یک فراخوانی بازگشتی، درواقع فراخوانی‌های پی‌درپی برروی پشته‌ای در حافظه ذخیره می‌شوند تا مقادیر آنها بعدا مورد استفاده قرار بگیرد.
- یک تابع بازگشتی از تابع غیربازگشتی معادل آن سریع‌تر نیست، اما کوتاهترین نوشته می‌شود و نوشتن و خواندن برنامه را آسان‌تر می‌کند.

پیش پردازش *

- قبل از مرحله کامپایل در زبان سی، یک مرحله پیش پردازش [1] وجود دارد. در این مرحله فایل‌هایی که نام آنها توسط کلمه include به فایل اضافه شده‌اند، محتوایشان نیز به فایل اضافه می‌شود. همچنین نمادهای ثابتی که توسط define تعریف شده‌اند، مقادیرشان در فایل جایگزین نمادها می‌شوند.
- فایل‌های به صورت #include <filename> مشخص شده‌اند، در آدرس‌های تعیین شده توسط سیستم عامل جستجو می‌شوند و فایل‌های که به صورت #include "filename" استفاده شد، در مکانی که برنامه ذخیره شده است جستجو می‌شوند.
- توسط کلمه define در زبان سی می‌توان یک نماد تعریف کرد. این نمادها ماکرو نامیده می‌شوند. این نمادها می‌توانند جایگزین هر قسمتی از برنامه شوند. برای مثال برای تعریف یک حلقه بینهایت می‌توانیم بنویسیم.

```
1 #define forever for (;;)
```

[1] prepoccessing

پیش پردازش *

- ماکروها را می‌توان با استفاده از پارامتر نیز تعریف کرد. در مثال زیر، نماد max با دو پارامتر با معادل آن در کد جایگزین می‌شود.

```
۱ #define max(A,B) ((A) > (B) ? (A) : (B))
```

یک برنامه‌ای که از این ماکرو استفاده می‌کند max(p+q , r+s) با عبارت
((p + q) > (r + s) ? (p + q) : (r + s)) جایگزین می‌شود.

- در ماکروها باید به پرانتزگزاری توجه کرد. برای مثال اگر ماکرویی به صورت

```
۱ #define square(x) x*x       /* WRONG */
```

تعریف کنیم، آنگاه square(z+1) معادل خواهد بود، z+1 * z+1 که معادل است با 2z+1

پیش پردازش *

- اگر یک نماد توسط یک ماکرو تعریف شود، می‌توان در قطعه‌ای از برنامه تعریف را توسط کلیدواژه undef لغو کرد.
- برای مثال :

```
۱ #undef getchar
۲ int getchar() {...}
```

- اگر در عبارت جایگزین شده در یک ماکرو قبل از یک متغیر از علامت # استفاده کنیم، آن متغیر در عبارت جایگزین شده در میان دو علامت نقل قول قرار می‌گیرد.
- برای مثال اگر داشته باشیم :

```
۱ #define dprint(expr) printf(#expr " = %g\n" , expr)
```

آنگاه با اجرای dprint(x/y) در واقع دستور printf("x/y" "=% g \n" , x/y) اجرا می‌شود.

پیش پردازش *

- در ماکروها می‌توانیم از علامت ## استفاده کنیم برای اتصال دو پارامتر به یکدیگر استفاده می‌شود.
- برای مثال اگر داشته باشیم :

```
1 #define paste(front, back) front ## back
```

آنگاه با اجرای `paste(name,1)` عبارت `name1` به دست می‌آید.

پیش پردازش *

- دستورات ماکرویی برای کنترل کردن ماکروها و اجرای شرطی آنها وجود دارد. برای مثال دستور ماکروی `#if` یک ورودی دریافت می‌کند و در صورتی که ورودی غیر صفر باشد دستورات ماکروی بعد از آن تا رسیدن به ماکروی `#endif` اجرا می‌شوند. همچنین دستورات ماکروی `#elif` و `#else` نیز برای اجرای دستورات `else if` و `else` در ماکرو وجود دارند.

پیش پردازش *

- فرض کنید می‌خواهیم در یک برنامه مطمئن شویم که یک فایل سر تیتر تنها یک بار به یک فایل افزوده می‌شود.
- برای این کار یک متغیر در هنگام اعلام توابع در فایل سر تیتر تعریف می‌کنیم و سپس اطمینان حاصل می‌کنیم که این متغیر تنها یک بار تعریف شده است.
- برای مثال فایل `hdr.h` به صورت زیر تعریف شده است.

```
۱ #if !defined(HDR)
۲ #define HDR
۳ /* contents of hdr.h go here */
۴ #endif
```

پیش پردازش *

- در برنامه زیر با توجه به نوع سیستم عامل عملیات متفاوت انجام می‌دهیم.

```
۱  #if SYSTEM == SYSV
۲  #define HDR "sysv.h"
۳  #elif SYSTEM == BSD
۴  #define HDR "bsd.h"
۵  #elif SYSTEM == MSDOS
۶  #define HDR "msdos.h"
۷  #else
۸  #define HDR "default.h"
۹  #endif
۱۰ #include HDR
```

پیش پردازش *

- ماکروی `ifndef` با یک پارامتر ورودی، بدین معنی است که اگر پارامتر ورودی تعریف نشده بود، آنگاه عملیات بعدی را تا رسیدن به `endif` انجام بده.

```
1 #ifndef HDR
2 #define HDR
3 /* contents of hdr.h go here */
4 #endif
```