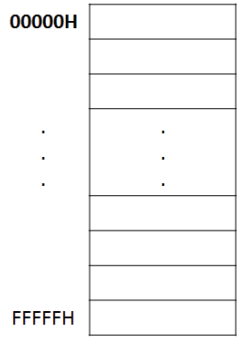
۸ اشاره‌گرها

حافظهٔ اصلی کامپیوتر یا RAM از تعدادی بایت که هر بایت از ۸ بیت تشکیل شده است، شکل گرفته است. به عبارت دیگر، حافظهٔ اصلی کامپیوتر را می‌توان به صورت آرایه‌ای از بایت‌ها در نظر گرفت. لذا اگر بایت‌ها را به ترتیب شماره‌گذاری کنیم، به هر بایت یک شماره اختصاص داده می‌شود که محل قرارگرفتن بایت در حافظه را نشان می‌دهد که به این شماره اصطلاحاً آدرس بایت می‌گوییم. همانطور که قبلاً بیان شد، هر متغیر بسته به فضای مورد نیاز، فضایی از حافظه (یک یا چند بایت متوالی) را اشغال می‌کند و هر متغیر نامی نیز دارد. شماره یا آدرس اولین بایتی که متغیر در آن محل از حافظه ذخیره شده است را اشاره‌گر به آن متغیر یا آدرس آن متغیر می‌نامیم.

اشاره‌گر در C++ کاربردهای فراوانی دارد و درک درست مفهوم اشاره‌گر جهت یادگیری قابلیت‌های پیشرفتهٔ زبان C++ بسیار حیاتی است.



۱.۸ معرفی اشاره‌گرها

اگرچه آدرس متغیرها صرفنظر از نوع آنها، یک عدد است ولی نمی‌توان آن‌ها را در متغیرهایی از انواع معمول عددی نظیر double و int ذخیره کرد. آدرس یک متغیر را باید در متغیری ذخیره کنیم که از نوع اشاره‌گر باشد که به این متغیرها، متغیرهای اشاره‌گری می‌گوییم.

فرض کنید i یک متغیر از نوع صحیح با مقدار ۱۰ و شماره اولین بایت تخصیص داده شده برای نگه داری آن ۱۰۲۴ باشد. در این صورت آدرس i، ۱۰۲۴ است و مقدار آن ۱۰ است که این دو عدد از هم مستقل هستند. برای نگهداری آدرس i یک متغیر از نوع اشاره گر مورد نیاز است. متغیرهای اشاره‌گری، متغیرهایی هستند که آدرس متغیرها در آنها قرار می‌گیرد.

نحوه تعریف اشاره‌گر دقیقاً مانند متغیر معمولی است با این تفاوت که قبل از نوشتن نام متغیر علامت * قرار می‌گیرد. این علامت به کامپایلر می‌گوید که متغیر یک اشاره‌گر است شکل کلی تعریف اشاره‌گر به صورت زیر است:

```
نوع‌متغیر   * نام اشاره‌گر;
```

منظور از نوع متغیر، نوع متغیری است که می‌خواهیم آدرس آن را در متغیر اشاره‌گری ذخیره کنیم. به عنوان مثال اشاره‌گر p جهت نگهداری آدرس یک متغیر از نوع عدد صحیح (int) به صورت زیر تعریف شده است:

```
int  *p;
```

سوالی که ممکن است مطرح شود این است که، آدرس یک متغیر، صرفنظر از نوع متغیر، از یک نوع است و به نوع متغیر بستگی ندارد. پس چرا فقط یک نوع متغیری برای نگهداری آدرس یا اشاره‌گر تعریف نمی‌شود تا از آن برای نگهداری آدرس متغیرها استفاده شود. دلیل این کار، در بخش مربوط به انجام محاسبات روی اشاره‌گرها مشخص خواهد شد.

۱.۱.۸ استفاده از اشاره‌گرها

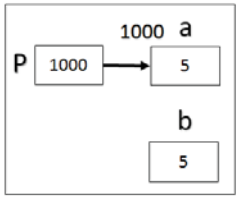
اما چگونه می‌توان به آدرس یک متغیر دست یافت و در صورتی که آدرس یک متغیر را داشته باشیم، چگونه می‌توانیم از آن استفاده کنیم؟

دو عملگر زیر برای کار با اشاره‌گرها مورد استفاده قرار می‌گیرند:

۱. عملگر &: با قراردادن این عملگر قبل از یک متغیر، ادرس آن متغیر برگردانده می شود.
۲. عملگر *: با قراردادن این عملگر قبل از یک اشاره‌گر حافظه مربوط به متغیری که آدرس آن در اشاره‌گر ذخیره شده در دسترس قرار می گیرد. بنابراین می توان از آن در خواندن و نوشتن در آن محل استفاده کرد.

دستورات زیر را در نظر بگیرید.

```
1  int  *p,a,b;
2  a=5;
3  p=&m;
4  b=*p;
```



در خط اول، دو متغیر a و b از نوع صحیح و یک اشاره‌گر p از نوع صحیح تعریف شده است. فرض کنید آدرس a و b در حافظه به ترتیب ۱۰۰۰ و ۱۰۴۸ باشد. در خط ۲، مقدار a را برابر ۵ قرار می‌دهیم و در خط ۳، آدرس متغیر یعنی ۱۰۰۰ را در p قرار می‌دهیم و در خط آخر محتویات جایی که p به آن اشاره می‌کند، یعنی a، را در b قرار می‌دهیم.

دقت کنید که آدرس یک متغیر (به عنوان مثال $\&m$) را باید در یک متغیر اشاره‌گری قرار داد و محتویات جایی که اشاره‌گر به آن اشاره می‌کند (به عنوان مثال $*p$) را باید در یک متغیر معمولی قرار داد.

وقتی اشاره‌گری از نوع * int تعریف می‌کنیم باید به متغیری از نوع int اشاره نماید و اشاره‌گر از نوع * double نیز باید به متغیری از همان نوع اشاره کند.

اگر متغیری که آدرس آن در یک اشاره‌گر قرار می‌گیرد با نوع اشاره‌گر یکسان نباشد، با توجه به عواقب پیش‌بینی نشده‌ای که این انتقال ممکن است ایجاد کند، کامپایلر C++ خطا اعلام می‌کند ولی می‌توان با استفاده از تغییر نوع، این خطا را برطرف و انتقال را انجام داد، اما برنامه‌نویس باید دقت کند که این امر ممکن است بر صحت اجرای برنامه اثر گذاشته و نتایج درستی حاصل نشود. دقت کنید که در انتساب متغیرهای معمولی، در صورتی که یک متغیر یا مقدار از نوعی به متغیری از نوع دیگر انتساب داده شود، با خطای کامپایلر مواجه نمی‌شود ولی در خصوص متغیرهای اشاره‌گری با توجه به عواقب بسیار عجیبی که این انتقال ممکن است ایجاد کند، کامپایلر با خطا در زمان کامپایل برنامه متوقف می‌شود.

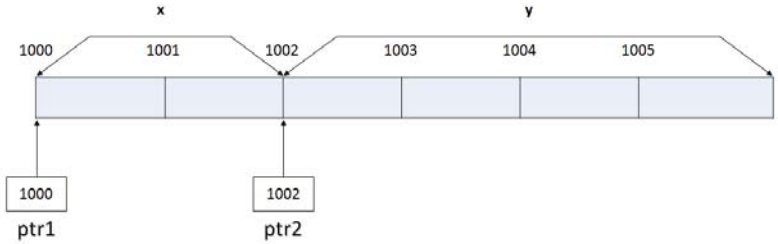
به عنوان مثال، در برنامه زیر، فرض کنید متغیر x از نوع int در محل ۱۰۰۰ از حافظه ذخیره شده باشد و ۲ بایت فضا اشغال کند و متغیر y از نوع float در محل ۱۰۰۲ ذخیره شده باشد و به اندازه ۴ بایت فضا از حافظه اشغال کند. شکل ۱.۸ ترتیب ذخیره‌سازی متغیرها را نشان می‌دهد.

```
1  #include  <iostream>
2  using  namespace std;
3  int  main(void)
```



```
4  {
5       int  *ptr1 ,*ptr2 ,m1,m2;
6       int  x=10;
7       float  y=13.25;
8       ptr1 =&x;
9       ptr2=(int  *)&y;
10      m1=*ptr1;
11      m2=*ptr2;
12      cout<<"m1="<<m1<<", m2="<<m2;
13 }
```

خروجی برنامه m1=10, m2=1096024067 است.



شکل ۱.۸: محل ذخیره‌سازی متغیرها.

در این شکل بایت‌های ۱۰۰۰ و ۱۰۰۱ در اختیار x و بایت‌های ۱۰۰۲ تا ۱۰۰۵ در اختیار y است. دستور ptr1=& x موجب می‌شود تا آدرس x که ۱۰۰۰ است در ptr1 قرارگیرد و دستور ptr2=& y نیز آدرس y که ۱۰۰۲ است را ptr2 قرار می‌دهد دستور m1=*ptr1 محتوای دو بایت از حافظه‌ای که ptr1 به آن اشاره می‌کند را در متغیر m1 قرار می‌دهد و دستور m2=*ptr2 با توجه به این که ptr2 یک اشاره‌گر به int است محتویات دو بایت ۱۰۰۲ ، ۱۰۰۳ را در $m2$ قرار می‌دهد بنابراین قسمتی از اطلاعات که در بایت‌های ۱۰۰۴ و ۱۰۰۵ است از بین می‌رود.

۲.۱.۸ عملیات محاسباتی روی اشاره‌گرها

اشاره‌گرها را می‌توان به هم منتسب کرد و بر روی آنها عملیات محاسباتی انجام داد. در مثال‌های زیر عمل‌های قابل انجام روی اشاره‌گرها را توضیح می‌دهیم.

۳.۱.۸ انتساب اشاره‌گرها

اگر یک متغیر اشاره‌گری را به یک متغیر اشاره‌گری انتساب دهیم، هر دو اشاره‌گر به یک محل از حافظه اشاره می‌کنند. به مثال زیر توجه کنید.

```
int *ptr1,*ptr2,x,y;
x=10;
y=12;
ptr1=&x;
ptr2=&y;
```

شکل زیر، به صورت شماتیک رابطهٔ بین متغیرها را نشان می‌دهد.

اگر پس از اجرای این کدها دستور *ptr1=*ptr2 اجرا کنیم محتویات اشارەگر ptr2 به ptr1 منتقل می‌شود. یعنی عملاً مقدار متغیر y در متغیر x کپی می‌شود و شکل زیر حاصل می‌شود.

اگر پس از اجرای این کدها دستور ptr1=ptr2 را اجرا کنیم آدرس موجود در ptr2 به ptr1 منتقل می‌شود یعنی عملاً اشارەگر ptr1 نیز به همان متغیری اشاره می‌کند که ptr2 به آن اشاره می‌کند.

پس اگر آدرس متغیرها را به هم منتسب کنیم اشارەگر سمت چپ علامت انتساب به جایی اشاره می‌کند که اشارەگر سمت راست علامت انتساب اشاره می‌کرده است و در نتیجه هر دو اشارەگر به یک محل از حافظه اشاره می‌کنند.

۴.۱.۸ عملیات محاسباتی و مقایسه‌ای روی اشارەگرها

مشابه سایر متغیرها، می‌توان روی اشارەگرها نیز عملیات محاسباتی و مقایسه‌ای انجام داد ولی عملکرد این عملیات کمی با آنچه در ذهن است متفاوت است. برای بیان عملیات محاسباتی و مقایسه‌ای مثال زیر را در نظر بگیرید.

```
#include <iostream>
using namespace std;
int main(void)
```



```
{
    int *ptr1,*ptr2,m;
    int x=10,y=12;
    ptr1=&x;
    ptr2=&y;
    (*ptr1)++;
    m=*ptr1+*ptr2;
    ptr1++;
}
```

در خط ۹ محتوای جایی که اشارەگر ptr1 به آنجا اشاره می‌کند یک واحد افزایش می‌یابد. به عبارت دقیق‌تر، مقدار x یک واحد افزایش می‌یابد. در خط ۱۰ محتوای محل‌هایی از حافظه که ptr1 و ptr2 به آنها اشاره می‌کنند را با هم جمع می‌شود و حاصل در متغیر m قرار می‌گیرد. در خط ۱۱ آدرس موجود در اشارەگر ptr1 را یک واحد افزایش می‌یابد.

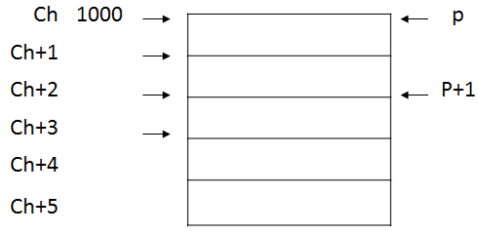
چند نکته مهم در این دستور وجود دارد. اولاً ptr1 دیگر به متغیر x اشاره نمی‌کند. ثانیاً با فرض این که int دو بایت فضا اشغال می‌کند، آدرس موجود در ptr1 به اندازه دو واحد افزایش می‌یابد و ثالثاً چون ما بلافاصله بعد از تعریف متغیر x متغیر y را تعریف کردیم این بدان معنی نیست که حتماً در حافظه نیز این دو پشت سر هم هستند و با افزایش آدرس x به متغیر y می‌رسیم.

به طور دقیق‌تر، وقتی که عملگر یک واحد افزایش و یا یک واحد کاهش را روی اشارەگر اعمال می‌کنیم، آدرس به اندازه فضایی که آن نوع داده اشغال می‌کند افزایش و یا کاهش می‌یابد. پس یک واحد افزایش در یک متغیر اشارەگری، بسته به نوع متغیر، به اضافه شدن مقادیر متفاوتی به متغیر منجر می‌شود. به ازای هر یک واحد افزایش یک متغیر اشارەگری، به اندازهٔ تعداد بایت‌هایی که آن نوع متغیر که متغیر اشارەگری از آن نوع است، به اشارەگر اضافه می‌کند. به عبارت دیگر، متغیر اشارەگری با یک واحد افزایش، به اولین فضای حافظهٔ بعد از فضای حافظه‌ای که متغیر اشارەگری به آن اشاره می‌کند، اشاره خواهد کرد. توجه کنید که حاصل عملیات محاسباتی روی اشارەگرها بسته به نوع آن، متفاوت است، اما از این قانون کلی پیروی می‌کند که هر واحد افزایش، به اولین بایت بعد از فضای استفاده شده توسط متغیر فعلی اشاره می‌کند.

شکل زیر را در نظر بگیرید.

خروجی قطعه کد زیر را بر روی سیستم خود مشاهده کنید:

```
#include <iostream>
using namespace std;
int main(void)
{
    int *ptr1,x=10;
    ptr1=&x;
    cout<<"*ptr1="<<*ptr1<<", ptr1="<<ptr1<<endl;
    (*ptr1)++;
```




```
 9      cout<<"*ptr1="<<*ptr1<<endl;
10      ptr1++;
11      cout<<"*ptr1="<<ptr1;
12  }
```

۵.۱.۸ اشارهگر به اشارهگر

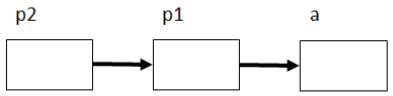
هر متغیر اشارهگری، خودش در یک محلی از حافظه نگهداری میشود و مشابه سایر متغیرها دارای آدرس است. به اشارهگر به یک متغیر اشارهگری، اشارهگر به اشارهگر گویند. ما میتوانیم متغیرهای اشارهگری تعریف کنیم که آدرس محل ذخیرهسازی یک متغیر اشارهگری را نگهداری کنند. برای تعریف این اشارهگرها از دستور کلی زیر استفاده میکنیم.

نوع متغیر ** نام اشارهگر ;

دقت کنید که با دو ستاره کنار هم این نوع اشارهگرها تعریف میشوند.

آدرس این نوع اشارهگرها یک آدرس غیرمستقیم است یعنی آدرسِ یک متغیر محتوی آدرس یک متغیر را نگهداری میکنند. این ویژگی کمی پیچیده به نظر میرسد شکل زیر نمونهای از این نوع اشارهگرها است.

```
1  int  a;
2  int  *p1  = &a;
3  int  **p2 = &p1;
```



۲.۸ اشارهگر و توابع

اشارهگرها نقش مهمی را در توابع ایفا میکنند. آنها امکان ارسال با ارجاع پارامترها را فراهم میکنند. همچنین میتوان برای توابع اشارهگر تعریف کرد و با استفاده از آنها به تابع دسترسی پیدا نمود. در ادامه، به طور جداگانه به بررسی هر قسمت میپردازیم.

۱.۲.۸ ارسال با ارجاع پارامترها به توابع

در فصل توابع با ارسال پارامترها به توابع آشنا شدیم و دیدیم که دو شیوه برای ارسال پارامترها وجود دارد.

ل با مقدار: در این روش مقدار متغیر تابع اصلی درون پارامترهای تابع فرعی کپی میشود و اگر در تابع فرعی مقدار متغیر تغییر کند هیچ تغییری در متغیر تابع اصلی ایجاد نمیشود.

با ارجاع : در ارسال با ارجاع به جای این که مقدار متغیر تابع اصلی درون پارامتر تابع فرعی کپی شود آدرس متغیر در تابع اصلی به تابع فرعی انتقال مییابد و این کار باعث میشود تابع با در اختیار داشتن آدرس متغیری که آدرس آن به تابع ارسال شده است بتواند محتوای متغیر را تغییر دهد.

برای تعریف پارامتر با ارجاع در امضا تابع به شکل زیر عمل میکنیم:

نوع تابع نام تابع (نوع متغیر* نام متغیر)

دقت کنید که پس از نوشتن نوع متغیر علامت * قرر گرفته است و علامت * میگوید که ارسال پارامترها با ارجاع خواهد بود و آدرس متغیر به تابع ارسال خواهد شد.

در برنامههای زیر برای محاسبه مساحت دایره از دو تابع استفاده میکنیم تا تفاوت بین فراخوانی با مقدار و فراوانی با ارجاع مشخص گردد. در برنامهٔ زیر، تابع Area1 به صورت فراخوانی با مقدار صورت میگیرد که مقدار مساحت یک دایره را محاسبه کرده و برمیگرداند.

```
 1  #include <iostream>
 2  using namespace std;
 3  const double pi=3.14;
 4  float Area1(float r)
 5  {
 6       float s;
 7      s=r*r*pi;
 8       return (s);
 9  }
10  int main(void)
11  {
```

```
12      float s,r;
13      cout<< "Enter radius :";
14      cin >>r;
15      s=Area1(r);
16      cout<<s;
17  }
```

در تابع زیر، مقدار مساحت از طریق یکی از پارامترهای تابع برگشت داده می‌شود. به عبارت دیگر، آدرس متغیر در پارامتر دوم فراخوانی تابع Area2 قرار می‌گیرد و از طریق آدرس، مقدار برگشت داده می‌شود.

```
1  #include <iostream>
2  using namespace std;
3  const double pi=3.14;
4  void Area2(int r, float *s)
5  {
6       *s=r*r*pi;
7  }
8  int main(void)
9  {
10      int r;
11      float s;
12      cout<< "Enter radius :";
13      cin >>r;
14      Area2(r,&s);
15      cout <<s;
16 }
```

یکی از ویژگی‌های خوب فراخوانی با ارجاع این است که می‌توانیم با کمک آن‌ها توابعی بنویسیم که چند مقدار را برمی‌گردانند. همانطورکه در بحث توابع بیان شد، هر تابع حداکثر یک مقدار بازگشتی می‌تواند داشته باشد.

برای این کار کافی است، به جای ارسال متغیرهایی به عنوان پارامتر به تابع، آدرس آنها را به تابع منتقل کنیم و سپس با استفاده از آدرس آنها، مقدار متغیرها را با مقادیری که در تابع محاسبه می‌شود، جایگزین کنیم.

برنامهٔ زیر محیط و مساحت مستطیلی با طول و عرض داده شده را در یک تابع فرعی محاسبه کرده و حاصل را به تابع اصلی ارسال می‌کند.

```
1  #include <iostream>
2  using namespace std;
3  const double pi=3.14;
4  void Area(double l,double w,double *p,double *s)
```



```
5  {
6       *s=l*w;
7       *p=2*(l+w);
8  }
9  int main(void)
10 {
11      double l,w,p,s;
12      cout<< "Enter lenght and width for  rectangle :";
13      cin >>l>>w;
14      Area(l,w,&p,&s);
15      cout <<p<<","<<s;
16 }
```

۲.۲.۸ اشاره‌گر به تابع

همان‌طور که یک متغیر دارای یک آدرس در حافظه است، توابع نیز دارای آدرس مشخصی در حافظه هستند. در حقیقت، روند اجرای یک برنامه به این صورت است که آن برنامه به قسمتی از حافظهٔ اصلی منتقل شده و سپس دستورات آن به ترتیب اجرا می‌شود. لذا اشاره‌گر به تابع را می‌توان آدرس محلی از حافظه دانست که تابع مربوطه در آن نقطه از حافظه ذخیره شده است. به متغیری که آدرس یک تابع را حفظ می‌کند، اشاره‌گر تابع می‌گویند. سیستم‌عامل ویندوز یک محیط رویدادگرا است، بدین معنی که ویندوز منتظر رخدادی از سوی کاربر می‌ماند و پس از بروز رخداد (کلیک کردن، فشردن کلید وغیره) پاسخ مناسب را بروز می‌دهد.

محیط‌های رویدادگرا که تغییر بنیادی در دنیای سیستم‌عامل‌ها و کامپیوتر ایجاد کردند، با استفاده از اشاره‌گر توابع پیاده‌سازی می‌شوند. اشاره‌گر یک تابع می‌گویند که برای پاسخ به یک رخداد، مانند کلیک کردن، ممکن است شرایطی پیش آید که براساس آن باید یک تابع خاص اجرا گردد. یعنی در این جا، تابع یک پارامتر است و براساس شرایط ممکن تغییر کند. قابلیت ارسال توابع به عنوان پارامتر به تابع دیگر، با استفاده از اشاره‌گر توابع صورت می‌گیرد. به عنوان مثالی ساده، کلیک کردن به روی کلیدهای کمینه کردن، بیشینه کردن و بستن را در پنجره‌های ویندوز در نظر بگیرید. در هر سه حالت، عمل کلیک کردن رخ می‌دهد ولی با توجه به این که کدام نقطه کلیک شده است یک تابع مناسب مانند Minimize ()، Maximize () و یا Close () اجرا می‌شود. بنابراین به تابع Click ()، یکی از سه تابع بالا به عنوان پارامتر ارسال می‌گردد و در داخل آن اجرا می‌شود. در حقیقت، مشابه آرایه‌ها، نام تابع، حاوی اشاره‌گر به آن تابع است. متغیر اشاره‌گری به تابع این گونه تعریف می‌شود:

نوع داده بازگشتی تابع (نام یک تابع*) (پارامترهای تابع);

به عنوان مثال اشاره‌گر زیر را در نظر بگیرید:

```
long int (*p)( int , float );
```

این تعریف مشخص میکند که p یک اشارهگر به تابعی است که دو پارامتر به ترتیب از نوع int و float دارند و یک متغیر صحیح از نوع long int بر میگرداند. آدرس هر تابعی که چنین ساختاری داشته باشد میتواند در اشارهگر p قرار بگیرد.

به قرار دادن پرانتز در دو طرف تعریف اشارهگر p هم توجه داشته باشید. نبود این پرانتزها تعبیر دیگری را برای کامپایلر تداعی میکند که در نهایت به خطا منجر میشود.

پس از تعریف چنین اشارهگری، با دستور انتسابهای زیر میتوانید آدرس تابعی را در آن قرار دهید:

```
p= نام تابع;
```

پس از این مقداردهی میتوانید از اشارهگر برای فراخوانی تابع استفاده کنید.

برای درک بهتر این مطلب مثال زیر را در نظر بگیرید. در این مثال، تابع sum مجموع اعداد از یک تا n را محاسبه کرده و برمیگرداند.

```
1  #include <iostream>
2  using namespace std;
3  int sum(int n)
4  {
5       int s=0,i;
6       for(i=1;i<=n;i++)
7            s+=i;
8       return s;
9  }
10 int main(void)
11 {
12      int n,s,(*p)(int);
13      cout<< "Enter number n:";
14      cin>>n;
15      p=sum;
16      s=p(n);
17      cout<<"Sum="<<s;
18 }
```

در مثال فوق به نحوه تعریف اشارهگر به تابع در خط ۱۲ دقت کنید.

همانطور که اشارهگر به متغیر را به صورت پارامتری به تابع ارسال کردیم میتوانیم اشارهگر به تابع را نیز به عنوان پارامتر به تابعی دیگر ارسال کنیم فقط باید توجه داشت که در تعریف تابع باید به شکل زیر عمل کرد.

فرض کنید در مثال فوق بخواهیم اشارهگر p که به تابع sum اشاره میکند را به همراه متغیر n به تابع func ارسال کنیم و در آن تابع، تابع sum فراخوانی شود. اعلان تابع func را به صورت زیر انجام میدهیم.


```
void func(int (*p)(int), int);
```

و در بدنهٔ تابع func فقط دستور p(n) را فراخوانی میکنیم این کار دقیقاً معادل فراخوانی تابع sum در تابع func است.

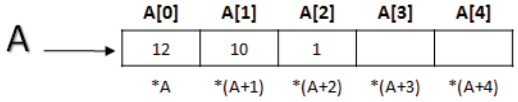
۳.۸ رابطهٔ بین آرایه و اشارهگر

میدانیم که عناصر یک آرایه در یک فضای به هم پیوسته در حافظه قرار میگیرند و لذا با دانستن آدرس محل شروع ذخیرهسازی آرایه در حافظه، میتوان به عناصر آن دسترسی داشت. در حقیقت، نام آرایه یک اشارهگر است که آدرس اولین بایت محل ذخیرهسازی آرایه را نگهداری میکند.

شکل زیر را در نظر بگیرید. آرایهای به طول ۵ در نظر گرفتهایم و مقادیری به آن نسبت دادهایم با توجه به این که نام آرایه یک اشارهگر است دستورات زیر معادل هستند.

```
int A[5];
```

$$A[0] = 12 \quad \leftrightarrow \quad *A = 12$$
$$A[1] = 10 \quad \leftrightarrow \quad *(A+1) = 10$$
$$A[2] = 1 \quad \leftrightarrow \quad *(A+2) = 1$$
$$\dots$$



در برنامه زیر به بررسی ارتباط اشارهگرها و آرایهها میپردازیم.

برنامهای که ۱۰ عدد از کاربر دریافت و سپس تمام اعداد زوج آن را در خروجی چاپ میکند.

```
1  #include <iostream>
2  using namespace std;
3  int main(void)
4  {
5       int A[10],i;
6       for(i=0;i<10;i++)
7            cin>>A[i];
8       for(i=0;i<10;i++)
9            if (*(A+i) %2==0)
10                cout<<*(A+i)<<",";
11 }
```

دقت کنید که در دستور خط ۵ از ابتدا ۱۰ خانه از حافظه برای A در نظر گرفته شده است. در قسمت بعد با روش دیگری برای گرفتن فضا از حافظه، معروف به تخصیص حافظهٔ پویا آشنا می‌شویم.

برای آرایه‌های دو و چند بعدی نیز نام آرایه حاوی اشاره‌گر به اولین بایت از محل ذخیره‌سازی آرایه است. به عناصر دو و چندبعدی را نیز می‌توان با عملیات محاسباتی روی اشاره‌گر به محل شروع ذخیره‌سازی، دسترسی داشت. البته، اگر بخواهیم به درایه‌های خاصی از یک آرایهٔ دو بعدی دسترسی داشته باشیم، لازم است تعداد ستون‌های آن را بدانیم و برمبنای آن، آدرس درایهٔ مربوطه را بیابیم.

به عنوان مثال، فرض کنید آرایه دوبعدی A با ۵ سطر و ۷ ستون به صورت زیر تعریف شده باشد:

```
double A [5][7];
```

برای دسترسی به درایهٔ سطر سوم و ستون پنجم با استفاده از روش معمول، $A[۲][۴]$ به این درایه اشاره می‌کند. یادآوری می‌شود که اندیس‌های آرایه از صفر شروع می‌شود. آدرس شروع محلی که آرایه در آن ذخیره شده است در A قرار دارد. برای دسترسی به همین درایه از آرایه، با توجه به این که آرایه به صورت سطری در حافظه ذخیره می‌شود، باید ۲ سطر را رد کرد تا به سطر سوم رسید و سپس در این سطر به اندازهٔ ۴ درایه جلو رفت تا به اول محل ذخیره‌سازی این درایه رسید. بر مبنای این توضیحات، با توجه به این که در هر سطر آرایه، ۷ عضو ذخیره می‌شود، آدرس محل شروع ذخیره‌سازی $A[۲][۴]$، به صورت $A + ۲ \times ۷ + ۴$ است. و لذا $(A + ۲ \times ۷ + ۴)$* معادل $A[۲][۴]$ است.

۴.۸ تخصیص حافظه پویا

وقتی متغیری را تعریف می‌کنیم یعنی برای آن حافظه‌ای را اختصاص داده‌ایم و از زمان تعریف متغیر تا پایان بلوکی که متغیر در آن بلوک تعریف شده است، به اندازه سایز متغیر حافظه اشغال شده است. اما اگر خودمان بخواهیم آن را از بین ببریم نمی‌توانیم این کار را انجام دهیم. پس می‌توان گفت که حافظه آن به صورت ایستا گرفته شده است. در برخی موارد یک متغیر در یک بخشی از برنامه نیاز است و لازم نیست تا پایان برنامه حافظه‌ای برای آن اشغال شود از این رو اگر بتوانیم در حین انجام برنامه حافظه‌ای را به متغیرها اختصاص بدهیم و سپس حافظه را آزاد کنیم برنامه بهتری از لحاظ حافظه مصرفی داریم. در اینجا اشاره‌گر به برنامه‌نویس کمک می‌کند تا چنین راه‌حلی را در برنامه اجرایی کند.

با استفاده از تخصیص پویای حافظه می‌توان در حین اجرای برنامه، هر زمانی به فضای حافظه نیاز داریم و به مقدار نیاز از سیستم عامل حافظه گرفت و زمانی که دیگر نیازی به آن نبود، آن را دوباره در اختیار سیستم‌عامل قرار داد تا در اختیار همین برنامه یا برنامه‌های دیگر قرار گیرد.

۱.۴.۸ تخصیص حافظه پویا به یک متغیر

در تخصیص حافظهٔ پویا، یا به عبارتی تخصیص پویای حافظه، از دو عملگر new و delete استفاده می‌کنیم.

- **عملگر new:** با عملگر new یک حافظه پویا را درخواست می‌کنیم.
- **عملگر delete:** حافظهٔ تخصیص داده شده توسط عملگر new را می‌توان با استفاده از عملگر delete آزاد کرد.

قطعه کد زیر را در نظر بگیرید.

```
1 int *p;
2 float *m;
3 p= new int;
4 m= new float;
5    .
6    .
7    .
8 delete p;
9 delete m;
```

جهت تخصیص حافظهٔ پویا برای یک متغیر، ابتدا یک متغیر اشاره‌گری از نوع آن متغیر تعریف می‌کنیم (خطوط ۱و۲). سپس با عمگر new فضایی به اندازه نوع آن متغیر از حافظه دریافت و به متغیر داده می‌شود.

مثلاً در خط ۳ ابتدا دستور new int قسمتی از حافظه به اندازه int را می‌گیرد و بعد اشاره‌گری به آن قسمت از حافظه را برمی‌گرداند، سپس این آدرس درون متغیر اشاره‌گری p ریخته می‌شود، بنابراین بعد از این می‌توان در هرجا که خواستیم، با اشاره‌گر p به حافظه گرفته شده دسترسی پیدا کنیم. حالا می‌توان با اشاره‌گر p هر تغییری که می‌خواهیم در این حافظه بدهیم یا این که مقدار p را در متغیر اشاره‌گری دیگری بریزیم و با آن این تغییر را انجام دهیم. یعنی این حافظه درست مانند حافظه یک متغیر معمولی (که به صورت ایستا گرفته شده) می‌باشد. اگر new به هر دلیلی نتواند حافظه را بگیرد، استثنای bad_alloc رخ داده و اجرای برنامه متوقف می‌شود. برای جلوگیری از توقف برنامه، می‌توان دستور new را به صورت زیر استفاده کرد:

```
foo = new (nothrow) int;
```

در این صورت، در صورت عدم تخصیص حافظه به هر دلیل، اشاره‌گر NULL (یا معادل آن مقدار ۰) برگردانده می‌شود و داخل متغیر قرار می‌گیرد. لذا توصیه می‌شود بعد از تخصیص حافظهٔ پویا با دستور فوق، آدرس آن بررسی شود تا در صورت صفر بودن، از عدم تخصیص حافظه مطلع شده و دستورات مناسب آن را انجام داد. در صورت عدم بررسی و قرارگرفتن صفر در اشاره‌گر، تمام

عملیات بعدی روی این حافظه، با آدرس صفر انجام خواهد شد که ممکن است مخصوصاً در زمان ذخیره‌سازی مقداری در این حافظه، به مشکلات عدیده‌ای منجر شود.

اگر اشارهگر حافظه پویا در p باشد، دستور delete p حافظه را آزاد می‌کند، یعنی دیگر جایی که p به آن اشاره می‌کند را به عنوان فضای آزاد حافظه در نظر گرفته و می‌تواند در همین برنامه یا سایر برنامه استفاده شود. پس از آزادسازی یک حافظه بهتر است اشارهگر آن را با NULL مقدار دهیم تا بعداً هر کجا خواستیم، با مقایسهٔ آن با NULL متوجه شویم که آیا اشارهگر به محل معتبر و قابل استفاده‌ای اشاره می‌کند یا نه.

همانطور که می‌بینید حافظه گرفته شده، فقط با اشارهگر به آن قابل دسترسی است و نامی ندارد، بنابراین اگر ما فقط یک اشارهگر به این قسمت حافظه داشته باشیم و بعداً آن هم به جای دیگری اشاره کند، یعنی آدرس آن قسمت از حافظه گم شود، خود حافظه نه قابل استفاده است و نه حتی می‌توان آن را آزاد کرد زیرا هر دو عمل نیاز به آدرس حافظه دارد.

در کار با حافظه پویا باید به نکات زیر توجه کنیم:

- به هنگام تخصیص حافظه می‌توان به خانه‌ای که اشارهگر به آن اشاره می‌کند مقداری را نیز نسبت داد و برای این کار از شکل کلی زیر استفاده می‌شود.

```
نام اشارهگر = new نام متغیر (مقدار اولیه);
```

در خط ۴ می‌توان به جایی که m به آن اشاره می‌کند، مقدار ۱۴,۵ را به عنوان مقدار اولیه اختصاص داد.

```
m=new float (14.5) ;
```

- دقت کنید که اشارهگرها فضای به اندازه چهار بایت را در کامپیوترهای ۳۲ بیتی و هشت بایت را در کامپیوترهای ۶۴ بیتی اشغال می‌کنند مستقل از این که به چه نوع داده‌ای اشاره می‌کنند. دقت کنید که در خصوص کامپیوترهای ۶۴ بیتی، باید هم کامپیوتر، هم سیستم‌عامل و هم کامپایلر مورد استفاده ۶۴ بیتی باشد تا آدرس متغیرها ۶۴ بیتی و در نتیجه هشت بایتی باشد.

- همیشه باید حافظه‌ای که با دستور new گرفته‌اید را پس از پایان استفاده با دستور delete آزاد کنید وگرنه این حافظه به برنامه‌های دیگر هم اختصاص نمی‌یابد و مشکلی به نام نشت حافظه[1] بوجود می‌آید، یعنی حافظه نه استفاده می‌شود و نه برای دیگران قابل استفاده است. وجود نشت حافظه در یک برنامه در صورت طولانی بودن زمان اجرای برنامه و بالا بودن حجم نشت، ممکن است حتی به مشکلات کمبود حافظه در اجرای برنامه منجر شود.

[1] memory leak

۲.۴.۸ تخصیص حافظه پویا به یک آرایه

در مثال‌های بخش آرایه‌ها، بیان کردیم که در زمان تعریف آرایه، باید اندازهٔ آن مشخص و ثابت باشد. در نتیجه در زمانی که مثلاً تعداد داده‌های ورودی توسط کاربر تعیین می‌شد، آرایه را به قدری بزرگ در نظر گرفته می‌شد که برای ورودی‌های محتمل مناسب باشد. این کار یک روش غیرکامل و در عین حال غیربهینه برای حل مشکل بود زیرا عملاً مشکل را کاملاً حل نمی‌کرد و مقدار زیادی حافظه بدون آنکه برنامه از آن در اکثر موارد استفاده کند، اشغال می‌شد. در اینجا با استفاده از تخصیص حافظهٔ پویا، امکان تعریف آرایه را دقیقاً به میزان دلخواه در زمان اجرای برنامه خواهیم داشت.

جهت تخصیص حافظه پویا به آرایه ابتدا یک اشارهگر تعریف می‌کنیم:

```
نام متغیر * نام متغیر;
```

تخصیص حافظهٔ پویا برای آرایه‌ها مشابه متغیرهای معمول است. تنها تفاوت آن این است که بعد از نوع متغیر، داخل کروشه، تعداد درایهٔ مورد نیاز برای آرایه را مشخص می‌کنیم. حال با دستور زیر تعداد خانه‌های مورد نیاز را برای آن در نظر می‌گیریم.

```
نام اشارهگر = new نوع متغیر [اندازه آرایه];
```

به عنوان مثال آرایه ۱۰ عنصری از اعداد اعشاری به صورت زیر تعریف می‌شود:

```
double *p;
p=new double[10];
```

پس از اتمام کار باید فضای تخصیص یافته را به صورت زیر آزاد کنیم.

```
delete [] نام اشارهگر;
```

به عبارت دیگر، برای آزاد کردن فضای تخصیص داده شده به یک آرایه، باید بعد از کلمهٔ کلیدی delete ، کروشه قرار داد. همانطور که قبلاً بیان شد، هنگام تعریف یک آرایه معمولی باید تعداد عناصر آن را با یک مقدار ثابت تعیین کرد (مثلاً یک عدد یا یک متغیر ثابت)، اما مزیت تخصیص پویا این است که می‌توان تعداد عناصر را با یک متغیر هم مشخص کرد و براساس مقدار آن تعداد عناصر مشخص می‌شود. مثلاً اگر دستور int n=4 را داشته باشیم، مقدار متغیر n مشخص می‌شود. حتی ممکن است مقدار n حاصل یک عبارت محاسباتی باشد. از دستور int *p=new int [n]; می‌توان برای تخصیص حافظه پویا برای یک آرایه با n درایه از نوع int استفاده کرد. بعد از این دستور، p به محلی از حافظه که برای یک آرایه پویا با n عضو تخصیص داده شده اشاره می‌کند.

به دو شیوه می‌توان به عناصر آرایه دسترسی پیدا کرد. روش اول، همان روش آرایه‌های معمولی است، یعنی جلوی متغیر اشارهگری کروشه قرار داد و داخل کروشه، اندیس را می‌آوریم. روش دوم انجام عملیات محاسباتی روی اشارهگرها و استفاده از عملگر * است.

مثال ۱.۴.۸. *برنامه‌ای بنویسید که تعدادی عدد را از کاربر دریافت و آنها را به صورت صعودی مرتب و چاپ کند.*

حل:

```
1  #include  <iostream>
2  using  namespace std;
3  int    sort(double  *, int );
4  int  main(void)
5  {
6       int  n, i ;
7       double *p;
8       cout<< "Enter number of elements: ";
9       cin >>n;
10      p=new double[n];
11      if (p == NULL)
12      {
13           cout<<"memory not allocated";
14           return  0;
15      }
16      for(i=0;i<n;i++)
17      {
18           cout<<"Enter number "<<i+1<<":";
19           cin>>*(p+i);
20      }
21      sort(p,n);
22      cout<<"Sorted  list :";
23      for(i=0;i<n;i++)
24      {
25           cout<<*(p+i)<<",";
26      }
27 }
28 int  sort(double *p, int  Len)
29 {
30      double  temp;
31      int  i , j ;
32      for(i=Len-1;i>0;i--)
33           for(j=0;j<i; j++)
34                if (*(p+j)>*(p+j+1))
35                {
36                     temp=*(p+j);
37                     *(p+j)=*(p+j+1);
38                     *(p+j+1)=temp;
```



```
39                }
40 }
```

□

همانطور که قبلاً بیان شد رشته‌ها آرایه‌ای از کاراکترها هستند و مشابه بحث اشاره‌گر و آرایه، می‌توان از اشاره‌گر در رشته‌ها نیز استفاده کرد.

مثال زیر، برنامه‌ای است که تعداد حروف a موجود در متن را محاسبه و چاپ می‌کند.

```
1  #include  <iostream>
2  using  namespace std;
3  int  main(void)
4  {
5       char  *S="programming languages";
6       int  m=0;
7       for(  ;  *S  ;  S++)
8            if (  *S  == 'a')
9                 m++;
10      cout<<m;
11  }
```

در مثال فوق، در خط ۵، به تعداد کاراکترهای لازم برای رشته بعلاوهٔ یک حافظه برای رشته در نظر گرفته می‌شود و آدرس محل ذخیره‌سازی رشته در S قرار می‌گیرد. در حلقهٔ for، S* به معنی محتوای محلی است که S به آن اشاره می‌کند که با توجه به این که در انتهای رشته کاراکتر '\0' که معادل صفر است قرار می‌گیرد، این حلقه تا رسیدن به انتهای رشته ادامه پیدا می‌کند. همچنین S++ با توجه به شکل عملیات روی اشاره‌گرها، باعث اشاره کردن S به محل بعدی در رشته خواهد شد.

۳.۴.۸ تخصیص حافظهٔ پویا برای آرایه‌های دوبعدی و چندبعدی

تخصیص حافظهٔ پویا برای آرایه‌های دوبعدی و چندبعدی به چند روش ممکن است.

روش اول:

استفاده از روشی مشابه تخصیص حافظهٔ پویا برای آرایه‌های یک بعدی است، با این تفاوت که برای تعداد عناصر آرایه، تعداد کل درایه‌های لازم برای ماتریس دو یا چندبعدی را استفاده می‌کنیم. مثلاً برای تخصیص حافظهٔ پویا برای یک ماتریس ۵×۴ از نوع صحیح، به صورت زیر عمل می‌کنیم:

```
int  *A;
A=new int[20];
```

در صورت تخصیص حافظهٔ پویا به این صورت، امکان ارجاع به درایه‌های آرایه به روش معمول ممکن نیست بلکه باید بر مبنای محاسبات، به درایه‌ها دسترسی داشت. به عنوان مثال، به درایهٔ سطر دوم و ستون سوم می‌توان به یکی از صورت‌های زیر دسترسی داشت:

```
*(A+2*5+2)
A[2*5+3]
```

روش دوم:

در این روش عملاً آرایهٔ دوبعدی به نوعی شبیه‌سازی می‌شود و این تکنیک را می‌توان به آرایه‌های چندبعدی نیز تعمیم داد. جهت تخصیص آرایهٔ دوبعدی $m \times n$، ابتدا یک آرایه از نوع اشاره‌گر به طول m برای نگهداری اشاره‌گرهای سطرهای آرایه تخصیص می‌دهیم. با توجه به این که آرایه از نوع اشاره‌گری، مثلاً int * ، است، حاصل تخصیص آرایه از نوع int ** خواهد بود. دستور زیر این کار را انجام خواهد داد:

```
int **A;
A=new int *[m];
```

حال برای هر سطر ماتریس، حافظه تخصیص داده و اشاره‌گر مربوطه را در درایهٔ متناظر در آرایه قرار می‌دهیم. این کار با استفاده از حلقهٔ زیر انجام می‌دهیم:

```
for( int i=0;i<m;i++)
    A[i]=new int[n];
```

با این ترتیب، به سادگی A[i][j] به درایهٔ سطر iام و ستون jام آرایه اشاره می‌کند. دلیل این امر آن است که A[i] به آدرس محل شروع سطر iام آرایه اشاره می‌کند و لذا A[i][j] به محل jام از سطر iام اشاره می‌کند.

لذا اگر بخواهیم برنامه‌ای را که قبلاً با استفاده از آرایهٔ دوبعدی نوشته شده است را به تخصیص حافظهٔ پویا مجهز کنیم، کافی است صرفاً دستور تعریف آرایه را با دستورات تخصیص حافظهٔ پویا جایگزین کنیم و سایر قسمت‌های برنامه تغییر نخواهد کرد.

دقت کنید که جهت آزاد کردن حافظهٔ تخصیص داده شده باید در جهت عکس ترتیب تخصیص حافظه عمل کرد و در این ترتیب، حافظه‌های تخصیص داده شده را آزاد کرد. دستورات زیر این کار را انجام خواهند داد.

```
1 for( i=0;i<m;i++)
2       delete [] A[i];
3 delete [] A;
```

در مثال زیر، یک ماتریس با ابعاد داده شده را با این روش تعریف کرده و استفاده می‌کند.

```
1 #include <iostream>
2 #include <cstddef>
3 using namespace std;
4 int main(void)
5 {
6       int **matrix,count, i , j ,row,col;
```



```
7       cout<<"Enter number of rows and columns of the matrix:";
8       cin>>row>>col;
9       // dynamically allocate an array
10      matrix = new (nothrow) int *[row];
11      if (matrix == NULL)
12      {
13        // handle the error
14      }
15      for (count = 0; count < row; count++)
16      {
17          matrix[count] = new (nothrow) int[col];
18          if (matrix[count] == NULL)
19          {
20            // handle the error
21          }
22      }
23      // input element for matrix
24      cout << endl << "Now enter the element for the matrix ... ";
25      for (i=0; i < row; i++)
26      {
27          for (j=0; j < col; j++)
28          {
29              cout << endl << "Row " << (i+1) << " Col " << (j+1) << " :";
30              cin >> matrix[i][j]; // is there any equivalent declaration
                    here?
31          }
32      }
33      // free dynamically allocated memory
34      for(i = 0 ; i < row ; i++)
35          delete [] matrix[i] ;
36      delete [] matrix ;
37  }
```

همین تکنیک برای آرایهٔ دوبعدی از کاراکترها نیز قابل استفاده است. علاوه بر این با توجه به این که هر سطر از یک ماتریس کاراکتری برای نگهداری یک رشته استفاده می‌شود، می‌توانیم برای هر سطر متناسب با طول رشته، حافظه اختصاص دهیم و از اتلاف حافظه در این قسمت جلوگیری کنیم.

دقت کنید که روش قبلی برای مرتب‌سازی رشته‌ها، عیناً برای این روش جدید تخصیص حافظه برای نگهداری رشته‌ها قابل استفاده نیست، زیرا در این روش از جابجایی رشته‌ها برای مرتب‌سازی استفاده می‌کند. اما چون در این روش، برای هر رشته، صرفاً به اندازهٔ طول آن حافظه در نظر گرفته

شده است، جابجایی دو رشته با طول غیرهمسان به اشکال برخورد می‌کند. البته با تغییری ساده که سریع‌تر نیز هست، می‌توان این مشکل را حل کرد. برای جابجایی دو رشته (دو سطر از ماتریس) کافی است صرفاً اشاره‌گر به آنها را در آرایه‌ای که آدرس سطرها را نگهداری می‌کند جابجا کنیم. دقت کنید که این روش برای آرایه‌هایی که به صورت معمول تعریف می‌شوند قابل استفاده نیست.

در هر دو برنامه زیر هدف دریافت اسامی ۵ دانشجو است و سپس جستجو برای یافتن نام دانشجویی در لیست است.

```
1  #include  <iostream>
2  #include  < cstring >
3  using  namespace std;
4  int  main(void)
5  {
6       char  name[5][10];
7       char  s [10];
8       int  i ;
9       for ( i =0; i <5; i ++)
10      {
11           cout<<"Enter Next Name:";
12           cin >>name[i];
13      }
14      cout<<"Enter a  name:";
15      cin >>s;
16      for ( i =0; i <5; i ++)
17           if (! strcmp(name[i ], s))
18              break;
19      if ( i ==5)
20           cout<<s<<" is not  in  the   list ";
21      else
22           cout<<s<<" is in  the   list ";
23 }
```

در برنامهٔ زیر همین برنامه، با تخصیص حافظهٔ پویا آمده است.

```
1  #include  <iostream>
2  #include  < cstring >
3  using  namespace std;
4  int  main(void)
5  {
6       char  *name[5];
7       char  s [10];
8       int  i ;
9       for ( i =0; i <5; i ++)
```



```
10      {
11           cout<<"Enter Next Name:";
12           cin >>s;
13           name[i]= new char[ strlen (s)+1];
14           strcpy (name[i ], s) ;
15      }
16      cout<<"Enter a  name:";
17      cin >>s;
18      for ( i =0; i <5; i ++)
19           if (! strcmp(name[i ], s))
20              break;
21      if ( i ==5)
22           cout<<s<<" is not  in  the   list ";
23      else
24           cout<<s<<" is in  the   list ";
25 }
```

شکل حافظه مورد استفاده در این دو برنامه در شکل زیر آمده است. مشخص است که روش دوم، حافظهٔ کمتری نسبت به روش اول استفاده می‌کند.

مرتب‌سازی رشته‌ها که با استفاده از این روش تخصیص داده شده است به عنوان تمرین به خواننده واگذار می‌شود.

۵.۸ پارامتر مرجع

پارامتر مرجع به عنوان جانشین یك متغیر تعریف می‌شود و تغییر در آن مانند تغییر در متغیر اصلی است. همچنین تغییر در متغیر اصلی باعث تغییر آن می‌شود. پارامتر مرجع به طور مستقیم در توابع نقش بازی می‌کند و تغییراتی که در تابع ایجاد می‌شود این پارامتر را نیز دچار تغییر می‌کند.

نحوه تعریف پارامتر مرجع به این صورت است:

نوع مرجع &نام مرجع=متغیر;

ظاهر این تعریف مانند تعریف یک متغیر است با این تفاوت که قبل از نام مرجع یک & می‌آید. به عنوان مثال با دستور زیر یک مرجع x از نوع اعداد صحیح تعریف می‌گردد.

```
int &x;
```

توجه به این نکته مهم می‌باشد که متغیری که پارامتر مرجع جانشین آن است می‌بایست در همان لحظه تعریف به پارامتر مرجع شناسانده شود. همچنین باید توجه داشت وقتی یك پارامتر مرجع را جانشین یك متغیر می‌کنیم دیگر نمی‌توانیم جانشین متغیری دیگر کنیم. همچنین مرجع باید هم‌نوع متغیری باشد که به آن ارجاع می‌دهد.

به قطعه کد زیر که مفهوم پارامتر مرجع را مشخص می‌کند توجه کنید:

```
1  #include <iostream>
2  using namespace std;
3  int main(void)
4  {
5       int j=0;
6       int &i=j;
7       i+=2;
8       cout<<j<<",";
9       j--;
10      cout<<i;
11 }
```

در خط ششم یك پارامتر مرجع تعریف کرده‌ایم و آن را به j نسبت داده‌ایم. در خط هفتم به پارامتر مرجع ۲ واحد اضافه کرده‌ایم و در خط هشتم مشاهده می‌شود که متغیر نیز ۲ واحد بیشتر شده است. بنابراین تغییر در پارامتر مرجع با تغییر در متغیر یکی است. در خط نهم از متغیر یك واحد کم کرده‌ایم و در خط دهم می‌بینیم که از پارامتر مرجع نیز یك واحد کم شده است. بنابراین تغییر در متغیر مانند تغییر در پارامتر مرجع می‌باشد.

در بحث پارامترهای مرجع به نکات زیر باید توجه کرد:

- می‌توان بیش از یک مرجع برای یک متغیر تعریف کرد پس ما یک متغیر داریم با چندین نام نه این که برای هر نام یک متغیر و فضایی از حافظه تخصیص داده شود.
- نمی‌توان بعداً تعیین کرد که یک مرجع به متغیر دیگری ارجاع دهد، پس یک مرجع همیشه به یک متغیر ارجاع می‌دهد.
- اگر یک مرجع به مرجع دیگری ارجاع دهد(با نام آن مقدار اولیه گیرد) در اصل بازهم به متغیر اصلی ارجاع می‌دهد و با این که با نام متغیر اصلی مقدار اولیه بگیرد فرقی نمی‌کند.

۱.۵.۸ متغیر مرجع و تابع

یکی از کاربردهای متغیر مرجع این است که توابعی تعریف کنیم که به صورت خودکار پارامترها را به صورت با ارجاع ارسال نماید. قبلاً در مورد نحوه ارسال پارامترها به توابع بحث شد که فراخوانی

توابع با مقدار و با ارجاع بود. در C++ برای فراخوانی با ارجاع به دو شیوه می‌توان عمل کرد یکی پارامترها را به صورت اشاره‌گر تعریف کنیم که در قسمت قبل توضیح داده شد و دیگری استفاده از پارامترهای مرجع است در اکثر حالات مناسب تر است از پارامتر مرجع استفاده شود. پس بطور کلی روش ارسال پارامتر به شکل زیر خلاصه می‌شود.

۱. **فراخوانی با مقدار:** تغییر در داخل تابع فرعی تغییری روی متغیر در تابع اصلی ندارد.
۲. **فراخوانی با ارجاع:**

- **با اشاره‌گر:** تغییر در داخل تابع فرعی باعث تغییر در تابع اصلی می‌شود.
- **با پارامتر مرجع:** تغییر در داخل تابع فرعی باعث تغییر در تابع اصلی می‌شود.

برنامه زیر عملکرد پارامترهای مرجع درکار با توابع را نشان می‌دهد.

```
1  #include <iostream>
2  using namespace std;
3  void func(int &p,int n)
4  {
5       n++;
6       p++;
7  }
8  int main(void)
9  {
10      int n=10,m=12;
11      func(m,n);
12      cout<<n<<","<<m;
13 }
```

۲.۵.۸ متغیرهای ثابت: استفاده از const

هرچند عنوان این بخش کمی متناقض به نظر می‌رسد اما در برنامه نویسی امکان تعریف متغیرهایی که مقدار آن قابل تغییر نیست، وجود دارد که به آنها، متغیرهای ثابت گوییم. برای تعریف یک متغیر ثابت، کافی است در دستور تعریف متغیر قبل و یا بعد از بیان نوع آن، کلمه کلیدی const قرار دهید. با توجه به قابل تغییر نبودن یک متغیر ثابت، باید در زمان تعریف مقدار آن را نیز مشخص کرد. به عنوان مثال، دو دستور زیر، یک متغیر ثابت از نوع صحیح تعریف می‌کند و با مقدار ۵ مقداردهی می‌کند.

```
const int x=5;
int const x=5;
```

از متغیرهای ثابت در موارد متعددی استفاده می‌شود. مثلاً اگر در یک برنامه، ثابتی وجود داشته باشد که در محل‌های مختلف برنامه ظاهر شود ولی این احتمال داده شود که در آینده بخواهیم برنامه را برای مقداری متفاوت با مقدار ثابت فعلی تغییر دهیم، می‌توان این مقدار ثابت را در متغیر ثابت قرار داد و از آن در برنامه استفاده کرد. در صورت نیاز به تغییر این ثابت، تنها کافی است در محل تعریف متغیر ثابت، مقدار آن را عوض کرد و نیازی به تغییر سایر قسمت‌های برنامه نیست. در حالی که اگر ثابت، مستقیماً در برنامه استفاده شود، برای تغییر آن باید تمام محل‌هایی که ثابت در برنامه ظاهر شده است را پیدا کرد و آن را تغییر داد. بدیهی است این کار هم وقت‌گیر است و هم امکان اشتباه دارد زیرا فقط عوض نکردن یک مورد، باعث اشتباه در اجرای برنامه خواهد شد.

کاربرد دیگر استفاده از const، در مواردی است که متغیری که به صورت ارجاعی (با استفاده از اشاره‌گر یا متغیر مرجع) پارامتر یک تابع است را در تابع به صورت ثابت تعریف کنیم، به این معنی که تابع امکان تغییر آن را نداشته باشد. مثلاً فرض کنید تابعی یک آرایه را به عنوان پارامتر دریافت می‌کند تا از آن برای محاسبات خود استفاده کند اما نیازی به تغییر آرایه در تابع نیست. بدیهی است که امکان تغییر درایه‌ای از این آرایه در تابع تغییر وجود دارد. اما اگر پارامتر مربوط به این آرایه در تابع به صورت ثابت تعریف شود، به این معنی خواهد بود که تابع، امکان تغییر آن آرایه را ندارد و در صورت وجود دستوری در تابع که مقدار آرایه را عوض کند، در موقع کامپایل پیغام خطا خواهد داد. لذا با این تغییر، ما مطمئن خواهیم شد که آرایه در تابع تغییر نمی‌کند. به عنوان یک مثال عملی تابع strcpy (str1 , str2) را در نظر بگیرید که str2 را در str1 کپی می‌کند. اگر این تابع به صورت strcpy (char * str1 , char * str2) تعریف شود، امکان تغییر هر دو رشته در تابع وجود دارد ولی اگر تابع به صورت strcpy (char * str1 , const char * str2) تعریف شود، امکان تغییر str2 در تابع وجود نخواهد داشت که در خصوص عملکرد این تابع کاملاً منطقی است.

اگر بخواهیم یک مرجع نتواند مقدار متغیری که به آن ارجاع می‌دهد را تغییر بدهد، از کلمه کلیدی const در تعریف متغیر مرجع استفاده می‌کنیم:

```
int  n=0;
const  int  &m = n;
```

مشخص می‌کند که m به n ارجاع می‌دهد اما نمی‌تواند مقدار آن را تغییر دهد، بنابراین دستوری مانند m=2; خطای زمان کامپایل تولید می‌کند. دقت کنید که دستور n=2; بدون خطا اجرا می‌شود و فقط نمی‌توان n را از طریق متغیر مرجع ثابت آن تغییر داد.

متغیرهای اشاره‌گری ثابت

مشابه سایر متغیرها، متغیرهای اشاره‌گری را نیز می‌توان به صورت ثابت تعریف کرد. اما این سوال پیش می‌آید که اگر متغیر اشاره‌گری از نوع ثابت تعریف شود، در در این صورت همان متغیر ثابت خواهد شد یا جایی که متغیر اشاره‌گری به آن اشاره می‌کند. مثلاً تعریف زیر را در نظر بگیرید:

```
int  x;
```



```
const  int  *xptr=&x;
```

حال با این تعریف آیا دستور *xptr=5; با خطا مواجه خواهد شد یا xptr=&y; که در آن y متغیری است که از نوع صحیح تعریف شده است؟

جواب این سوال این است که دستور *xptr=5; با خطا مواجه می‌شود به این معنی که امکان تغییر متغیری که xptr به آن اشاره می‌کند از طریق این اشاره‌گر وجود ندارد. دستور دیگر که محتوای xptr را تغییر می‌دهد بدون مشکل اجرا خواهد شد. پس در این تعریف، خود xptr متغیر ثابت نخواهد بود و امکان تغییر محتوای آن وجود دارد. اما اگر بخواهیم متغیر اشاره‌گری ثابت تعریف کنیم چطور؟ دستورات زیر این کار را انجام خواهد داد:

```
int  x;
int  * const  xptr=&x;
```

با این تغییر، دستور *xptr=5; بدون خطا اجرا می‌شود به این معنی که امکان تغییر متغیری که xptr به آن اشاره می‌کند از طریق این اشاره‌گر وجود دارد، اما دستور دیگر که محتوای xptr را تغییر می‌دهد با خطا مواجه خواهد شد. با تعریف زیر نه می‌توان xptr را تغییر داد و نه جایی که xptr به آن اشاره می‌کند:

```
int  x;
const  int  * const  xptr=&x;
```

تمرین

۱. خروجی برنامه زیر چیست؟

```
1  #include <iostream>
2  using  namespace std;
3  void  func( int  &p, int  n)
4  {
5       n++;
6       p++;
7  }
8  int  main(void)
9  {
10       int  n=10,m=12;
11       func(m,n);
12       cout<<n<<","<<m;
13  }
```

۲. تابع زیر را به کمک عملگرهای اشاره‌گر باز نویسی کنید؟

```
1  void  insertion_sort ( int  arr[ ],  int  n )
```

```
{
    int i, j, t;
    for( i = 1 ; i < n ; i++)
    {
        t = arr[ i ];
        for( j = i  ; j > 0 ; j-- )
        {
            if ( t >= arr[ j - 1 ] )
                break;
            arr[ j ] = arr[ j - 1 ];
        }
        arr[ j ] = t;
    }
}
```

۳. اشاره‌گر سرگردان چیست و از وجود اشاره‌گرهای سرگردان چه نتایج نامطلوبی ممکن است اتفاق بیفتد؟

۴. برنامه‌ای بنویسید که با کمک آرایه‌های پویا لیستی از اسامی دانشجویان و معدل آن‌ها را در سیستم ذخیره کند. سپس اسامی دانشجویانی که بیشترین وکمترین معدل را کسب کرده اند در خروجی چاپ کند.

۵. خروجی برنامهٔ زیر چیست؟

```
#include <iostream>
using namespace std;
int main(void)
{
    int A[32],*Aptr;
    Aptr=A;
    for(int i=0;i<32;i++)
        *Aptr++=i*i;
    for(int i=0;i<32;i++)
        cout << A[i] << ",";
}
```

پیوست آ. راهنمای نصب کامپایلر و اجرای برنامه‌های C++

در این فصل، به روش نصب کامپایلرهای C++ روی کامپیوترهای شخصی و گوشی‌های هوشمند و تبلت می‌پردازیم. در خصوص هر کدام، روی پلتفرم‌های معروف جزئیات بیان شده است ولی نصب روی پلتفرم‌های دیگر نیز به طور مشابه و با در نظر گرفتن پلتفرم، قابل انجام خواهد بود.

آ.۱ کامپایلر روی کامپیوترهای شخصی

آ.۱.۱ راهنمای نصب کامپایلر **CodeBlocks** روی کامپیوترهای شخصی

اگر کاربر ویندوز هستید، ابتدا به یکی از این دو آدرس بروید و فایل نصب را بارگذاری کنید.

http://sourceforge.net/projects/codeblocks/files/Binaries/13.12/Windows/codeblocks-13.12mingw-setup.exe/download

یا

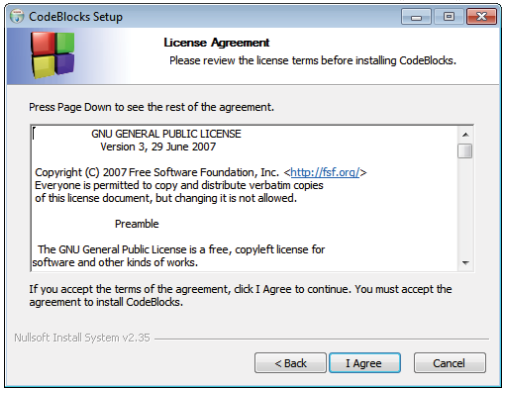
http://prdownload.berlios.de/codeblocks/codeblocks-13.12-setup.exe

برای نصب مراحل زیر را طی کنید:

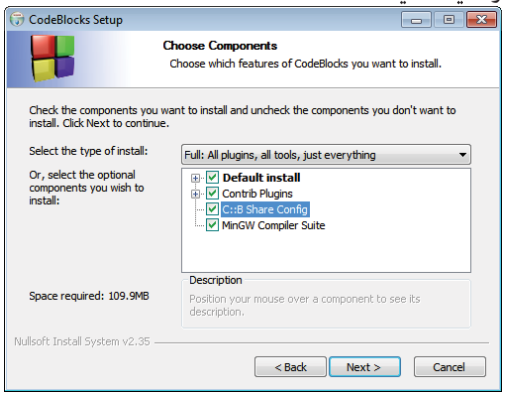
۱. فایل دانلود شده را اجرا کنید.

کلید Next را کلیک کنید.

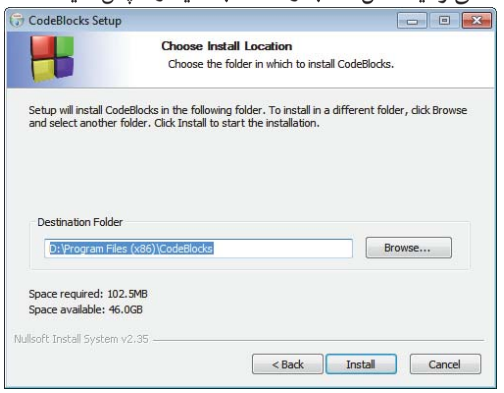
۲. روی I Agree کلیک کنید.



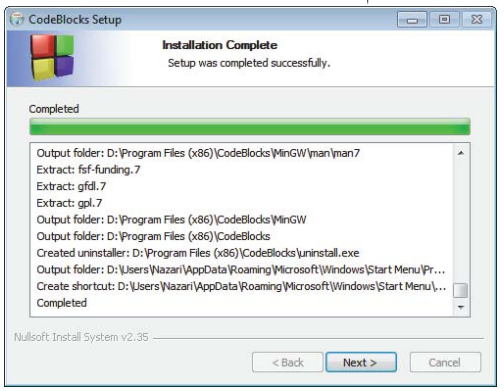
۳. کلید Next را کلیک کنید.



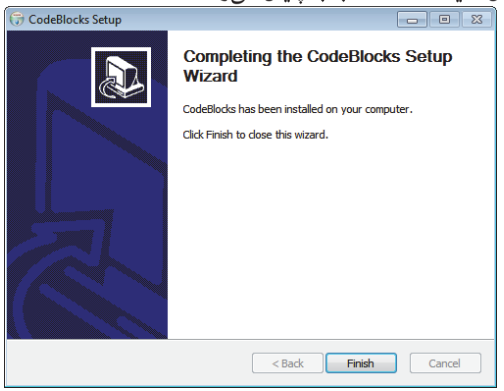
۴. در این مرحله می‌توانید محل نصب را انتخاب کنید و سپس کلید Install را فشار دهید.



۵. منتظر بمانید تا نصب انجام شود.



۶. با فشار دادن کلید Finish نصب به پایان می‌رسد.



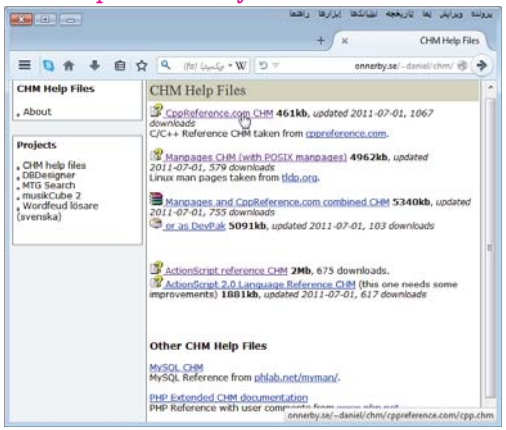
به طور پیش‌فرض میان‌بر CodeBlocks در Desktop ساخته می‌شود. در غیر اینصورت می‌توانید در منوی Start آن را پیدا کنید.

آ.۲.۱ نصب و تنظیم Help برای CodeBlocks

۱. در قدم اول نیاز است فایل‌های Help نرم‌افزار را از اینترنت دانلود کنید.

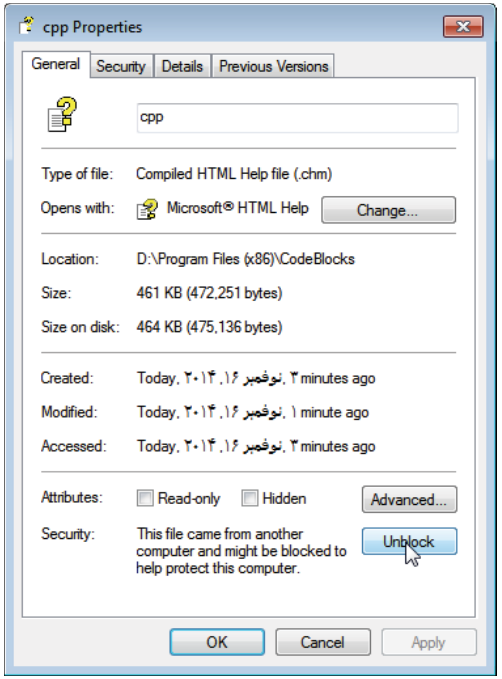
به عنوان مثال می‌توانید با مراجعه به این آدرس این فایل‌ها را بارگذاری کنید:

http://onnerby.se/~daniel/chm

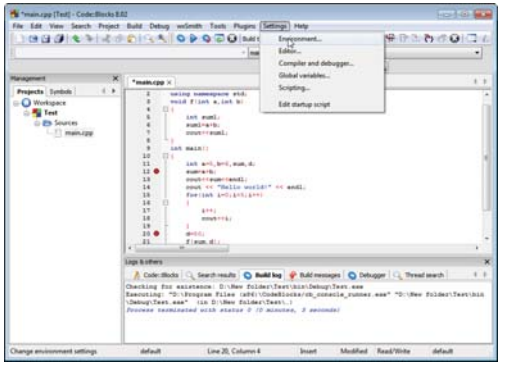


۲. فایل دریافتی را در پوشه محل نصب نرم‌افزار ذخیره کنید. به عنوان نمونه این محل می‌تواند C:\Program Files\Codeblocks باشد.

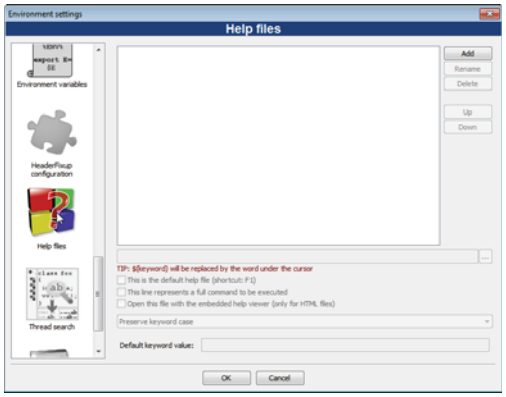
۳. بعد از ذخیره فایل در پوشه CodeBlocks روی فایل راست کلیک کرده و properties را انتخاب کنید. سپس فایل را Unblock کنید.



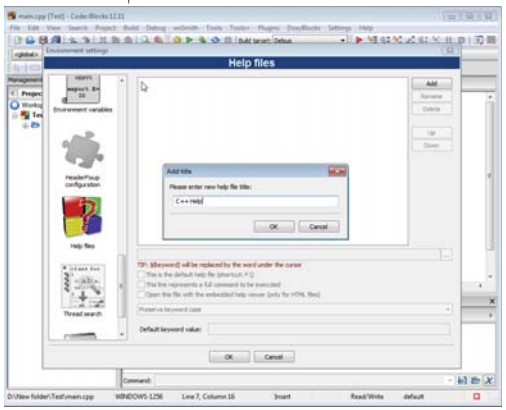
۴. در قدم بعد در CodeBlocks از منوی Settings گزینه Environment را انتخاب کنید.



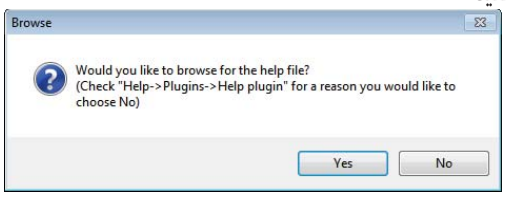
۵. در سمت چپ پنجره باز شده Help file را انتخاب کنید.



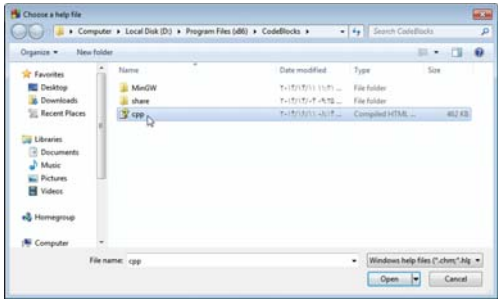
۶. کلید Add را فشار داده و در قسمت اضافه کردن title نام دلخواهی را تایپ کنید.



۷. با زدن کلید OK در مرحله بعد برای انتخاب مسیر فایل Help در پنجره نمایش داده شده Yes را انتخاب کنید.



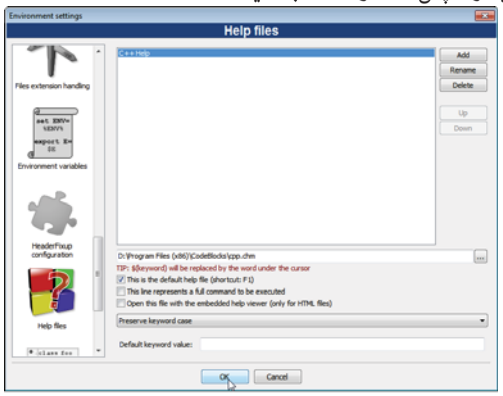
۸. مسیر فایل Help را تعیین کرده و فایل را انتخاب کنید.



۹. به منظور تعیین کلید F1 برای باز کردن این فایل جهت Help نرم‌افزار گزینه

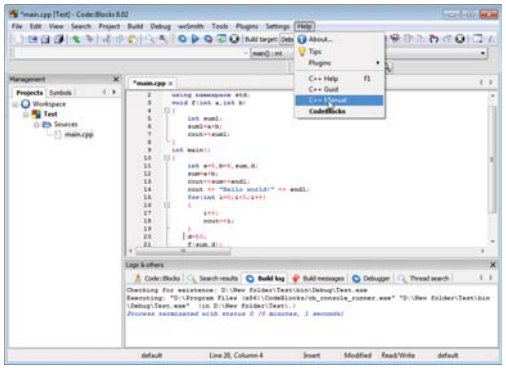
This is the default help file (shortcut: F1)

را تیک بزنید و سپس OK را انتخاب کنید.



۱۰. تنظیمات به پایان رسید. برای باز کردن این فایل Help کافی است روی کلمه مورد نظر در محیط نرم‌افزار کلیک کرده و کلید F1 را فشار دهید.

البته می‌توانید بیش از یک فایل را به همین روش جهت Help نرم‌افزار تنظیم کنید که در این حالت برای یکی از فایل ها کلید F1 را به عنوان کلید میانبر انتخاب کنید. برای دیدن بقیه فایل‌ها می‌توانید از منوی Help، این راهنماها را انتخاب کنید.

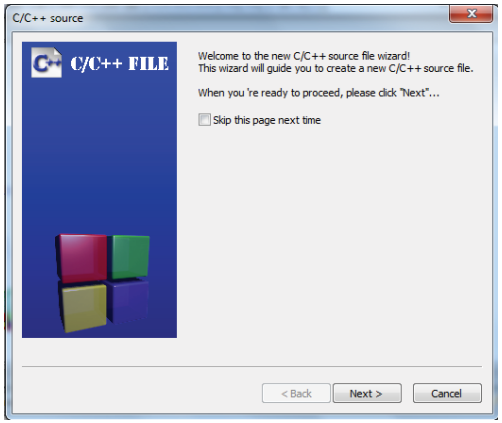


آ.۳.۱ شیوه ورود برنامه و اجرای آن

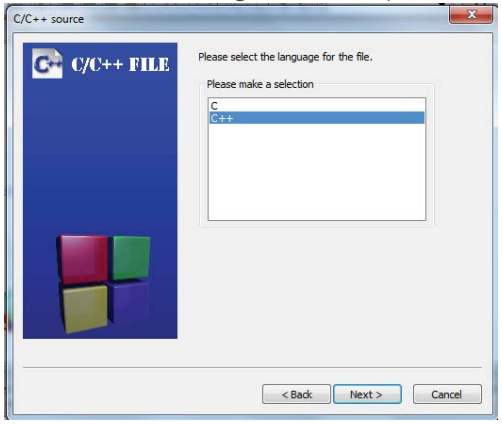
نوشتن برنامه در **CodeBlocks** (روش اول)

۱. از منوی File گزینه New و سپس File را انتخاب کنید. پنجرهٔ زیر ظاهر می‌شود.

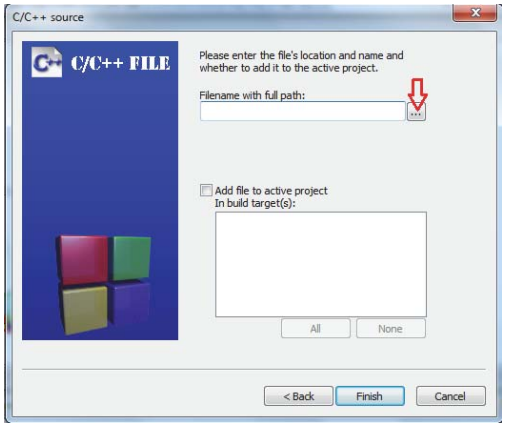
۲. روی آیکون C/C++ Source کلیک کنید. پنجرهٔ زیر ظاهر می‌شود.



۳. روی Next کلیک کنید. پنجرهٔ زیر ظاهر می‌شود.



۴. روی C++ و سپس روی Next کلیک کنید. پنجرهٔ زیر ظاهر می‌شود.

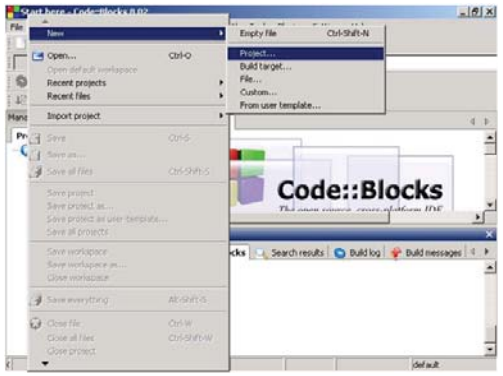


۵. در این پنجره، روی ... کلیک کنید. پنجره زیر ظاهر می‌شود. در این پنجره می‌توانید به مسیر مورد نظر جهت ذخیره کردن برنامه بروید و سپس در قسمت Filename نام فایل را وارد کنید. نام فایل باید انگلیسی و بدون فاصله باشد و همچنین پسوند آن باید cpp. باشد. دقت کنید که با نگذاشتن cpp. به عنوان پسوند فایل، فایل با پسوند c. ایجاد می‌شود که در مراحل بعدی اجرای برنامه را با مشکل مواجه می‌کند.

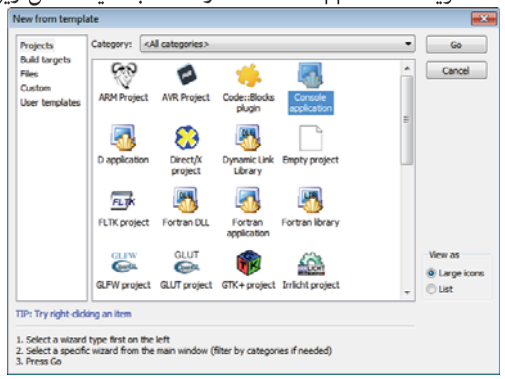
۶. پس از انتخاب نام فایل، روی Save کلیک کنید. با این کار پنجره بسته شده و از پنجرهٔ قبلی نیز Finish را کلیک کنید. حال می‌توانید برنامه خود را بنویسید. برای اجرای آن به قسمت ۳.۱.آ بروید.

نوشتن برنامه در **CodeBlocks** (روش دوم)

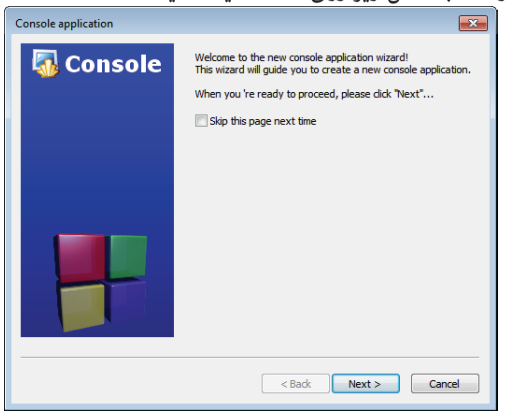
۱. از منوی File گزینه New و سپس Project را انتخاب کنید.



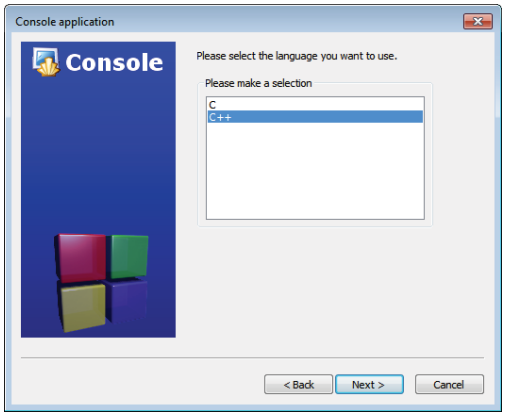
۲. در کادر باز شده گزینه Console Application را انتخاب کنید. شکل زیر را ببینید.



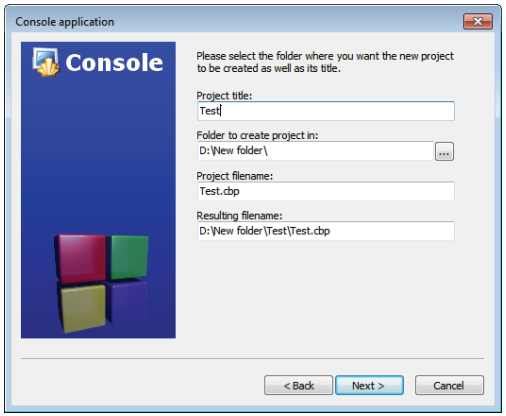
۳. در پنجرهٔ باز شده به شکل زیر روی Next کلیک کنید.



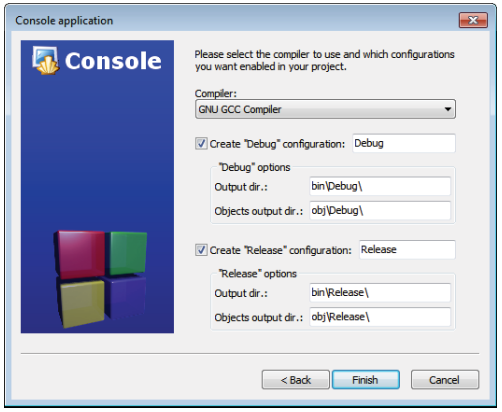
۴. در کادر باز شده C++ را انتخاب کنید و سپس Next را فشار دهید.



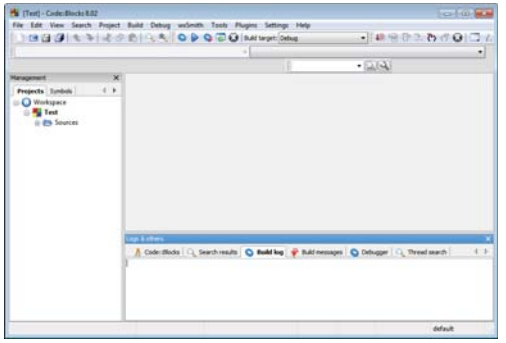
۵. در این مرحله نام و محل ذخیره پروژه را مشخص کنید. در قسمت Project title نام و در قسمت Folder to create project in محل ذخیره را تعیین کنید و سپس Next را فشار دهید.



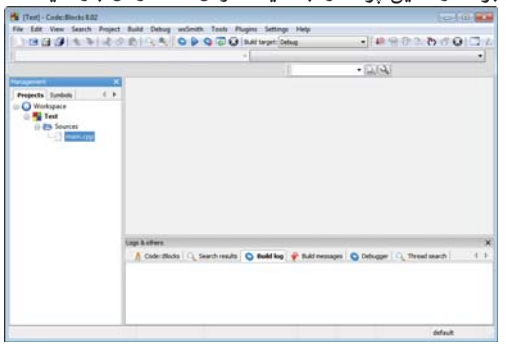
۶. پیش فرض‌های این مرحله را تغییر نمی‌دهیم و در نهایت Finish را فشار می‌دهیم.



نرم‌افزار محیط برنامه‌نویسی جدیدی را برای شما باز خواهد کرد.

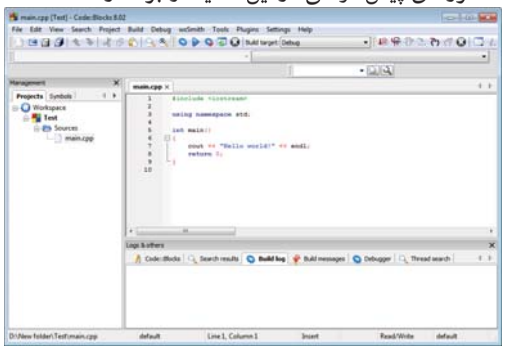


در سمت چپ محیط این نرم‌افزار، پنل Management را مشاهده می‌کنید. اولین گزینه در این پنل Project هست. در این قسمت می‌توانید نام پروژه خود را ببینید و در پایین آن پوشه Sources وجود دارد. این پوشه را با کلیک کردن + کنار آن باز کنید.



در این پوشه فایل main.cpp وجود دارد. این فایل در زمان ایجاد برنامه به طور خودکار ساخته می‌شود.

با کلیک کردن روی این فایل و باز کردن آن، محیط برنامه‌نویسی برای شما در سمت راست باز می‌شود. دستورهای پیش فرضی در این محیط وجود دارد.

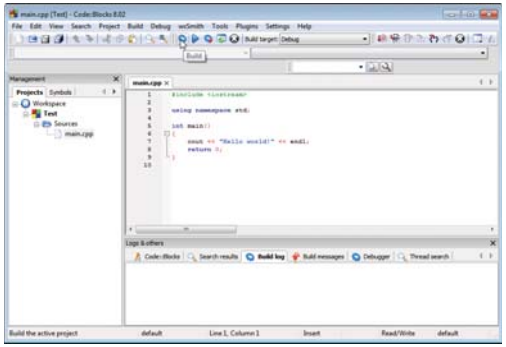


بعد از نوشتن برنامه ابتدا باید آن را Build کنید. منظور از Build، ترجمهٔ برنامه به زبان ماشین است. برای اجرای آن، لازم است ابتدا برنامه به زبان ماشین تبدیل شود. روش این دو کار در قسمت بعد آمده است.

روش ترجمه و اجرای برنامه

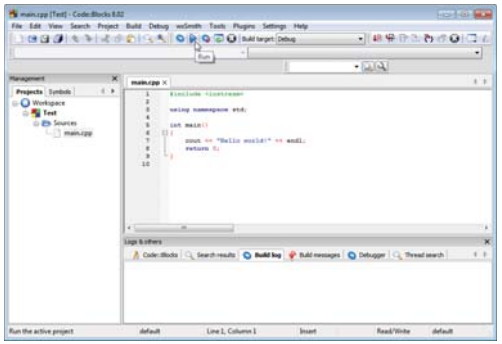
برای این کار می‌توانید از سه روش استفاده کنید:

۱. از منوی Build اولین گزینه را اجرا کنید.
۲. همچنین می‌توانید با استفاده از کلیدهای ترکیبی Ctrl+F9 این کار را انجام دهید.
۳. گزینه Build را در بالای صفحه کلیک کنید. آیکون این کار یک چرخ دندهٔ زرد رنگ است.



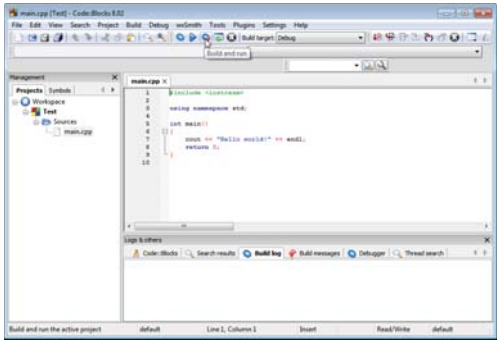
برای اجرای برنامه نیاز است که برنامه run شود که برای این کار نیز می‌توان به سه روش عمل کرد:

۱. از منوی Build گزینه run را انتخاب کنید.
۲. از کلیدهای ترکیبی Ctrl+F10 استفاده کنید.
۳. گزینه Run را در بالای صفحه اجرا کنید. آیکون اجرا مثلث سبز رنگ است.

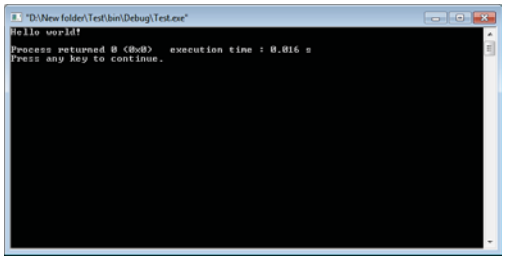


البته می‌توانید اعمال Build و Run را با هم اجرا کنید. که برای این کار نیز می‌توان از سه طریق عمل کرد:

۱. از منوی Build and run را انتخاب کنید.
۲. از کلید F9 استفاده کنید.
۳. گزینه Build and run را در بالای صفحه اجرا کنید.



با اجرای برنامه خروجی در صفحه‌ای مشکی برای شما نمایش داده خواهد شد.



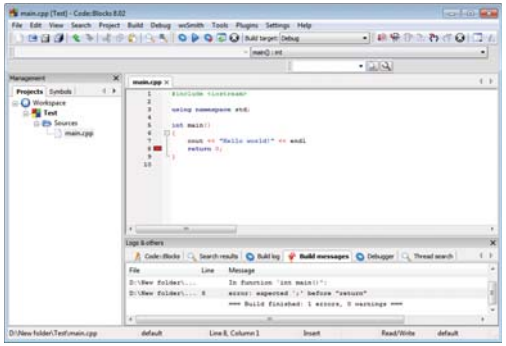
آ.۱.۴ خطاگیری- گرامری و منطقی

خطاهای برنامه را می‌توان به دو دسته تقسیم کرد:

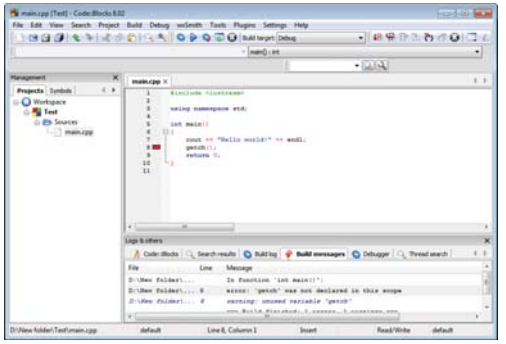
۱. خطای گرامری
۲. خطای منطقی

خطای گرامری

این دسته از خطاها به خطاهایی مربوط می‌شود که معمولاً کاربران مبتدی در نوشتن دستورات مرتکب می‌شوند و دستورات را به شکل صحیح نمی‌نویسند. بیشترین خطا در این زمینه فراموش کردن نوشتن ؛ (سمی کالن) است. می‌توانید نوع خطا و محل خطا را در پایین صفحه در قسمت Build messages مشاهده کنید. در محیط نرم‌افزار نیز یک مربع قرمز ابتدای خطی که در آن خطا وجود دارد، ظاهر می‌شود.



البته اگر خطا به ننوشتن ؛ مربوط باشد نرم‌افزار شماره خط خطا را خط بعد مشخص می‌کند.
دسته دیگر خطاهای این نوع، ننوشتن فایل‌های کتابخانه‌ای توابعی است که در برنامه استفاده شده است.



خطاهای منطقی

منظور از این نوع، خطاهایی است که در اجرای برنامه مشکلی ایجاد نمی‌کند و خروجی برنامه نیز ظاهر می‌شود اما جواب‌های درستی به دست نمی‌دهد.

تشخیص این خطا به مراتب سخت‌تر از خطای نوع اول است.

با محدود کردن اجرای برنامه و مشاهده مقادیر متغیرهای تعریف شده می‌توانید خطا را شناسایی کنید.

برای دیدن مقدار متغیرها از منوی Debug و زیرمنوی Debugging windows گزینه Watches را انتخاب کنید. در پنجره باز شده نام متغیرها را تایپ کنید و سپس با محدود کردن اجرا می‌توانید مقادیر این متغیرها را مشاهده کنید و به این ترتیب محل بروز خطا را تشخیص دهید.

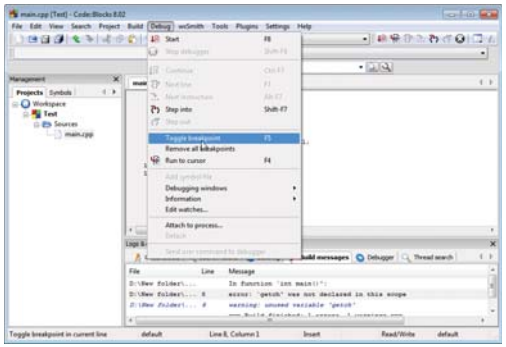
در قدم بعد باید محدوده اجرای برنامه را مشخص کرد. به این منظور از Breakpointها استفاده می‌شود. زمانی که برای دستوری Breakpoint مشخص شود برنامه با رسیدن به این دستور مکث می‌کند و وارد محیط دیباگ می‌شود. سپس می‌توان از دستورات Step و سایر موارد استفاده کرد.

برای تعیین Breakpoint یک دستور می‌توان از روش‌های زیر استفاده کرد:

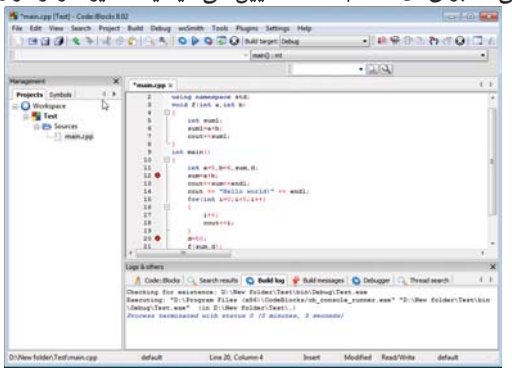
۱. بر روی خط دستور راست کلیک کرده و گزینه Toggle breakpoint را انتخاب کنید.

۲. بر روی خط دستور کلیک کرده و از منوی Debug گزینه Toggle breakpoint را انتخاب کنید.

۳. بر روی خط دستور کلیک کرده و کلید F5 را فشار دهید.



در ابتدای خطی که برای آن Breakpoint تعیین می‌کنید یک دایره قرمز قرار می‌گیرد.



بعد از تعیین Breakpointها برای اجرای برنامه می‌توانید از منوی Debug گزینه Start/Continue را انتخاب کنید یا از کلید F8 استفاده کنید. برنامه تا رسیدن به اولین Breakpoint اجرا می‌شود و صفحه خروجی نمایش داده می‌شود.

در ادامه موارد زیر به شما این امکان را می‌دهند که مراحل پیشروی دستورات را همان گونه که تمایل دارید انتخاب کنید:

۱. Next line: این گزینه برنامه را خط به خط اجرا می‌کند. می‌توانید از منوی Debug انتخاب کنید یا کلید F۷ را فشار دهید

۲. Step into: عملکرد این گزینه شبیه Next line هست با این تفاوت که در Next line زمانی که با دستور فراخوانی یک تابع روبرو شود، وارد آن تابع نمی‌شود و مکان نما به خط بعدی منتقل می‌شود. اما در Step into اجرای برنامه به آن تابع منتقل می‌شود و مکان نما ابتدای اولین دستور تابع قرار می‌گیرد و بعد از اجرای کامل آن تابع به تابع فراخوانی کننده بر می‌گردد. این گزینه را می‌توانید از منوی Debug انتخاب کنید یا از کلیدهای ترکیبی Shift+F7 استفاده کنید.

۳. Step out: این گزینه از جهت منطقی عملکردی بر عکس Step into دارد. اگر با اجرای Step into یا دلیل دیگری وارد یک تابع شده‌اید می‌توانید با اجرای Step out از آن خارج شده و مکان نما به تابع فراخوانی کننده منتقل می‌شود. این گزینه را می‌توانید از منوی Debug انتخاب کنید یا از کلیدهای ترکیبی Ctrl+F7 استفاده کنید.

برای حذف کردن Breakpoint می‌توان روی آن کلیک کرد. همچنن برای حذف همه Break-pointها می‌توان از منوی Debug گزینه Remove all breakpoints را انتخاب کرد.
گزینه دیگری غیر از Breakpoint نیز برای محدود کردن اجرا وجود دارد و آن Run to cursor هست.
ابتدا مکان نما را روی خطی که قصد دارید عمل اجرا تا آن مکان انجام شود قرار دهید. سپس از منوی Debug گزینه Run to cursor را انتخاب کنید یا کلید F4 را فشار دهید. در آغاز خط مشخص شده، مثلث زرد رنگی قرار می‌گیرد و برنامه بلافاصله تا این دستور اجرا شده و سپس مکث می‌کند.

آ.۲ کامپایلرهای C++ روی گوشی‌های هوشمند و تبلت

به منظور اجرای برنامه‌های C++ بر روی سیستم‌عامل اندروید، چندین نرم‌افزار وجود دارد که از جمله معروف‌ترین و قوی‌ترین آنها C4droid هست. برای استفاده از این کامپایلر مراحل زیر را انجام دهید:

۱. به این آدرس‌های زیر مراجعه کنید. هر سه فایل مربوط به گوشی‌های هوشمند را بارگذاری کنید.

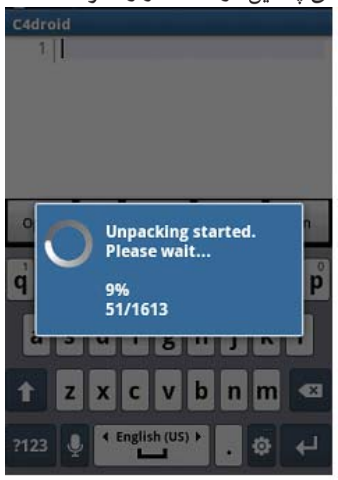
http://bayanbox.ir/download/7724964434733362302/C4droid-CCpp-compiler-IDE-4.97.apk
http://bayanbox.ir/download/2203927414682157 68/GCC-plugin-for-C4droid-C-IDE-4.9.1.apk

http://bayanbox.ir/download/6747071117453643935/SDL-plugin-forC4droid.apk

۲. نرم‌افزار C4droid – C/C++ compiler & IDE را نصب کنید.
۳. پلاگین‌های GCC و SDL را نیز برای اجرای کامل برنامه‌های C++ نصب کنید.
۴. آیکون برنامه به این شکل است که در قسمت برنامه‌های کاربردی قابل مشاهده است.

برنامه را اجرا کنید. در کادر باز شده گزینه اول Install to internal memory (نصب در حافظه داخلی) را انتخاب کنید. در صورتی که گوشی روت شده باشد می توانید گزینه Install to external memory (نصب در حافظه خارجی) را کلیک کنید.

منتظر بمانید تا فایل‌های پلاگین در حافظه وارد شوند.



۵. از منوی Preferences برنامه و در قسمت Select compiler گزینه G++ را انتخاب کنید.

آ.۱.۲ نوشتن و اجرای برنامه

محیط برنامه شامل یک صفحه سفید و کلیدهای کنترلی است.

برای نوشتن توابع کتابخانه‌ای برنامه دقت کنید که تابع iostream نیازی به پسوند h. ندارد.

بعد از اتمام دستورات و به منظور کامپایل برنامه دکمه Compile و جهت اجرای برنامه دکمه Run را انتخاب کنید.

خروجی این برنامه به شکل زیر است: