عنوان کتاب یا مطلب آموزشی : آموزش کامل خانواده C مانند: ... , C , C++ , C#
نویسنده : هادی محمدپور
تاریخ نشر : خرداد ۱۳۸۶ ه.ش
تایپ : هادی ممحدپور
توزیع:

http://www.hadi-vista.blogfa.com

به زودی در آدرس زیر فعال خواهیم شد:

http://www.hadi-vista.tk

.....

آموزش کامل C++

با سلام خدمت شما عزیزان.قراره که توی این سایت با ارسال مقالات کامپیوتری معلومات شما رو در مورد این پدیده نوین تکنولوژی افزایش بدیم.خوب پس بدون معطلی میرم سر اصل مطلب : اولین مقاله در مورد برنامه نویسیه

عرض کنم که: از زمانی که اولین کامپیوتر ها در آمریکا جهت تحقیقات دانشگاهی ومقاصد نظامی اختراع شدند,برای اینکه بتونن از قابلیت های این کامپیوتر در جهت رفع نیاز های خودشون استفاده کنند لازم بود تا به کامپیوتر دستور بدن که براشون کاری انجام بده.اما همونطور که می دونید کامپیوتر یه ماشین دیجیتالی هستش وبه غیر از ۰ و۱ زبون دیگه ای حالیش نیست,بنابراین متخصصان علوم کامپیوتر زبان های برنامه نویسی رو اختراع کردند.که این زبانها اون اوایل چندتا بیشتر نبودن ولی حالا برای خودشون دنیایی دارن.وبه انواع زبانهای سطح بالاوپایین تقسیم میشن..از زبانهایی که اون موقع خیلی رایج بود میشه فورتن و کوبال رو نامبرد. البته این زبانها امروزه دیگه منسوخ شده وکمتر از اونها استفاده میشه بعد از این زبانها کم کم پای زبانهای دیگه ای مثل بیسیک به میون اومد و همین زبان بیسیک بود که باعث شد آقای بیل گیتس وارد دنیای کامپیوتر بشه و شرکت عظیم مایکروسافت رو بنیانگذاری کنه.

اما زبان مورد علاقه من

زبان مورد علاقه من زبانC (سی)هستش این زبان در سال ۱۹۷۰ توسط دنیس ریچی روی یه ماشینDECPDP که از سیستم عامل یونیکس استفاده می کرد ابداع و پیاده سازی شد.علت نامگذاری این زبان به سی هم این بود که زبان های قبل از اونAوBنام داشتندخوب طبیعتا اسم این زبان رو هم سی گذاشتن این زبان به مرور زمان و توسط برنامه نویس های حرفه ای تکمیل شد به طوری که بعد از چند سال زبان سی به زبان محبوب دنیای برنامه نویسها تبدیل شد.اما تا چندین سال استاندارد عملی سی همون ویرایشی بود که توسط دنیس ریچی و برایان کرینگهان در کتاب **The C programing language**توضیح داده شده بود.عاقبت در دسامبر ۱۹۸۹با اتخاذ استاندارد **ANCI C** این زبان هم استاندارد وبه قول معروف رسمی شد.

C++:

C++ چرا ؟

خوب در طی سالیان متمادی برنامه های کامپیوتری جامع تر وکاملتر شدندوبه تبع اون سورس یا منبع برنامه ها هم پیچیده تر شدند به طوری که نوشتن و تفکیک این سورس ها حتی برای برنامه نویس های حرفه ای هم مشکل شده بود .در پاسخ به لزوم مدیریت این پیچیدگی ها زبانC++در سال ۱۹۸۰توسط شخصی به نام بجارنه استراستروپ,در آزمایشگاه بل در موری هیل نیوجرسی اختراع شد.اگه سی رو بهترین زبان برنامه نویسی دنیا بدونیمC++ قدرت اون رو دوچندان کرده.این زبان علاوه بر این که قابلیت های زبان سی رو در خودش نهفته داره از تکنولوژی(**OOP) Object Oriented Programming**)یا برنامه نویسی شی گرا بهره می گیره که باعث شده حجم برنامه های کم تر ونوشتن و مدیریت اونها آسون تر بشه.راجع به برنامه نویسی شی گرا در مقالات بعدی توضیحات کاملی خواهم داد.امازبان مورد بحث ما یعنیC++از زمان اختراع سه بار مورد ویرایش یا بهتر بگم باز نگری قرار گرفت تا استاندارد اون به وجود بیاداما خوب این زبان در حال توسعه و پیشرفته وهنوز هم قابلیت های این زبان در حال افزایشه و تا ایجاد یه استاندارد تکمیل ممکنه چند سال طول بکشه.خوب فکر می کنم برای بار اول کافی باشه

آموزش **C++**

سلام دوستان این هم درس دوم آموزش برنامه نویسی.خوب تو این درس شما رو با اصول اولیه برنامه نویسی آشنا می کنم.برای نوشتن هر برنامه ای یه سری قواعد هستش که رعایت اونها موجب تسریع در امر برنامه

نویسی میشه.اول این که بفهمین از برنامه تون چی می خواهید دوم این که تمام کارهایی رو که برنامه تون باید انجام بده بنویسین و ورودی ها وخروجی ها رو تعریف کنید وبرای تمام سوالات یه جواب در نظر بگیرین در واقع الگوریتم برنامه رو طراحی کنین که خوب برای طراحی الگوریتم اگه به طور خلاصه بخوام بگم باید یه نقطه شروع داشته باشین یعنی که مساله از اینجا آغاز میشه بعدش بسته به نوع برنامه دستور العملهای متفاوتی باید بکار گرفته بشن این دستور العملها فرمانهای معینی هستن که باید به ترتیب معینی اجرا بشن و در نهایت منجر به حل مساله بشن و در نهایت هم قسمت پایان.این جملات اجرایی یا همون دستور العملها رو معمولا به سه صورت متفاوت بیان میکنند اول به صورت جملات معمولی ومحاوره ای دوم به صورت گزاره ها و روابط ریاضی و سوم با استفاده از اشکال هندسی .که تو مقالات بعدی توضیحات جامعتری رو راجع به الگوریتم بهتون میدهم.

تا اینجا فقط به صورت تئوریک راجع به برنامه نویسی بهتون گفتم حالا بریم سراغ یه مثال عملی برای اینکار بهتره که اصول زیر رو به خاطر بسپارید

چرخه برنامه نویسی یه سری مراحل داره که شما باید در طول برنامه نویسی رعایت کنین.۱-در یک ادیتور کد برنامه رو بنویسین و در یک فایل ذخیره کنین۲ -در این مرحله فایل رو کامپایل کنین خروجی این مرحله یک فایل شی یا
object file
است۳-حال باید برنامه کامپایل شده رو لینک کنین تا یک فایل اجرایی یا
executable file. ایجاد شود
۴-اجرای برنامه و تست آن خب این هم اولین برنامه شما به زبان سی

```
#include
int main()
{
printf("Hello, World!n");
return 0;
}
```

این اولین برنامه به زبان سی است که همه در شروع این برنامه رو مینویسن.اگه این برنامه رو کامپایل و بعد لینک و بعد اجرا کنین. پیام

Hello, World!

رو صفحه به نمایش در میاد. اگه مثل من از کامپایلر برلند استفاده میکنین با زدن کلید

Alt-F9

برنامه کامپایل میشه.بعد ش باید برنامه رو لینک کنین برای این کار تو کامپایلر برلند کلید

F9

رو بزنین

تا فایل اجرایی ایجاد بشه بعد برنامه رو ذخیره کنین

حالا به محل ذخیره برنامه برین و اون رو اجرا کنین.

اما توضیح برنامه:در خط اول علامت

#include

به معنی ضمیمه کردن یک فایل است .وعلامت

اسم فایلی به همین نام است که در دایرکتوری کامپایلر قرار دارد و در آن تابع

printf()

تعریف شده.اما در تمام برنامه های سی تابع

main()

جز جدا نشدنی آن برنامه است که اکثرا فراخوانی توابع دیگر در این تابع صورت میگیرد.و واژه ی

return 0;

جزی از تابع

main()

است.اما در فایلی که به برنامه ضمیمه شده است یک سری توابع از پیش تعریف شده وجود داره که تابع

printf()

این تابع متن مورد نظر شما رو روی صفحهء مانیتور نمایش میده .

متغیر ها مکانهایی از حافظه رم کامپیوتر هستند که نوع خاصی از داده رو در خودشون نگه میداره حالا این داده میتونه هر چیزی باشه مثلا یه کارکتر یا یه عدد صحیح و یا یه عدد اعشاری ویا هر چیز دیگه ای در زبان سی ، شما میتونین واسه خودتون هم متغیر هایی رو تعریف کنین که اصطلاحا به اونها شیء گفته میشه.که راجع به اون به موقع براتون توضیح میدم.اما متغیر های استانداردی که ما با هاشون سر و کار داریم هر کدوم

با توجه به نوعشون مقداری از حافظه کامیوتر رو میگیرن مثلا توی زبان سی در کامیوترهای استاندارد آی بی ام متغیر کاراکتری یک بایت فضا اشغال میکنه که همین متغیر در سیستم های دیگه ممکنه مقدار فضای بیشتری رو اشغال کنه .

اما برای اینکه بفهمین چطور باید از مقدار فضایی که یه متغیر تو حافظه کامیوتر میگیره آگاه بشین باید از یک تابع کتابخانه ای استفاده کنین که تعریف اون به صورت زیر هستش

int sizeof(type of variable);

برای استفاده از این تابع باید نام اون رو بنویسین و درون پرانتز هم نوع متغیر مورد نظر خودتون رو بنویسید.مقداری رو که این تابع برمیگردونه همون مقدار فضایی هست که متغیر تو حافظه کامپیوتر میگیره.اگه دقیقا نفهمیدین منظور من چی بود برنامه ای رو که اخر این درس نوشتم بنویسید واجرا کنید.در اینجا من برا تون انواع متغیر ها نوشتم

1-char	-127 to 128
2-int	-32768 to 32768
3-long	-2,147,488,648 to 2,147,483
5-unsignned char	0 to 255
6-unsigned int	0 to 65535
7-unsigned long	0 to 4,294,967,295
8-float	-1.2e38 to 3.4e38
9-double	-2.2e308 to 1.8e308

خوب اینهم انواع متغیر ها در زبان سی و سی پلاس پلاس–شماره یک کاراکتر هستش و شماره های دو و سه عدد صحیح هستند منتها شماره سه میتونه اعداد صحیح بلند تری رو در خودش نگه داره .متغیر های پنج هم کاراکترهای بزگتر از صفر رو نگه میداره و شش و هفت هم اعداد صحیح بدون علامت رو در خودشون نگه میدارن متغیر های هشت و نه هم اعداد اعشاری رو در خودشون نگه میدارن ولی همونطور که می بینید نوع متغیر نه اعداد اعشاری بزرگتری رو می تونه در خودش نگه داره .اما برای تعریف یه متغیر ابتدا باید نوع اون رو بنویسید بعد اسمی رو که برای اون متغیر در نظر گرفتید بنویسید.مثلا

ch1 ;char

1 number ;int

اما برای مقدار دهی به متغیر باید ابتدا اسم متغیر رو بنویسید بعد علامت مقدار دهی یعنی = قرار بدین بعـدش مقدار مورد نظر رو به اون بدید..این برنامه ای که من نوشتم احتمالا با متغیر ها بیشتر اشنا میشین

```
#include <iostream.h>
#include <stdlib.h>

main(){
char ch1='a';
int num1=10;
long num2=10000;
float f1=2.23;
double f2=1.9344334;

;"n">>f2>>"n">>f1>>"n">>num2>>"n">>num1>>"n">>ch1>>cout

cout<<sizeof(ch1)<<"n"<<sizeof(num1)<<"n"<<sizeof(num2)<<
"n"<<sizeof(f1)<<"n"<<sizeof(f2)<<"n
```

درس چهارم در مورد یکی از قابلیت های مهم و منحصر به فرد زبان سی یعنی در مورد اشاره گر هاست اگه یادتون باشه تو درس قبل در مورد متغیر ها براتون گفتم که متغیر مکانی در حافظه است که نوع خاصی از داده رو در خودش نگه میداره .اما اشاره گر چیه؟؟اشاره گر ها به زبان ساده مکانی از حافظه کامیوتر هستند که آدرس متغییری رو در خودشون نگه میدارن یعنی آدرس حافظه یه متغیر رو در خودشون نگه میدارن و شما از طریق اشاره گرها می تونین به این متغِر دسترسی داشته باشید.اما ممکنه بگین این کار به چه دردی میخوره برای جواب این سوال باید یه کم اطلاعات شما در مورد برنامه نویسی زیاد تر بشه ولی از هر کسی که به زبان سی برنامه مینویسه بپرسید بهتون میگه که این قابلیت بزرگ برنامه نویسی که فقط توی زبان سی وجود داره کمک زیادی به شما تو نوشتن برنامه هاتون میکنه پس حتما یاد بگیرید.برای تعریف یه اشاره گر اول باید نوع متغیری رو که اشاره گر شما بهش اشاره میکنه رو تعریف کنین .بعد یه *بنویسین بعد هم اسم دلخواهتون رو برای اشاره گر بنویسین بعد هم کاراکتر سمی کالن.خوب تا اینجا اشاره گر رو تعریف کردید اما این اشاره گر هنوز نمی دونه به کجا باید اشاره کنه بنا بر این شما بایِد به اشاره گر بفهمونین که به کجا اشاره کنه

```
int number1;
int *pointer ;
pointer = & number1;
```

خوب در خط اول متغیر رو تعریف کردیم در خط دوم اشاره گر رو تعریف کردیم و در خط سوم اشاره گر رو به متغیر ربط دادیم.یادتون نره حتما خط سوم رو بنویسین.اما علامت & قبل از اسم متغیر محلی از حافظه رو که متغیر توی اون نوشته شده رو نشون میده.خوب حالا شما می تونین از اشاره گر بجای متغیر استفاده کنید مثلا دو دستور زیر هر دو یه کار رو برای شما انجام میدن

```
number1 = 10;
*pointer = 10;
```

اما یادتون باشه هرگز هیچ اشاره گری رو الکی تعریف نکنین این کار عوارض سویی برای برنامه داره.چون در اون صورت ممکنه اشاره گر به جایی از حافظه اشاره کنه که فایلهای حیاتی برنامه شما و یا حتی سیستم عامل شما در اونجا ذخیره شده.پس مواظب باشید.

قبل از بررسی تفاوت بین کلاسها بین C++ و C# ابتدا خاصیت های عمومی یک کلاس را بررسی کرده سپس به بررسی تفاوت بین این دو زبان قدرتمند خواهیم پرداخت.

تعریف. کلاس در حالت ساده مجموعه ای از خواص(attributes) و روش ها (methodes) است که در رابطه با هم هدف مشترکی را دنبال می کنند و خدماتی را ارائه می کنند. البته این تعریف در حد یک تعریف علمی می باشد ولی ذکر این نکته ضروری است که اکثر تعاریف در شی گرایی در طبیعت وجود دارد و در ساده ترین حالت برای تعریف یک کلاس می توان از تعریف انسان استفاده کرد که این بحث در این مقاله نمی گنجد. در هر حال ما به تعریف فوق اکتفا کرده و بحث را ادامه می دهیم.

خواص ها و روش ها که در تعریف کلاس ارائه شد می تواند نحوه دسترسی متفاوتی مانند public و private و protected داشته باشد که برای برنامه نویسان C++ نیازی به توضیح نیست.

تفاوت در تعریف یک کلاس:

چنانچه در مقاله لایه ها در #C نیز اشاره شد یک کلاس در یک برنامه #C در داخل یک لایه تعریف خواهد شد(اجباری نیست). تعریف یک کلاس در #C تفاوتهایی با ++C دارد که سعی خواهیم کرد این تفاوت ها را با چند مثال توضیح دهیم.

فرض کنید کلاس زیر در ++C تعریف شده است که این کلاس را به یک کلاس #C تبدیل خواهیم کرد.

```
// Test.h
class CTest{
    private:
        int   m_at1;
        int   f1();
    public:
        float m_at2;
        int   f2();
            CTest();
            ~CTest();
};

// Test.cpp
#include "test.h"
CTest::CTest(){
    m_at1 = 0;
    m_at2 = 0.0;
```

```
}
CTest::~CTest(){
    // cout << "destructor call";
}
int CTest::f1(){
        return m_at1;
}
int CTest::f2(){
    return m_at1+(int)m_at2;
}
```

تفاوت اول:

بر خلاف C++ در C# تعریف و نحوه عملکرد یک تابع یا Method در خود تعریف کلاس قرار داده می شود. در C++ غالبا تعریف کلاس در فایل h. و بدنه توابع در فایل cpp. قرار می گرفت. البته ذکر این نکته ضروری است که می توان همین عمل را در h. نیز قرار داد ولی در یک برنامه بزرگ این عمل کنترل برنامه را از دست برنامه نویس خارج کرده و همچنین مشکلات دیگری را برای برنامه نویسان ایجاد می کند که برنامه نویسان C++ با این مشکلات آشنا هستند.

تفاوت دوم:

بر خلاف C++ در C# یک دسته از متغییرها را نمی توان بصورت public یا private و ... تعریف کرد و برای هر متغییر باید نحوه دسترسی به آن نیز مشخص شود.

تفاوت سوم:

بر خلاف C++ در C# تابع destructor وجود ندارد و خود C# مسئول از بین بردن یک شی می باشد که این از بین بردن با توجه به محدوده تعریف این شی انجام می شود. البته تعریف destructor وجود دارد ولی نمی توان destructor را فراخوانی کرد . در C++ فراخوانی Destructor با استفاده از عملگر delete قابل انجام بود ولی چون C# مدیریت object ها را خود بعهده می گیرد شی ایجاد شده نمیتواند توسط برنامه خراب شود. این مطلب در مقالات بعدی مورد بررسی قرار خواهد گرفت.

با توجه به تفاوتهای ذکر شده می توان این کلاس را به صورت زیر برای یک کلاس C# ارائه کرد.

```
// Test.cs

using System;
namespace NS

{

    publicclass CTest

    {

        privateint   m_at1;

        public  float m_at2;

        privateint   f1()

        {

                return m_at1;

        }

        public  int   f2()

        {

                return m_at1+(int)m_at2;

        }

        public CTest()
```

```
        {
            m_at1 = 0;
            m_at2 = 0.0f;
        }
        ~Test()
        {
            // Console.WriteLine("destructor call");
        }
    }
}
```

ایجاد یک کلاس در ++ و #

در موقع استفاده از یک کلاس اگر از default constructor برای ایجاد یک کلاس استفاده می شود در #C باید constructor بصورت void فراخوانی شود (مانند توابع void معمولی)

به مثال زیر دقت کنید.

```
// C++ code
CTest        *a;
a = new CTest;

//      C# Code
CTest        a;
a = new CTest();
```

ذکر این نکته ضروری است که در #C وقتی کلاسی تعریف می شود در واقع اشاره گر به آن کلاس معرفی می شود بنابراین با تعریف a CTest کلاسی از CTest ایجاد نمی شود و فقط یک اشاره گر از CTest تعریف می شود و برای ایجاد یک کلاس استفاده از عملگر new اجباری است.

ولی در C++ تعریف a CTest به منزله ایجاد یک کلاس و فراخوانی Default Constructor می باشد.

با توجه به این که برنامه های تحت داس رابط ضعیفی نسبت به ویندوز و لینوکس و... دارند ما حداقل باید برنامه هایمان را به ماوس مجهز کنیم تا این نقص تا حدودی برطرف شود. مخصوصا اگر بخواهیم یک برنامه گرافیکی بنویسیم برنامه مان بدون ماوس غیر قابل استفاده خواهد بود .

خب بهتره بریم سر اصل مطلب. در این مقاله فعال کردن ماوس در زبان C توضیح داده شده. البته ما ماوس را با استفاده از وقفه ها که در تمام کامپیوترهای IBM یکی هستند فعال می کنیم و کدهای ارائه شده با کمی تغییر در پاسکال و بیسیک نیز کار خواهند کرد. اگر شما با زبان C آشنایی ندارید ممکن است این کدها برایتان ناآشنا باشند. پس بهتر است از خواندن این مقاله صرفنظر کنید. البته ممکن است به زودی آموزش C و C++ را نیز در این سایت بگذاریم. ابتدا برای این که از ماوس استفاده کنیم باید بفهمیم که درایور ماوس نصب شده یا نه؟ اگر نصب نشده باشد نمی توان ماوس را فعال کرد. برای این کار از تابع شماره hex۰۰ از وقفه hex۳۳ استفاده می کنیم. اگر قبلا با وقفه ها کار کرده باشید حتما می دانید که شماره تابع وقفه در ثبات AH قرار می گیرد ولی توابع hex۳۳ به جای AH از AX استفاده می کنند. یعنی برای فراخوانی این وقفه شماره تابع آن را در ثبات AX قرار می دهیم. پس از فراخوانی این وقفه اگر مقدار ثبات AX برابر hex۰۰۰۰ باشد یعنی درایور ماوس نصب نشده و ما نمیتوانیم از ماوس استفاده کنیم. پس از این کار باید نشانگر ماوس را د ر صفحه ظاهر کنیم. نشانگر ماوس دقیقا در وسط صفحه ظاهر خواهد شد. اگر در حالت متنی باشیم این نشانگر به صورت یک مستطیل و اگر در حالت گرافیکی باشیم نشانگر به صورت یک فلش کوچک نشان داده خواهد شد. برای ظاهر کردن کافی است از تابع شماره ۰۱ وقفه h۳۳ استفاده کنیم. برنامه زیر هم درایور ماوس را تشخیص داده و هم آن را در حالت متنی نشان میدهد:

```
#include <STDIO.H>
#include <CONIO.H>
#include <STDLIB.H>
#include <DOS.H>
void main()
{ union REGS r;
clrscr();
r.x.ax= 0;
int86(0x33,&r,&r);
if(r.x.ax==0)
{
printf("No Mouse Available.....");
getch();
exit(1);
}
r.x.ax=1;//place 01 in AX register
int86(0x33,&r,&r);//showing the mouse pointer
printf("press any key to exit…");
getch();
}
```

تابع مفید دیگری برای کار با ماوس تابع ۰۲ از وقفه hex۳۳ میباشد. این تابع نشانگر ماوس را پنهان می کند. فرض کنید می خواهید مانند برنامه Paint وقتی ماوس را میکشید یک خط هم با آن کشیده شود. در این حالت اگر نشانگر ماوس دیده شود بعضی از نقاط خط پاک خواهند شد و بهتر است که در این مواقع نشانگر را پنهان کنیم. بعدا مثال کاملتری برایتان خواهم نوشت. برای کار با ماوس ما باید بدانیم که نشانگر اکنون د رکجای صفحه می باشد یعنی مختصات (x,y) آن را بدست آوریم. برای این کار از تابع شماره ۰۳ استفاده می کنیم. که پس از فراخوانی تابع ثبات CX حاوی مختصات افقی(x) و DX حاوی مختصات عمودی نشانگر خواهد بود. کار مهم دیگر تشخیص کلیدهای فشرده شده ماوس است. تابع ۰۳ همچنین تعیین می کند که کدام کلید از ماوس فشرده شده است. این کلیدها را ثبات BX تعیین میکند. فقط همین کافی است که بدانید پس از فراخوانی وقفه اگر BX برابر با ۰۰۰۰۰۰۰۱)Hex (باشد کلید چپ ماوس فشار داده شده است. راجع به فشار دادن کلیدهای دیگر اگر علاقه مند بودید به من ایمیل بزنید تا نحوه تشخیص آنها را هم بنویسم. مثال زیر نحوه کار را مشخص می کند:< /P><P/ >

```
#include <STDIO.H>
#include <CONIO.H>
#include <STDLIB.H>
#include <DOS.H>
void main() {
union REGS i,o;
clrscr();
i.x.ax=0;
int86(0x33,&i,&o);
if(o.x.ax==0) {
printf("No Mouse Available...");
exit(1);
}
i.x.ax=1;
int86(0x33,&i,&o);
gotoxy(25,23);
printf("Press any key to exit...");
while(!kbhit())
{
i.x.ax=3;
int86(0x33,&i,&o);
gotoxy(2,2);
printf("x->co-ordinate=(%d) \n y->co-ordinate=(%d)
",o.x.cx,o.x.dx);
if(o.x.bx==0x01) printf("Right button of mouse pressed."); }
i.x.ax=2; int86(0x33,&i,&o);
}
```

برای پایان کار می خواهیم یک برنامه که ابزار pencil برنامه های گرافیکی را شبیه سازی می کند بنویسیم. برای این کار ما ابتداصفحه را درحالت گرافیکی قرار می دهیم برای این کهبرنامه زیر کار کند در دستور initgraph(&gd,&gm,""); در داخل کوتیشن مسیرفایلهای bgi. را بنویسید.

```
#include <CONIO.H>
#include <STDIO.H>
#include <STDLIB.H>
```

```
#include <GRAPHICS.H>
#include <DOS.H>
union REGS i,o;
main() {
int show_mouse();
int hide_mouse();
int get_mouse_pos(int *,int *,int *);
int gd=DETECT,gm,button,x1,y1,x2,y2;
initgraph(&gd,&gm,"");
i.x.ax=0; int86(0x33,&i,&o);
if(o.x.ax==0)
{ printf("No Mouse is available..");
exit(1);
restorecrtmode();
}
outtextxy(230,400,"Press any key to exit....");
while(!kbhit())
{
show_mouse(); get_mouse_pos(&x1,&y1,&button);
x2=x1;
y2=y1;
while(button==1) {
hide_mouse();
line(x1,y1,x2,y2);
x1=x2;
y1=y2;
get_mouse_pos(&x2,&y2,&button); }
}
restorecrtmode();
} show_mouse()
{
i.x.ax=1; int86(0x33,&i,&o);
}
hide_mouse()
{
```

```
i.x.ax=2; int86(0x33,&i,&o);
}
get_mouse_pos(int *x,int *y,int *button)
{
i.x.ax=3;
int86(0x33,&i,&o);
*x=o.x.cx;
*y=o.x.dx; *button=o.x.bx&1;
}
```

- **دستورات ورودی و خروجی:**

دستورات استاندارد ورودی و خروجی، که در واقع به صورت تابع در کامپایلر C تعریف شده اند در فایلهایی که پسوند آنها h است، هستند و ما برای استفاده از آنها باید آنها را در بالای برنامه ذکر کنیم. برای این کار از دستور #include<FILE name> استفاده می کنیم.

فایلهایی که معمولا در ابتدای هر برنامه باید آنها را ذکر کنیم عبارتند از:

conio.h
stdio.h
stdlib.h

البته این فایلها ممکن است در تمام برنامه ها احتیاج نباشند ولی فعلا برای این که کارمان پیش برود نام فایلهاراذکر کنید(ضرری نداره). پس در ابتدای هر برنامه سه دستور زیر را حتما تایپ کنید بعدا در بخش نوشتن هدر فایل(header file) مفصلا در مورد این نوع فایلها بحث خواهیم کرد .

```
#include<CONIO.H>
#include<STDIO.H>
```

```
#include<STDLIB.H>
```

برای این که بدانید تابعی را که به کار می برید در کدام هدر فایل است مکان نما را زیر آن تابع برده و کلید Ctrl+F1 را فشار دهید. این کار را باید در محیط ادیتور کامپایلر C انجام دهید. میتوان دستورات ورودی و خروجی را به دو نوع تقسیم کرد:

الف)کاراکتری. ب)با فرمت مشخص .

الف) دستورات ورودی و خروجی کاراکتری:

۱. **دستور ()putch:** شکل کلی این دستور به صورت زیر میباشد و برای نشان دادن یک کاراکتر در صفحه نمایش به کار میرود. putch(نام متغیر کاراکتری یا خود کاراکتر); باید توجه کرد که اگر مستقیما کاراکتر را بخواهیم بنویسیم باید آن را در داخل کاما(‘ ’)قرار دهیم. ولی برای متغیر رشته ای این کار نیاز نیست.

```
putch('a');
putch(chvar);
```

دستور اول a را در صفحه نمایش چاپ میکند. و دستور دوم مقداری را که در متغیر کاراکتری chvar قرار دارد چاپ می کند. این متغیر به صورت ; char chvar تعریف شده است. یک دستور دیگر به نام ()putchar وجود دارد که در نسخه های قدیمی تر C به کار میرفت. ولی امروزه با این که باز هم وجود دارد ولی به کار نمیرود. شما اگر دوست داشتید از این دستور استفاده کنید. عملکردش مانند ()putch میباشد.

۲. **دستو () getch :** از این دستور برای گرفتن یک کاراکتر از صفحه کلید استفاده می کنند. شکل کلی دستور بصورت زیر است:

```
نام متغیر کاراکتری=getch();
```

اگر این دستور را بدون استفاده از متغیر به کار بریم برنامه منتظر فشار دادن کلیدی از طرف کاربر می ماند. کاراکتری که توسط این دستور دریافت میشود در صفحه نمایش نشان داده نمیشود.

۳. **دستور getche: ()** این دستور مانند ()getch میباشد با این تفاوت که دستور ()getche کاراکتر گرفته شده را در صفحه نیز نشان میدهد .

۴. **دستور ()getchar:** این دستور نیز برای گرفتن کاراکتر از کاربر است. و شکل کلی این دستور نیز دقیقا مانند دو دستور قبل میباشد. با این تفاوت که این دستور پس از گرفتن کاراکتر منتظر کلید Enter می ماند در حالی که دو دستور قبل اینطور نبودند. برای مثال میخواهیم برنامه ای را بنویسیم که سه کاراکتر را از کاربر گرفته و آنها را بصورت معکوس نشان دهد:

```
#include<CONIO.H>
#include<STDIO.H>
#include<STDLIB.H>
void main()
{
char ch1,ch2,ch3;
clrscr();
getche();
getche();
getchar();
putch(ch3);
putch(ch2);
putch(ch1);
getch();
}
```

اگر ورودی این برنامه را بصورت abc وارد کنیم برنامه cba را به عنوان خروجی برمیگرداند.

ب)دستورات ورودی و خروجی با فرمت مشخص:

۱. دستور ()printf : شکل کلی دستور بصورت زیر است:

printf("پیغام و کاراکترهای کنترلی") ;

printf("پیغام و کاراکترهای کنترلی و کارامترهای فرمت",نام متغیرها یا مقادیر داده ها);

فرم اول دستور بیشتر برای چاپ پیغام به کار میرود. برای مثال دستور زیر پیغام Hello world! را نشان میدهد.

```
printf("Hello world!");
```

فرم دوم این دستور مواقعی به کار می رود که میخواهیم مقادیر متغیرها را نیز همراه پیغام چاپ کنیم. منظور از کاراکترهای کنترلی و کاراکترهای فرمت را در جدول زیر نشان داده ام. توجه کنید که این کاراکترها را باید در داخل کوتیشن به کار برید.

کاراکتر فرمت printf()	مفهوم
%c	Character
%d	Decimal
%f	Float
%ld	Long decimal
%u	Unsigned
%s	String
%p	Pointer
%%	کاراکتر %

کاراکتر کنترلی printf()	مفهوم
\n	New line
\r	Carriage-return
\t	Tab
\b	Back space
\"	نمایش کاراکتر "
\'	نمایش کاراکتر '
\\	نمایش کاراکتر \

کاراکترهای فرمت را موقعی به کار میبریم که میخواهیم یک متغیر از نوع خاصی را در خروجی نشان دهیم. بری این کار ابتدا نوع کاراکتر را در داخل کوتیشن با استفاده از کاراکترهای فرمت مشخص میکنیم و سپس بیرون از کوتیشن نام متغیر را ذکر میکنیم. به مثال زیر توجه کنید:

```
printf("your average is:%f", ave);
```

این دستور ابتدا پیغام your average is: را نشان میدهد و سپس به یک کاراکتر فرمت که یک متغیر از نوع اعشاری را بیان میکند برخورد میکند و در خارج از کوتیشن متغیر متناظر با f٪ را میابد که در این مثال متغیر ave می باشد. ave بصورت float ave; تعریف شده است. اگر یک دستور printf() دیگر بعد از این دستور قرار دهیم پیغام بلافاصله بعد از آن چاپ خواهد شد اگر بخواهیم پیغام در سر خط بعد ظاهر شود از کاراکتر کنترلی \n استفاده میکنیم. برای این کار می توانیم به دو روش عمل کنیم یا در آخر دستور قبل کاراکتر کنترلی را قرار دهیم یعنی دستور بالا بصورت زیر باشد:

```
printf("your average is:%f \n", ave);
```

یا اینکه \n را در ابتدای دستور بعد قرار دهیم.

پس از نوشتن یک الگوریتم باید آن را با استفاده از یک زبان برنامه نویسی تبدیل به یک برنامه قابل اجرا برای کامپیوتر نماییم. این زبانها به سه دسته کلی تقسیم میگردند :

۱- زبان ماشین (سطح پایین) : این زبان مستقیما با صفر و یک نوشته می شود و بدون هیچ واسطه ای برروی کامپیوتر قابل اجرا است. طراحان سخت افزار هر کامپیوتر، زبان ماشین خاص خود را برای آن ماشین طراحی می نمایند. به همین دلیل هر برنامه ای که به زبان ماشین نوشته شود، فقط برروی همان ماشین خاص کار می کند، بهمین دلیل برنامه های نوشته شده به زبان ماشین را غیر قابل حمل می نامند. از طرف دیگر یادگیری این زبان بسیار مشکل بوده و برنامه نویسی با آن نیز بسیار سخت است و همچنین احتمال بروز خطا نیز در آن زیاد است.

۲- زبان اسمبلی : این زبان شکل ساده تر زبان ماشین است، بدین صورت که برای هر دستورالعمل زبان ماشین، یک اسم نمادین انتخاب شده است (مانند دستور ADD بجای کد دودویی دستورالعمل جمع) که بخاطر سپردن و برنامه نویسی با آنها برای انسانها ساده تر است. اما این برنامه ها برای ماشین قابل فهم نیست و باید قبل از اجرا شدن توسط برنامه مترجمی بنام اسمبلر به زبان ماشین تبدیل شود. توجه کنید که از آنجا که

هر دستور زبان اسمبلی معادل یک دستور زبان ماشین است، این زبان نیز وابسته به ماشین می باشد و برنامه های نوشته شده به این زبان فقط برروی همان کامپیوتری که برای آن نوشته شده اند قابل اجرا است. علاوه براین کار با این زبانها هنوز هم نسبتا مشکل بود و فقط متخصصین کامپیوتر قادر به استفاده از آنها بودند.

۳- زبانهای سطح بالا : دستورالعملهای این زبانها بسیار نزدیک به زبان انسانها (بطور مشخص زبان انگلیسی) می باشد و بهمین دلیل برنامه نویسی به آنها بسیار ساده تر بوده و می توان الگوریتمها را به راحتی به این زبانها تبدیل کرد. از آنجا که این زبانها به هیچ ماشین خاصی وابسته نیستند، برنامه های نوشته شده با این زبانها (تا حد زیادی) قابل حمل می باشند. مثالهایی از این زبانها عبارتند از :

- بیسیک (Basic): برای کاربردهای آموزشی
- فرترن (Fortran) : برای کاربردهای علمی و مهندسی
- پاسکال (Pascal) : برای کاربردهای آموزشی و علمی

و بالاخره زبان برنامه نویسی C که درمورد آن بیشتر صحبت خواهیم کرد. البته برنامه های نوشته شده به این زبانها ابتدا باید به زبان ماشین ترجمه شوند تا برروی کامپیوتر قابل اجرا باشند. برای ترجمه این زبانها از کامپایلرها و یا مفسرها (به فصل ۱ مراجعه کنید) استفاده می شود.

تاریخچه C

برای بررسی تاریخچه زبان C باید به سال ۱۹۶۷ بازگردیم که مارتین ریچاردز زبان BCPL را برای نوشتن نرم افزارهای سیستم عامل و کامپایلر در دانشگاه کمبریج ابداع کرد. سپس در سال ۱۹۷۰ کن تامپسون زبان B را بر مبنای ویژگیهای زبان BCPL نوشت و از آن برای ایجاد اولین نسخه های سیستم عامل Unix در آزمایشگاههای بل استفاده کرد. زبان C در سال ۱۹۷۲ توسط دنیس ریچی از روی زبان B و BCPL در آزمایشگاه بل ساخته شد و ویژگیهای جدیدی همچون نظارت بر نوع داده ها نیز به آن اضافه شد. ریچی از این زبان برای ایجاد سیستم عامل Unix استفاده کرد اما بعدها اکثر سیستم عاملهای دیگر نیز با همین زبان نوشته شدند. این زبان با سرعت بسیاری گسترش یافت و چاپ کتاب " The C Programming Language" در سال ۱۹۷۸ توسط کرنیگان و ریچی باعث رشد روزافزون این زبان در جهان شد.

متاسفانه استفاده گسترده این زبان در انواع کامپیوترها و سخت افزارهای مختلف باعث شد که نسخه های مختلفی از این زبان بوجود آید که با یکدیگر ناسازگار بودند. در سال ۱۹۸۳ انستیتوی ملی استاندارد آمریکا

(ANSI) کمیته ای موسوم به X3J11 را را مامور کرد تا یک تعریف فاقد ابهام و مستقل از ماشین را از این زبان تدوین نماید.در سال ۱۹۸۹ این استاندارد تحت عنوان ANSI C به تصویب رسید و سپس در سال ۱۹۹۰، سازمان استانداردهای بین المللی (ISO) نیز این استاندارد را پذیرفت و مستندات مشترک آنها تحت عنوان ANSI/ISO C منتشر گردید.

در سالهای بعد و با ظهور روشهای برنامه نویسی شی گرا نسخه جدیدی از زبان C بنام C++ توسط بیارنه استراوستروپ در اوایل ۱۹۸۰ در آزمایشگاه بل توسعه یافت. در C++ علاوه بر امکانات جدیدی که به زبان C اضافه شده است، خاصیت شی گرایی را نیز به آن اضافه کرده است.

با گسترش شبکه و اینترنت، نیاز به زبانی احساس شد که برنامه های آن بتوانند برروی هر ماشین و هر سیستم عامل دلخواهی اجرا گردد. شرکت سان مایکروسیستمز در سال ۱۹۹۵ میلادی زبان Java را برمبنای C و C++ ایجاد کرد که هم اکنون از آن در سطح وسیعی استفاده می شود و برنامه های نوشته شده به آن برروی هر کامپیوتری که از Java پشتیبانی کند(تقریبا تمام سیستمهای شناخته شده) قابل اجرا می باشد. شرکت مایکروسافت در رقابت با شرکت سان، در سال ۲۰۰۲ زبان جدیدی بنام C# (سی شارپ) را ارائه داد که رقیبی برای Java بشمار می رود.

برنامه نویسی ساخت یافته

در دهه ۱۹۶۰ میلادی توسعه نرم افزار دچار مشکلات عدیده ای شد. در آن زمان سبک خاصی برای برنامه نویسی وجود نداشت و برنامه ها بدون هیچگونه ساختار خاصی نوشته می شدند. وجود دستور پرش (goto) نیز مشکلات بسیاری را برای فهم و درک برنامه توسط افراد دیگر ایجاد می کرد، چرا که جریان اجرای برنامه مرتبا دچار تغییر جهت شده و دنبال کردن آن دشوار می گردید. لذا نوشتن برنامه ها عملی بسیار زمان بر و پرهزینه شده بود و معمولا اشکال زدایی, اعمال تغییرات و گسترش برنامه ها بسیار مشکل بود. فعالیتهای پژوهشی در این دهه باعث بوجود آمدن سبک جدیدی از برنامه نویسی بنام روش ساختیافته گردید؛ روش منظمی که باعث ایجاد برنامه هایی کاملا واضح و خوانا گردید که اشکال زدایی و خطایابی آنها نیز بسیار ساده تر بود.

اصلی ترین نکته در این روش عدم استفاده از دستور پرش (goto) است. تحقیقات بوهم و ژاکوپینی نشان داد که می توان هر برنامه ای را بدون دستور پرش و فقط با استفاده از ۳ ساختار کنترلی ترتیب، انتخاب و تکرار نوشت.

ساختار ترتیب، همان اجرای دستورات بصورت متوالی (یکی پس از دیگری) است که کلیه زبانهای برنامه نویسی در حالت عادی بهمان صورت عمل می کنند.

ساختار انتخاب به برنامه نویس اجازه می دهد که براساس درستی یا نادرستی یک شرط، تصمیم بگیرد کدام مجموعه از دستورات اجرا شود.

ساختار تکرار نیز به برنامه نویسان اجازه می دهد مجموعه خاصی از دستورات را تا زمانیکه شرط خاصی برقرار باشد، تکرار نماید.
(برای شرح بیشتر موارد فوق به فصل ۳ مراجعه نمایید).

هر برنامه ساختیافته از تعدادی بلوک تشکیل می شود که این بلوکها به ترتیب اجرا می شوند تا برنامه خاتمه یابد(ساختار ترتیب). هر بلوک می تواند یک دستورساده مانند خواندن، نوشتن یا تخصیص مقدار به یک متغیر باشد و یا اینکه شامل دستوراتی باشد که یکی از ۳ ساختار فوق را پیاده سازی کنند. نکته مهم اینجاست که درمورد دستورات داخل هر بلوک نیز همین قوانین برقرار است و این دستورات می توانند از تعدادی بلوک به شرح فوق ایجاد شوند و تشکیل ساختارهایی مانند حلقه های تودرتو را دهند.

نکته مهم اینجاست که طبق قوانین فوق یک حلقه تکرار یا بطور کامل داخل حلقه تکرار دیگر است و یا بطور کامل خارج آن قرار می گیرد و هیچگاه حلقه های روی هم افتاده نخواهیم داشت.

از جمله اولین تلاشها در زمینه ساخت زبانهای برنامه نویسی ساختیافته، زبان پاسکال بود که توسط پروفسور نیکلاس ویرث در سال ۱۹۷۱ برای آموزش برنامه نویسی ساختیافته در محیطهای آموزشی ساخته شد و بسرعت در دانشگاهها رواج یافت. اما بدلیل نداشتن بسیاری از ویژگیهای مورد نیاز مراکز صنعتی و تجاری در بیرون دانشگاهها موفقیتی نیافت.

کمی بعد زبان C ارائه گردید که علاوه بر دارا بودن ویژگیهای برنامه نویسی ساختیافته بدلیل سرعت و کارایی بالا مقبولیتی همه گیر یافت و هم اکنون سالهاست که بعنوان بزرگترین زبان برنامه نویسی دنیا شناخته شده است.

مراحل اجرای یک برنامه C

برای اجرای یک برنامه C ابتدا باید آن را نوشت. برای اینکار می توان از هر ویرایشگر متنی موجود استفاده کرد و سپس فایل حاصل را با پسوند .C ذخیره نمود (فایلهای C++ با پسوند .CPP ذخیره می گردند). به این فایل، کد مبدا (source code) گفته می شود. مرحله بعدی تبدیل کد مبدا به زبان ماشین است که به آن کد مقصد (object code) گفته می شود. همانطورکه قبلا نیز گفته شد برای اینکار از یک برنامه مترجم بنام کامپایلر استفاده می شود. کامپایلرهای متعددی برای زبان C توسط شرکتهای مختلف و برای سیستم عاملهای مختلف نوشته شده است که می توانید برحسب نیاز از هریک از آنها استفاده نمایید. اما هنوز برنامه برای اجرا آماده نیست. معمولا برنامه نویسان از در برنامه های خود از یک سری از کدهای از پیش آماده شده برای انجام عملیات متداول (مانند محاسبه جذر و یا سینوس) استفاده می کنند که برنامه آنها قبلا نوشته و ترجمه شده است. این برنامه ها یا در قالب کتابخانه های استاندارد توسط شرکتهای ارائه کننده نرم افزار عرضه شده است و یا توسط دیگر همکاران برنامه نویس اصلی نوشته و در اختیار وی قرار داده شده است. در این مرحله باید کد مقصد برنامه اصلی با کدهای مربوط به این برنامه های کمکی پیوند زده شود. برای اینکار نیاز به یک پیوند زننده (Linker) داریم و نتیجه این عمل یک فایل قابل اجرا خواهد بود (در ویندوز این فایل پسوند .EXE خواهد داشت). مرحله بعدی اجرای برنامه و دادن ورودیهای لازم به آن و اخذ خروجیها می باشد. در شکل زیر این مراحل نشان داده شده اند.

مسلما طی مراحل بالا برای اجرای هر برنامه زمانبر می باشد، بهمین دلیل اکثر تولید کنندگان کامپایلرها، محیطهایی را برای برنامه نویسی ارائه کرده اند که کلیه مراحل بالا را بطور اتوماتیک انجام می دهند.

به این محیطها Development Environment IDE (Integrated) یا محیط مجتمع توسعه نرم افزار گفته می شود. این محیطها دارای یک ویرایشگر متن می باشند که معمولا دارای خواص جالبی همچون استفاده از رنگهای مختلف برای نشان دادن اجزای مختلف برنامه مانند کلمات کلیدی، و یا قابلیت تکمیل اتوماتیک قسمتهای مختلف برنامه می باشد. پس از نوشتن برنامه و با انتخاب گزینه ای مانند Run کلیه عملیات فوق بطور اتوماتیک انجام شده و برنامه اجرا می گردد. علاوه براین، این محیطها معمولا دارای امکانات اشکالزدایی برنامه (Debug) نیز می باشند که شامل مواردی همچون اجرای خط به خط برنامه و یا دیدن محتویات متغیرها در زمان اجرا است. چند محیط معروف برنامه نویسی عبارتند از :

Borland C++ 3.1 برای محیط DOS

Borland C++ از نسخه ۴ به بالا برای Windows

Microsoft Visual C++ برای محیط Windows
Borland C++ Builder برای محیط Windows

برای شروع ما از محیط Borland C++ 3.1 تحت Dos که نحوه کار ساده تری نسبت به سایرین دارد استفاده می کنیم.
پس از نصب این نرم افزار، برنامه BC.exe را اجرا کنید تا وارد محیط borland c شوید

همانطور که می بینید، این محیط از ۳ قسمت اصلی تشکیل شده است :

– **بخش ویرایش برنامه :** بخش آبی رنگ وسط می باشد که در حقیقت یک ویرایشگر است که برنامه در آن تایپ می شود. همانطور که می بینید در این ویرایشگر از رنگهای مختلف برای نشان دادن قسمتهای مختلف برنامه استفاده می شود. مثلا برای کلمات کلیدی از رنگ سفید استفاده شده است.

– **بخش منوهای کاری :** این بخش که در قسمت بالا واقع شده است،– حاوی تعدادی منو (گزینه) برای انجام وظایف مختلف است. خلاصه این عملیات عبارتند از :

- منوی File : عملیاتی مانند باز کردن و یا ذخیره یک برنامه
- منوی Edit : عملیات ویرایش مانند حذف،o کپی و یا چسباندن یک قسمت از برنامه
- منوی Search : جستجوی و یا تعویض یک متن در برنامه
- منوی Run : اجرای برنامه بصورت کامل یا دستور به دستور
- منوی Compile : عملیات مربوط به کامپایل و پیوند برنامه
- منوی Debug : عملیات مربوط به اشکالزدایی مانند دیدن مقادیر متغیرها در زمان اجرا
- منوی Project : عملیات مربوط به مدیریت برنامه هایی که شامل چندین فایل مستقل هستند (پروژه)
- منوی Options : عملیات مربوط به تنظیمات سیستم مانند نحوه کامپایل و یا رنگ پیش فرض محیط
- منوی Windows : عملیات مربوط به پنجره های باز فعلی (مربوط به چندین برنامه یا نمایش متغیرها و ...)

خطاهای برنامه نویسی

بنظر می رسد خطاها جزء جداناپذیر برنامه ها هستند. بندرت می توان برنامه ای نوشت که در همان بار اول بدرستی و بدون هیچگونه خطایی اجرا شود. اما خطاها از لحاظ تاثیری که بر اجرای برنامه ها می گذارند،

متفاوتند. گروهی ممکن است باعث شوند که از همان ابتدا برنامه اصلا کامپایل نشود و گروه دیگر ممکن است پس از گذشت مدتها و در اثر دادن یک ورودی خاص به برنامه، باعث یک خروجی نامناسب و یا یک رفتار دور از انتظار (مانند قفل شدن برنامه) شوند. بطور کلی خطاها به دو دسته تقسیم می شوند:

خطاهای نحوی (خطاهای زمان کامپایل): این خطاها در اثر رعایت نکردن قواعد دستورات زبان C و یا تایپ اشتباه یک دستور بوجود می آیند و در همان ابتدا توسط کامپایلر به برنامه نویس اعلام می گردد. برنامه نویس باید این خطا را رفع کرده و سپس برنامه را مجددا کامپایل نماید. لذا معمولا این قبیل خطاها خطر کمتری را در بردارند.

خطاهای منطقی (خطاهای زمان اجرا): این دسته خطاها در اثر اشتباه برنامه نویس در طراحی الگوریتم درست برای برنامه و یا گاهی در اثر درنظر نگرفتن بعضی شرایط خاص در برنامه ایجاد می شوند. متاسفانه این دسته خطاها در زمان کامپایل اعلام نمی شوند و در زمان اجرای برنامه خود را نشان می دهند. بنابراین، این خود برنامه نویس است که پس از نوشتن برنامه باید آن را تست کرده و خطاهای منطقی آن را پیدا کرده و رفع نماید. متاسفانه ممکن است یک برنامه نویس خطای منطقی برنامه خود را تشخیص ندهد و این خطا پس از مدتها و تحت یک شرایط خاص توسط کاربر برنامه کشف شود. بهمین دلیل این دسته از خطاها خطرناکتر هستند. خود این خطاها به دو دسته تقسیم می گردند:

a. **خطاهای مهلک:** در این دسته خطاها کامپیوتر بلافاصله اجرای برنامه را متوقف کرده و خطا را به کاربر گزارش می کند. مثال معروف این خطاها،b. خطای تقسیم بر صفر می باشد.

c. **خطاهای غیرمهلک:** در این دسته خطا،d. اجرای برنامه ادامه می یابد ولی برنامه نتایج اشتباه تولید می نماید. بعنوان مثال ممکن است دراثر وجود یک خطای منطقی در یک برنامه حقوق و دستمزد،e. حقوق کارمندان اشتباه محاسبه شود و تا مدتها نیز کسی متوجه این خطا نشود!

با توجه به آنچه گفته شد، در می یابیم که رفع اشکال برنامه ها بخصوص خطاهای منطقی از مهمترین و مشکلترین وظایف یک برنامه نویس بوده و گاهی حتی سخت تر از خود برنامه نویسی است! بهمین دلیل است که بسیاری از شرکتها(همانند مایکروسافت) ابتدا نسخه اولیه نرم افزار خود را در اختیار کاربران قرار می دهند تا اشکالات آن گزارش شده و رفع گردد. بسیار مهم است که در ابتدا سعی کنید برنامه ای بنویسید که حداقل خطاها را داشته باشد، در گام دوم با آزمایش دقیق برنامه خود هرگونه خطای احتمالی را پیدا کنید و در گام

سوم بتوانید دلیل بروز خطا را پیدا کرده و آنرا رفع نمایید. هر سه عمل فوق کار سختی بوده و نیاز به تجربه و مهارت دارد.

آخرین نکته اینکه در اصطلاح برنامه نویسی به هر گونه خطا، bug و به رفع خطا debug گفته می شود.

یک برنامه نمونه

در این قسمت برای آشنایی اولیه با زبان C یک برنامه نمونه آورده شده است که بدون هیچ تغییری در محیط BorlandCPP قابل اجرا است.

```
// This Program Computes the Area of a Circle
#include <stdio.h>
void main() {
int radius ;
float area;
printf("please enter radius : ");
scanf("%d",&radius);
area = 2 * 3.14 * radius;
printf("Area is %f",area);
{
please enter radius : 10
Area is 62.8
```

درمورد برنامه فوق به نکات زیر توجه کنید :

– خط اول یک توضیح درمورد برنامه است. در زبان C برای توضیحات یک خطی از علامت // استفاده می گردد. اما چنانچه توضیحات بیش از یک خط بود،– آن را با علامت /* شروع کرده و با */ پایان دهید. کامپایلر از این توضیحات صرفنظر خواهد کرد. این توضیحات باعث می شوند که برنامه شما خواناتر شده و دیگران بهتر آن را درک کنند.

– هر دستوری که با علامت # شروع شود،– یک دستور C نیست،– بلکه جزو دستورات پیش پردازنده محسوب می گردد. دستورات پیش پردازنده،– دستوراتی هستند که توسط کامپایلر قبل از شروع به کامپایل انجام می شوند. بعنوان مثال دستور #include باعث می شود که تعاریف اولیه مربوط به توابعی (زیربرنامه هایی) که قصد استفاده از آنها را داریم به برنامه اضافه شود. در مثال بالا برای استفاده از توابع printf و scanf که در کتابخانه استاندارد C تعریف شده اند،– فایل سرآمد stdio.h را که این توابع در آن تعریف شده اند را استفاده کرده ایم.

– هر برنامه C باید دارای تابعی به نام main باشد که اجرای برنامه از آن شروع می شود و در حقیقت همان برنامه اصلی است. البته می توان هر تعداد دیگری تابع (زیربرنامه) نیز تعریف کرد،– اما وجود تابع main الزامی است. دقت کنید که گرچه این تابع پارامتر ورودی ندارد،– اما از پرانتز باز و بسته تنها استفاده شده است.

– در زبان C هر بلوک برنامه با علامت { آغاز شده و با } پایان می یابد. این دو معادل دستورات begin و end در زبانهای دیگر ازجمله پاسکال می باشند که برای سادگی زبان انتخاب شده اند.

– دو خط بعدی به تعریف متغیرهای radius و area می پردازد. در زبان C قبل از استفاده از هر متغیری باید آن را اعلان نمایید. اعلان متغیر شامل نام و نوع متغیر است. در مثال فوق،– متغیر radius از نوع عدد صحیح(integer) و متغیر area از نوع عدد اعشاری (float) تعریف شده اند.

– توابع printf و scanf جزو کتابخانه استاندارد C محسوب می گردند و به ترتیب برای چاپ اطلاعات در خروجی استاندارد (نمایشگر) و دریافت اطلاعات از ورودی استاندارد (صفحه کلید) استفاده می شوند. برای چاپ رشته مورد نظر باید آنها را در داخل علامت " قرار داد. درمورد این توابع بعداد توضیح خواهیم داد.

– دقت کنید که در پایان هر دستورالعمل از علامت ; استفاده شده است. در مجموع C یک زبان قالب آزاد است و شما می توانید دستورات را به هر نحوی که دوست دارید قرار دهید (مثلا چند دستور در یک خط از برنامه). تنها چیزی که نشاندهنده پایان یک دستور است،– علامت ; است(و نه انتهای خط).

– از آنجا که C یک زبان قالب آزاد است،– می توان با استفاده از مکان نوشتن دستورات شکل بهتری به برنامه داد. بعنوان مثال دقت کنید که پس از شروع تابع main،– دستورات حدود ۳ کاراکتر جلوتر نوشته شده اند. به این نحوه نوشتن دستورات دندانه گذاری می گویند. بطور کلی هربار که بلوک جدیدی آغاز می شود،– باید آن را کمی جلوتر برد. این مسئله باعث جدا شدن بلوکها از یکدیگر و خوانایی بهتر برنامه می شود.

– در پایان برنامه و در داخل مستطیل خاکستری،– یک نمونه از اجرای برنامه که شامل یک ورودی و خروجی نمونه است،– آورده شده است.

مفاهیم اولیه زبان C

در این مقاله به مقدمات اولیه برنامه نویسی به زبان C می پردازیم و اصول اولیه آن را بررسی می نماییم....

شناسه ها در C

شناسه (identifier) نامی است که به یک قسمت از برنامه مانند متغیر، تابع، ثابت و یا ... داده می شود. در زبان C برای انتخاب شناسه ها فقط می توان از علائم زیر استفاده کرد: – حروف انگلیسی کوچک و بزرگ (A...Z a...z)

– ارقام (۰...۹)

– علامت خط پایین یا _

البته یک شناسه تمی تواند با یک رقم شروع شود. مثلا شناسه های sum، average، name2 و یا student_average مجاز هستند، اما شناسه های name۲ و یا student average مجاز نیستند. البته در برنامه نویسی امروزی پیشنهاد می شود بجای شناسه هایی همانند student_average از studentAverage استفاده گردد. یعنی بجای اینکه در شناسه ها از _ برای جداکننده استفاده کنیم، اولین حرف هر قسمت را بصورت بزرگ بنویسیم.

نکته مهم دیگری که باید به آن اشاره کرد آنستکه زبان C برخلاف بسیاری از زبانهای دیگر به کوچک و بزرگی حروف حساس است (case sensitive). در نتیجه شناسه های زیر با یکدیگر متفاوتند :

```
Sum ≠ sum ≠ SUM
```

این مسئله معمولا درهنگام برنامه نویسی باعث ایجاد بعضی خطاها می شود.
آخرین نکته اینستکه در هنگام انتخاب شناسه نمی توانید از کلمات کلیدی که برای منظورهای خاص در زبان C رزرو شده اند استفاده کنید. زبان C دارای ۳۲ کلمه کلیدی است که عبارتند از :

Const	char	case	break	auto
Else	double	do	default	continue
Goto	for	float	extern	enum
Return	register	long	int	if
Struct	static	sizeof	signed	short

Voidunsigned uniontypedef switch
while volatile

البته در نسخه های جدید C کلمات کلیدی دیگری نیز به لیست فوق اضافه شده است. دقت کنید که کلیه این کلمات با حروف کوچک نوشته شده اند.

انواع داده ها در C

همانطور که قبلا نیز گفتیم هر متغیر پیش از آنکه استفاده گردد ابتدا باید اعلان گردد. اعلان یک متغیر تعیین نوع آن را نیز دربر می گیرد. سوال اصلی آنستکه چه نوع داده هایی در زبان C و جود دارد. انواع داده های متداول عبارتند از :

نوع داده	توضیح	اندازه (بیت)	محدوده
char	کاراکتر	۸	to ۱۲۸– ۱۲۷+
int	عدد صحیح	۱۶	to ۳۲۷۶۸– ۳۲۷۶۷ +
float	عدد اعشاری	۳۲	e-38 ۳,۴ to e+38۳,۴
double	عدد اعشاری با دقت مضاعف	۶۴	e-308 ۱,۷ to e+308۱,۷

البته چند نکته مهم درمورد جدول فوق قابل ذکر است :

– اندازه int در محیطهای ۱۶ بیتی مانند DOS برابر ۱۶ بیت است. اما در محیطهای ۳۲ بیتی همانند Windows اندازه آن ۳۲ بیت می باشد که در اینصورت محدوده ای برابر –۲،۱۴۷،۴۸۳،۶۴۷ تا +۲،۱۴۷،۴۸۳،۶۴۷ را پوشش می دهد.

– در بعضی از کامپایلرهای C، نوع داده bool نیز وجود دارد که می تواند یکی از مقادیر true(درست) یا false (غلط) را نشان دهد . اما در نسخه های اولیه C از همان نوع داده صحیح یا int برای اینکار استفاده می شد. بدین صورت که ۰ نشاندهنده false و هر عدد غیر صفر (معمولا ۱) نشاندهنده true است.

– از آنجا که برای ذخیره سازی کاراکترها کد اسکی آنها ذخیره می گردد، از یک متغیر کاراکتری یا char می توان بعنوان یک عدد صحیح کوچک نیز استفاده کرد و اعمال ریاضی برروی آنها نیز مجاز است.

علاوه براین چندین اصلاح کننده نیز وجود دارد که به ما اجازه می دهد نوع داده مورد نظر را با دقت بیشتری برای نیازهای مختلف استفاده نماییم.

این اصلاح کننده ها عبارتند از : unsigned ,short, long, signed

تمام اصلاح کننده های فوق می توانند به نوع داده int اعمال شوند و اثر آنها بستگی به محیط دارد. مثلا در یک محیط ۱۶ بیتی، short int بازهم برابر ۱۶ بیت است ولی long int برابر ۳۲ بیت خواهد بود. علاوه براین unsigned int باعث می شود که یک عدد ۱۶ بیتی بدون علامت داشته باشیم که بازه بین ۰ تا ۶۵۵۳۵ را پوشش می دهد. signed int نیز همانند int معمولی بوده و تفاوتی ندارد. البته ترکیب این اصلاح کننده ها نیز ممکن است. مثلا unsigned long int یک عدد ۳۲ بیتی بدون علامت است که بازه ۰ تا ۴،۲۹۴،۹۶۷،۲۹۵ را پوشش می دهد.

اما برروی نوع داده char فقط اصلاح کننده های signed و unsigned را می توان اعمال کرد. معمولا از اصلاح کننده unsigned وقتی استفاده می شود که قصد داشته باشیم از char بعنوان یک عدد صحیح مثبت بین ۰ تا ۲۵۵ استفاده کنیم.

برروی نوع داده double تنها اصلاح کننده long قابل اعمال است و دراینصورت عدد اعشاری با ۸۰ بیت را خواهیم داشت که قادر است هر عددی در بازه ۳,۴e-4932 تا ۱,۱e+4932 را در خود نگاه دارد.

تعریف متغیرها

برای تعریف متغیرها به شکل زیر عمل می کنیم:

```
<type> <variable-list>;
```

که type یکی از نوع داده های گفته شده و variable-list لیستی از متغیرها است که با کاما از یکدیگر جدا شده اند. بعنوان مثال :

```
int sum;
float average;
long int a, b, c ;
unsigned long int i, j, k ;
```

علاوه براین می توان در هنگام تعریف متغیر به آن مقدار اولیه نیز داد. مثال :

```
int d = 0 ;
```

نکته مهم آنکه زبان C به متغیرها مقدار اولیه نمی دهد (حتی ۰) و برنامه نویس خود باید اینکار را صریحا انجام دهد، درغیر اینصورت مقدار اولیه متغیر، نامعین خواهد بود.
تعریف متغیرها طبق اصول زبان C میتواند درهرجایی از برنامه صورت پذیرد، و متغیرهای تعریف شده از همان خط به بعد قابل استفاده خواهد بود. اما معمولا توصیه می گردد که تعریف متغیرها در همان خط ابتدایی تابع و بلافاصله پس از { صورت پذیرد.

ثوابت

ثابتها مقادیر ثابتی هستند که مقدار آنها در حین اجرای برنامه تغییر نمی یابد. ثابتها می توانند از هریک از نوع داده های اصلی باشند. برای نمایش هر ثابت بسته به نوع آن عمل می کنیم که شرح هریک در زیر آمده است :
۴ ثوابت عددی صحیح : برای نمایش این دسته از ثوابت، از دنباله ای از ارقام بعلاوه علامت + یا – استفاده می کنیم. بعنوان مثال ۴۵- و یا ۳۴۸۹ ثوابت صحیح هستند. در حالت عادی C هر عدد را در کوچکترین نوع داده ای که می تواند قرار می دهد. مثلا عدد ۸۵ در یک int قرار می گیرد، اما عدد ۱۴۵۳۹۸ در یک long int قرار خواهد گرفت. اما اگر قصد دارید یک عدد کوچک بعنوان long محسوب گردد، می توانید از پسوند L در انتهای آن استفاده می کنیم. مثلا L۲۴۵ یک عدد long محسوب می شود.

```
long int a = 20L;
```

ضمن اینکه پسوند U در انتهای عدد نیز نشانه بدون علامت بودن آن است.
نکته دیگر آنکه C به شما اجازه می دهد درصورت لزوم ثوابت صحیح خود را در مبنای ۸ یا ۱۶ نیز که از

مبناهای متداول در برنامه نویسی هستند، بنویسید. برای نوشتن عدد در مبنای ۸ باید آن را با ۰ آغاز کنید، مثلا ۰۳۴۲ یک عدد در منای ۸ محسوب می گردد. اما برای نوشتن یک عدد در مبنای ۱۶ باید آن را با x۰ آغاز نمایید، مانند x27A4۰.

۸ ثوابت عددی اعشاری : برای نمایش اعداد اعشاری، باید از نقطه اعشار استفاده کنیم، مانند ۲۳۷,۴۵ ، اما نکته جالب آنستکه می توانید از نماد علمی نیز برای نمایش اعداد اعشاری استفاده کنید. برای اینکار کافی است از حرف e برای نمایش قسمت توان استفاده نمایید. بعنوان مثال :

```
-23.47 × 10 5 = -23.47e5
42.389 × 10 -3 = 42.389e-3
```

دقت کنید که قسمت توان، حتما یک عدد صحیح است. نکته جالب اینجاست که برخلاف مورد قبل، کامپایلر بطور پیش فرض داده های اعشاری را از نوع double فرض می کند.چنانچه دوست دارید ثابت شما ازنوع float درنظر گرفته شود، در انتهای آن F قرار دهید. ضمنا پسوند L نیز داده اعشاری را از نوع long double درنظر می گیرد. ۱۰ ثوابت کاراکتری : برای نشان دادن ثوابت کاراکتری، آنها را در داخل ' قرار می دهیم. بعنوان مثال 'A' یک ثابت کاراکتری است.

```
char ch = 'S' ;
```

دقت کنید که همانطور که قبلا گفته شد، کد اسکی کاراکترها در متغیر ذخیره می گردد، بنابراین می توان از آن بعنوان یک عدد صحیح نیز استفاده کرد. نکته مهم دیگر آنستکه به تفاوت عدد ۵ و کاراکتر '۵' دقت داشته باشید. در حقیقت '۵' برابر است با عدد ۵۳ که همان کد اسکی آن است.

از آنجا که بعضی کاراکترها، مانند enter قابل نمایش نیستند، بهمین دلیل آنها را با استفاده از \ نمایش می دهیم. علاوه براین بعضی کاراکترها مانند خود " نیز چون دارای معنای خاص هستند، باید با استفاده از \ نمایش داده شوند. موارد مهم عبارتند از :

```
\n خط جدید
\t tab کاراکتر
\a بوق کامپیوتر \" "
\' ' \\ \
```

۱۴ ثوابت رشته ای : C علاوه بر ثوابت فوق، از یک ثابت دیگر بنام رشته نیز حمایت می کند. رشته، دنباله ای

از کاراکترها است که در داخل " قرار می گیرند. بعنوان مثال "this is a test" یک رشته است. دقت کنید که 'a' یک کاراکتر است، اما "a" یک رشته است که فقط شامل یک کاراکتر می باشد.

عملگرها

عملگر، نمادی است که به کامپایلر می گوید تا عملیات محاسباتی یا منطقی خاصی را برروی یک یا چند عملوند، انجام دهد. به عملگرهایی که فقط یک عملوند دارند، عملگر یکانی می گوییم و همواره عملگر در سمت چپ عملوند قرار می گیرد(مانند عدد -۱۲۵). اما عملگرهایی که برروی دو عملوند اثر می کنند را عملگر دودویی نامیده و عملگر را بین دو عملوند قرار می دهیم (مانند ۲۳+۸۶). هر ترکیب درستی از عملگرها و عملوندها را یک عبارت می نامیم.

C از نقطه نظر عملگرها یک زبان بسیار قوی است. این عملگرها به چند دسته اصلی تقسیم می گردند که آنها را به ترتیب بررسی می کنیم..

عملگرهای محاسباتی

این عملگرها، همان اعمال متداول ریاضی هستند که در زبان C مورد استفاده قرار می گیرند. این اعمال عبارتند از :

عمل	نوع	عملگر
منفی کردن عملوند سمت راست	یکانی	-
جمع دو عملوند	دودویی	+
تفریق دو عملوند	دودویی	-
ضرب دو عملوند	دودویی	*
تقسیم دو عملوند	دودویی	/
محاسبه باقیمانده تقسیم دو عملوند	دودویی	%

عملگرهای فوق برروی همه انواع داده های C عمل می کنند بجز عملگر ٪ که فقط برروی نوع داده های صحیح عمل میکند و برروی داده های اعشاری تعریف نشده است. اما نکته مهمی که باید به آن اشاره کرد، نحوه کار عملگر تقسیم برروی نوع داده های مختلف است. درصورتیکه هردو عملوند صحیحی باشند، تقسیم بصورت صحیح بر صحیح انجام خواهد شد. اما اگر یکی یا هر دو عملوند اعشاری باشند، تقسیم بصورت اعشاری انجام خواهد پذیرفت. فرض کنید تعاریف زیر را داریم :

```
int a,b ;
float c,d ;
a = 10 ; b = 4 ;
c = 8.2; d = 4.0;
```

اکنون به نتایج عملیات زیر دقت کنید :

```
a / b = < 2
c / d = < 2.05
a / d = < 2.5
a / 4 = < 2
```

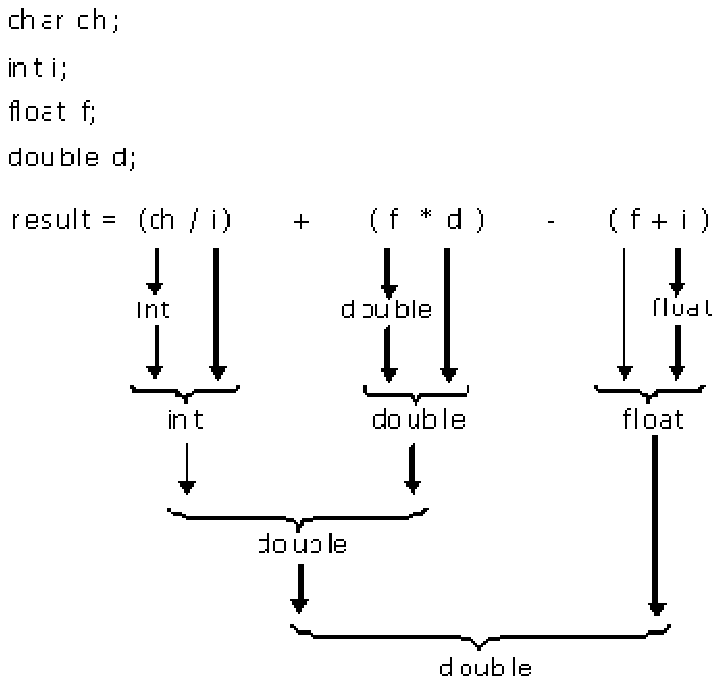
چنانچه به آخرین مورد توجه کنید، متوجه می شوید که از آنجا که a یک متغیر صحیح است و ۴ نیز یک ثابت صحیح در نظر گرفته شده، درنتیجه تقسیم بصورت صحیح بر صحیح انجام گرفته است. چنانچه بخواهیم تقسیم بصورت اعشاری صورت پذیرد، به دو شکل می توانیم عمل کنیم. اول اینکه آن را بصورت a / 4.0 بنویسیم و دوم اینکه از عملگر قالب ریزی استفاده کنیم.
عملگر قالب ریزی می تواند یک نوع را به نوع دیگری تبدیل نماید. شکل کلی آن به شکل زیر است :

```
(< type >) < expression >
```

که type نوع مورد نظر است که قصد تبدیل عبارت expression به آن نوع را داریم. بعنوان مثال در مورد بالا می توان به شکل زیر عمل کرد :

```
(float) a / 4
```

در این حالت ابتدا از کامپایلر خواسته ایم که ابتدا عدد a را به اعشاری تبدیل کند (البته بصورت موقت) و سپس آن را بر ۴ تقسیم نماید، که مسلما حاصل اعشاری خواهد بود.
بطورکلی هرگاه ثابتها و متغیرهایی از انواع مختلف در یک عبارت محاسباتی داشته باشیم، کامپایلر C همه آنها را به یک نوع یکسان که همان بزرگترین عملوند موجود است تبدیل خواهد کرد. بعنوان مثال به مورد زیر دقت کنید :



نکته بسار مهم دیگری که باید به آن توجه کرد، اولویت عملگرها است. یعنی در عبارتی که شامل چندین عملگر است، کدامیک در ابتدا اعمال خواهد گردید. اولویت عملگرهای محاسباتی از بالا به پایین بشرح زیر است:

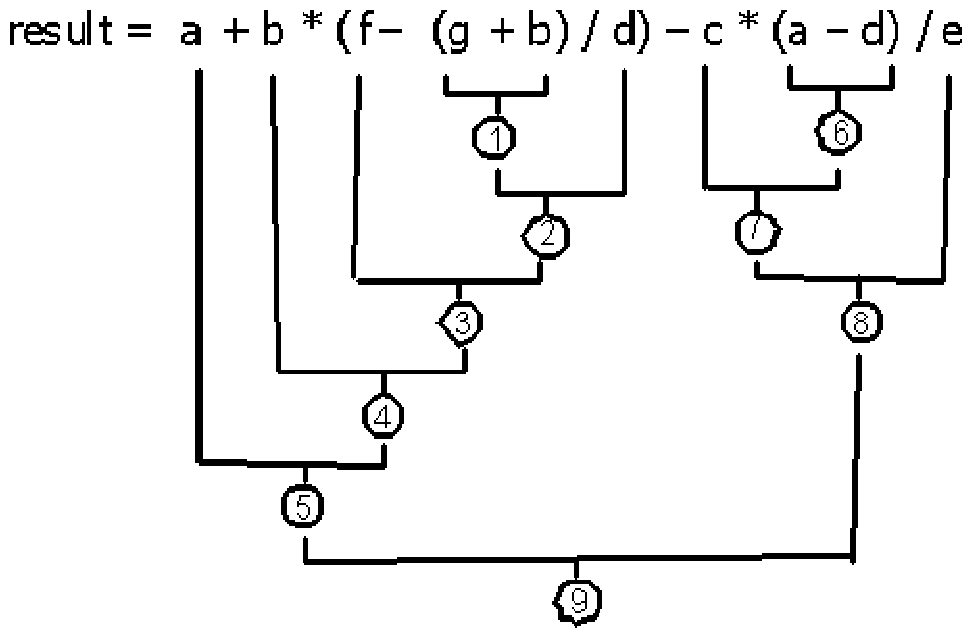
۱ عملگر یکانی –
۲ عملگرهای * و / و ٪
–و +عملگرهای 3

چنانچه اولویت دو عملگر یکسان باشد، این عملگرها از چپ به راست محاسبه خواهند شد. چنانچه بخواهیم یک عمل با اولویت پایین زودتر انجام شود، باید از پرانتز استفاده کنیم. بنابراین اولویت عملگر پرانتز از همه

موارد فوق بیشتراست. در مورد پرانتزهای متداخل، ابتدا پرانتز داخلی محاسبه می شود؛ اما در مورد پرانتزهای هم سطح، ابتدا پرانتز سمت چپتر محاسبه می گردد. به چند مثال زیر دقت کنید:

عبارت زبان C	ترتیب اجرای عملگرها
b + c * d	ابتدا عمل * و سپس عمل +
b + c) * d)	ابتدا عمل + و سپس عمل *
b + c / d * e	ابتدا عمل / سپس عمل * و در انتها عمل +
(b + c / (d * e	ابتدا عمل * سپس عمل / و در انتها عمل +

مثال زیر، یک مثال کامل در این زمینه می باشد :



عملگرهای انتساب

در زبان C برای انتساب چندین عملگر وجود دارد. ساده ترین عملگر انتساب، همان عملگر = است که در بسیاری از زبانها استفاده می شود. بعنوان مثال :

```
a = 5;
b = c + 2 * d;
```

این عملگر باعث می شود که عبارت سمت راست در عبارت سمت چپ قرار گیرد. توجه کنید که مقدار سمت چپ باید عبارتی باشد که بتوان به آن یک مقدار را نسبت داد(مانند یک متغیر) که به آن Lvalue گفته می شود، بنابراین یک ثابت نمی تواند در سمت چپ قرار گیرد. نکته دیگر اینکه اولویت عملگر = از عملگرهای ریاضی کمتر است و درنتیجه ابتدا آن عملیات انجام شده و در پایان حاصل در عبارت سمت چپ ریخته می شود. لازم به ذکر است که در هنگام انتساب، درصورت لزوم نوع عبارت سمت راست به نوع عبارت سمت چپ تبدیل می شود. مثال:

```
int a;
a = 2.5 * 5.0;
```

که دراینصورت عدد ۱۲ در a ذخیره خواهد شد.
شرکت پذیری این عملگر از راست به چپ می باشد، بدین معنا که چنانچه چندین عملگر نسبت دهی داشته باشیم، این عملگرها از راست به چپ محاسبه می شوند. مثلا پس از اجرای دستور زیر :

```
a = b = c = 10;
```

مقدار هر ۳ متغیر برابر ۱۰ خواهد شد.
نکته جالب درمورد زبان C آنستکه دارای یک سری عملگرهای انتساب خلاصه شده است که باعث می شوند در بعضی موارد بتوانیم عبارات کوتاهتری را بنویسیم. این عملگرها عبارتند از :

عملگر	مثال	عبارت انتساب معادل
++	a++;	a = a + ۱;
--	a--;	a = a - ۱;
=+	a =+ ۵;	a = a + ۵;
=-	a =- ۸;	a= a - ۸;
=*	a =* ۱۰;	a= a * ۱۰;
=/	a =/ ۲;	a = a / ۲;

a = ٪ a ;۱۰	a =٪ ;۱۰	=٪

عملگر اول و دوم، عملگرهای یکانی هستند که بترتیب عملگر افزایش و عملگر کاهش نامیده می شوند. نکته جالب آنستکه این دو عملگر به دوصورت مورد استفاده قرار می گیرند. در حالتی که از دستور a ++; استفاده شود به آن پس افزایش می گویند و بدین معناست که ابتدا از مقدار فعلی a در عبارت موردنظر استفاده کن و سپس آن را افزایش بده. اما دستور ++ a که به آن پیش افزایش گفته می شود، ابتدا a را افزایش داده و سپس از آن در عبارت استفاده می کند. به برنامه زیر دقت کنید:

```
#include < stdio.h >
void main() {
int a ,b ;
a = 5 ;
b = a ++;
printf("a=%d and b=%d \n",a,b);
a = 5 ;
b = ++ a;
printf("a=%d and b=%d \n",a,b);
}
```

```
a=6 b=5
a=6 b=6
```

و اما پنج عملگر بعدی هنگامی می توانند استفاده شوند که عبارتی بصورت زیر داشته باشیم:

عبارت عملگر متغیر = متغیر

که عملگر یکی از عملگرهای محاسباتی (+ ، − ، * یا /) باشد، دراینصورت می توانیم آن را بصورت زیر بنویسیم:

عبارت =عملگر متغیر

عملگرهای مقایسه ای (رابطه ای)

این عملگرها دو عبارت را بایکدیگر مقایسه کرده و نتیجه را باز می گردانند. نتیجه می تواند درست (true) یا غلط (false) باشد. همانطور که قبلا نیز گفته شد در کامپایلرهای جدید C++ نوع داده bool اضافه شده است و بنابراین نتیجه این عملگرها یک مقدار bool است. اما در کامپایلرهای قدیمیتر، نتیجه این عملگرها یک عدد صحیح است که درصورت درست بودن ۱ و درصورت غلط بودن ۰ باز می گردانند. این عملگرها عبارتند از:

عملگر	مفهوم عملگر	مثال
>	بزرگتر (>)	a < b
<	کوچکتر (<)	a > b
>=	بزرگتر یا مساوی (≥)	a =< b
<=	کوچکتر یا مساوی (≤)	a => b
==	مساوی (=)	a == b
=!	نامساوی (≠)	a =! b

نکته مهمی که باید به آن دقت کرد عملگر مساوی (==) است، چرا که یک اشتباه بسیار متداول برنامه نویسان C استفاده اشتباه از عملگر انتساب (=) بجای عملگر تساوی (==) است که باعث ایجاد خطا در برنامه می شود.

اولویت این عملگرها از بالا به پایین بشرح زیر است:

۳ عملگرهای <، > ، =< و = >

۴ عملگرهای == و =!

لازم بذکر است که در هنگام مساوی بودن اولویت چند عملگر، این عملگرها از چپ به راست محاسبه می گردند.

عملگرهای منطقی

این عملگرها به شما اجازه می دهند که شرطهای ساده ای را که با استفاده از عملگرهای مقایسه ای ایجاد شده اند را با یکدیگر ترکیب نموده و شرطهای پیچیده تری را بسازید. این عملگرها عبارتند از :

عملگر	مفهوم عملگر	نحوه کار	مثال
&&	AND منطقی	اگر هر دو عملوند درست باشند, درست و در غیر اینصورت نادرست باز می گرداند.	۰<a && sw==1
||	OR منطقی	اگر هر دو عملوند نادرست باشند, نادرست	۱۰۰=>a || b!=0

	و در غیر اینصورت درست باز می گرداند.		
!(a==1 || b<۱۰)	اگر عملوند درست باشد، نادرست و اگر نادرست باشد، درست برمی گرداند.	NOT منطقی	!

همانطور که قبلا نیز گفته شد مقدار بازگشتی این عملگرها، درست (۱) یا نادرست (۰) می باشد. اولویت این عملگرها بترتیب عبارتست از :

۱- عملگر !

۲- عملگر &&

۳- عملگر ||

عملگر شرطی

گاهی لازم است که ابتدا یک شرط بررسی شده و سپس برمبنای نتیجه (درست یا نادرست بودن آن) یکی از دو عبارت ممکن بازگردانده شود. گرچه معمولا برای اینکار از دستور if (که در فصل بعدی بحث شده است) استفاده می شود، اما یک عملگر ۳تایی (با ۳ عملوند) نیز برای آن وجود دارد. شکل کلی این عملگر بصورت زیر است:

```
عبارت۲ : عبارت۱ ? شرط
```

نحوه کار بدینصورت است که درصورت درست بودن شرط عبارت۱ و در غیراینصورت عبارت ۲ بازگردانده می شود. مثال :

```
a = (k<10) ? 100 : 50;
```

که این عبارت معادل دستور زیر است:

```
if(k<10) a=100;
else a=50;
```

البته این عبارت به شکلهای پیچیده نیز می تواند مورد استفاده قرار گیرد. مثلا :

```
c += (a>0 &&a<10) ? ++a : a/b;
```

معادل است با که :

```
if (a>0 &&a<10) {
a= a + 1;
c = c + a;
}
else c = c + a/b;
```

که البته توصیه می شود از این عبارات پیچیده استفاده نشود.

چند نکته درمورد عملگرها

در یک عبارت می توان بین عملوندها و عملگرها از فضاهای خالی به میزان دلخواه استفاده کرد. گرچه استفاده از فضای خالی اجباری نیست اما معمولا باعث بالا رفتن خوانایی عبارت می گردد. علاوه براین در عبارات پیچیده می توان از پرانتز برای نشان دادن ترتیب اجرای عملگرها، حتی درمواردی که مورد نیاز نباشد استفاده کرد. بعنوان مثال به مورد زیر توجه کنید:

شکل نامناسب

```
result=a-b*c/d+f*g;
```

شکل بهتر

```
result = a – b * c / d + f * g
```

شکل کاملا مناسب

```
result = a – ( b * c / d) + (f * g)
```

مورد دیگری که باید به آن اشاره کرد، نحوه محاسبه عبارات پیچیده ای است که عملگرهای مختلفی در آنها استفاده شده باشد. درچنین حالتی باید از اولویت گروههای مختلف عملگرها نسبت به یکدیگر آگاه باشیم.
جدول زیر اولویت عملگرهای گفته شده تاکنون را بترتیب از بالا به پایین آ ورده است:

اولویت	عملگر	شرکت پذیری
۱	()	از چپ به راست
۲	++ -- ! + -	از راست به چپ
۳	% / *	از چپ به راست
۴	- +	از چپ به راست
۵	<< >>	از چپ به

		راست
۶	=< < => >	از چپ به راست
۷	=! ==	از چپ به راست
۸	&&	از چپ به راست
۹	||	از چپ به راست
۱۰	?:	از راست به چپ
۱۱	=٪ =/ =* =- =+ =	از راست به چپ
۱۲	,	از چپ به راست

خواندن و نمایش اطلاعات

همانطور که قبلا نیز گفته شد، یکی از اهداف زبان C قابل حمل بودن آن است. بهمین منظور سعی شده است که از دستوراتی که ممکن است وابسته به ماشین خاصی باشد، اجتناب گردد. بهمین دلیلی زبان C برخلاف سایر زبانها دارای هیچ دستوری برای خواندن از ورودی و یا نوشتن در خروجی نیست.

اما در عوض دارای تعدادی تابع (زیربرنامه) استاندارد می باشد که تقریبا تمامی کامپایلرها از آنها حمایت می کنند. این کار به کامپایلر این امکان را می دهد که بر حسب نوع سخت افزار موردنظر، توابع ورودی و خروجی را طراحی نماید. این توابع در یک فایل سرآمد بنام stdio.h تعریف شده اند و بنابراین قبل ازاینکه بتوانید از این توابع استفاده نمایید، باید این فایل را نیز با استفاده از #include در برنامه خود بگنجانید.

تابع نمایش در خروجی

برای نمایش اطلاعات در خروجی از تابع printf استفاده می شود. این تابع رشته موردنظر شما را به خروجی استاندارد (که در حالت عادی همان صفحه نمایش یا مانیتور است) می فرستد. شکل کلی این تابع بصورت زیر است:

```
printf(< لیست متغیرها > , < رشته کنترلی >) ;
```

رشته کنترلی همان متنی است که قصد چاپ آن را داریم، با ذکر این نکته که در قسمتهایی که باید مقدار یک متغیر چاپ شود، از یک مشخصه تبدیل استفاده می شود. هر مشخصه تبدیل از یک علامت ٪ بعلاوه یک کاراکتر که نوع متغیر مورد نظر را نشان می دهد تشکیل می گردد. مشخصه تبدیل های متداول عبارتند از :

مشخصه تبدیل	مفهوم
c٪	کاراکتر
d٪	عدد صحیح در مبنای ۱۰
f٪	عدد اعشاری بدون نماد علمی
e٪	عدد اعشاری با نماد علمی
g٪	عدد اعشاری با حالت کوتاهتر بین e و f
s٪	رشته
ld٪	عدد صحیح بزرگ
lf ٪	عدد اعشاری بزرگ
%le	

lg٪	
o٪	عدد صحیح در مبنای ۸
x٪	عدد صحیحی در مبنای ۱۶
u٪	عدد صحیح بدون علامت

البته بعضی موارد دیگر نیز وجود دارد که از بحث فعلی ما خارج است. توجه کنید که از آنجا که رشته کنترلی یک ثابت رشته ای است، باید در داخل " قرار گیرد. لیست متغیرها نیز همان متغیرهایی هستند که قصد چاپ آنها را داریم. این متغیرها باید بترتیب قرار گرفته و با کاما (,) از یکدیگر جدا شوند. برای نمونه به مثال زیر توجه کنید:

```
#include <stdio.h>
void main() {
int age = 20;
floate average = 18.23;
printf("You are %d years old and your average is %f
\n",age,average);
}
You are 20 years old and your average is 18.230000
```

دقت کنید که علامت \n انتهای رشته کنترلی باعث می شود که خروجی بعدی برنامه (در صورت وجود) در خط بعد چاپ شود. البته ممکن است نحوه چاپ عدد اعشاری به شکل فوق چندان برای شما دلخواه نباشد و بخواهید تعداد ارقام اعشاری قابل نمایش را محدود کنید. برای اینکار باید از مشخصه طول میدان استفاده کنید. مشخصه طول میدان برای اعداد به شکل زیر استفاده می شود:

برای اعداد صحیح از ٪nd استفاده می کنیم که n تعداد ارقام را نشان می دهد (مثلا ٪d۳). در اینصورت برای هر متغیر n رقم درنظر گرفته می شود. اگر اندازه عدد از n کوچکتر باشد، به سمت چپ آن فضای خالی اضافه می شود و اگر اندازه عدد بیش از n رقم باشد، طول میدان نادیده گرفته شده و عدد بطور کامل چاپ می شود.

برای اعداد اعشاری از ٪n.mf استفاده می کنیم که n اندازه کل عدد (شامل علامت ممیز) و m تعداد ارقام اعشار است (مثلا ٪f۵,۲). در صورتیکه تعداد ارقام اعشاری عدد موردنظر از m بیشتر باشد، عدد به m عدد اعشار گرد می شود و در صورتیکه از m کمتر باشد، در سمت راست آن ۰ قرار داده می شود.

البته موارد دیگری نیز در مورد نحوه چاپ وجود دارد که برای اطلاع از آنها می توانید به قسمت Help کامپایلر خود مراجعه نمایید.

آخرین نکته درمورد تابع printf اینکه این تابع مقدار بازگشتی نیز دارد که درصورت موفقیت عمل چاپ، تعداد کاراکترهای چاپ شده و درصورت عدم موفقیت مقدار -۱ را باز می گرداند. البته معمولا از این مقدار بازگشتی صرفنظر می گردد.

تابع خواندن از ورودی

برای خواندن اطلاعات از ورودی از تابع scanf استفاده می شود. این تابع اطلاعات را ازورودی استاندارد (معمولا صفحه کلید) خوانده و در متغیرهای تعیین شده قرار می دهد. شکل کلی این تابع بصورت زیر است:

```
scanf(< لیست متغیرها > , < رشته کنترلی >) ;
```

همانطور که می بینید نحوه احضار تابع scanf نیز مشابه printf است. تنها تفاوت در آن است که در scanf باید لیست آدرسهای متغیرها ارسال شود. مبحث مربوط به آدرسها در فصول بعدی بررسی خواهد شد ولی در حال حاضر بخاطر بسپارید که برای بدست آوردن آدرس یک متغیر از علامت & استفاده می کنیم. بعنوان مثال &age بمعنای آدرس متغیر age است. بطور کلی در C قدیمی هرگاه که یک تابع دارای پارامترهای خروجی بود (یعنی پارامترهایی که یک مقدار را باز می گرداندند) از آدرس متغیرها استفاده می شد که امروزه این مسئله وجود ندارد.

رشته کنترلی حاوی تعداد و نوع متغیرهایی است که باید دریافت شوند و از همان مشخصه های تبدیل مربوط به printf استفاده می کند. به مثال زیر توجه کنید:

```
#include <stdio.h>
void main() {
int age;
float average ;
printf("Please enter your age and average : ");
scanf("%d %f",&age,&average);
printf("You are %d years old and your average is %5.2f
\n",age,average);
}
```

```
Please enter your age and average : 19 16.72
You are 19 years old and your average is 16.72
```

لازم به ذکر است که هنگامی که قصد وارد کردن دو عدد فوق را دارید، هم می توانید این دو عدد را با یک فاصله (فضای خالی یا tab) نوشته و سپس کلید enter را فشار دهید و یا پس از نوشتن هر عدد کلید enter را فشار دهید.

توجه کنید که معمولا رشته کنترلی فقط حاوی مشخصه های تبدیل است و هیچ متنی در آن نوشته نشده است. ازنظر منطقی نیز این کار درست است ولی چنانچه متنی در آن نوشته شود بدین معنا است که کاربر باید علاوه بر داده های مورد نظر، عین آن متن را نیز وارد کند. بعنوان مثال عبارت زیر:

```
scanf("level=%d", &a) ;
```

بدین معناست که کاربر باید ابتدا کلمه level= را عینا تایپ کرده و سپس عدد موردنظر را وارد نماید. معمولا از چنین حالاتی استفاده نمی گردد.

نکته آخر اینکه در scanf نیز می توان از مواردی همچون طول میدان برای کنترل نحوه ورود اطلاعات استفاده کرد که از حوصله بحث ما خارج است. برای اطلاعات بیشتر به Help کامپایلر خود مراجعه نمایید.

نکته مهم: توابع printf و scanf از رنگ استاندارد یعنی سفید برروی زمینه مشکی استفاده می کنند، اما می توانید برای تغییر رنگ از توابع textcolor و textbackground نیز استفاده کنید. البته دراینصورت برای دریافت و نمایش اطلاعات نیز باید از توابع cprintf و cscanf استفاده نمایید. برای توضیحات بیشتر به Help کامپایلر خود مراجعه کنید.

ورودی و خروجی اطلاعات در CPP

زبان C++ یک زبان شی گرا است، بهمین دلیل در این زبان برای ورودی و خروجی از اشیاء بجای توابع استفاده می گردد. از آنجا که امروزه معمولا برنامه نویسان C از کامپایلرهای C++ استفاده می کنند، می توانند از اشیای ورودی و خروجی آن نیز استفاده کنند. اینکار در بین بسیاری از برنامه نویسان C متداول است، بهمین دلیل ما در اینجا نحوه کار با اشیای خواندن و نوشتن در C++ را بطور مقدماتی توضیح می دهیم؛ گرچه توضیح کامل این موارد نیاز به آشنایی با شی گرایی و زبان C++ دارد. قبل از هرچیز لازم به ذکر است که کلیه اشیای مربوط به ورودی و خروجی در فایل سرآمدی بنام iostream.h تعریف شده اند، بنابراین ابتدا باید این فایل به برنامه توسط دیتور #include الحاق گردد.

زبان C++ برای نمایش اطلاعات از یک شیئ بنام cout استفاده می نماید. برای ارسال اطلاعات مورد نظر برای چاپ به cout باید از عملگر درج در جریان یا >> استفاده نماییم. بعنوان مثال :

```
cout << "Please enter your name: " ;
```

و یا مثال دیگر :

```
int a = 10;
float b = 2.86;
cout << a;
cout << b;
```

نکته جالبی که در این مثالها دیده می شود، آنستکه برخلاف تابع printf هیچ نیازی به مشخص کردن نوع متغیری که قصد چاپ آن را داریم نیست و خود شئ cout نوع آن را تشخیص می دهد. علاوه براین می توان چندین عملگر درج در جریان را با یکدیگر الحاق کرد و چندین متغیر را با یک دستور چاپ کرد. بعنوان مثال به برنامه زیر دقت کنید:

```
#include < iostream.h >
void main() {
int age = 20;
floate average = 18.23;
cout << "You are " << age << " years old and your average is "
<< average ;
}
```

```
You are 20 years old and your average is 18.230000
```

برای دریافت اطلاعات از کاربر، از شئ دیگری بنام cin استفاده می شود. برای ارسال متغیر مورد نظر به cin باید از عملگر استخراج از جریان یا << استفاده نماییم. بعنوان مثال:

```
int a;
cin >> a;
```

بازهم همانطور که می بینید نیازی به تعیین نوع متغیر موردنظر نیست و خود شئ cin نوع متغیر را بطور اتوماتیک تشخیص داده و داده ای از همان نوع را از کاربر دریافت و در متغیر مورد نظر قرار می دهد.
عملگرهای استخراج از جریان را نیز می توان با یکدیگر الحاق کرد. اکنون به یک برنامه کاملتر توجه کنید:

```
#include < iostream.h >
void main() {
int age;
float average ;
cout << "Please enter your age and average : " ;
cin >> age >> average ;
cout << "You are " << age << "years old and your average is "
<< average;
}
```

```
Please enter your age and average : 19 16.72
You are 19 years old and your average is 16.72
```

برای رفتن به خط بعد در شئ cout می توان از دستکاری کننده endl استفاده کرد. مثلا در دستور زیر پس از چاپ پیغام، مکان نما به خط بعد منتقل می شود:

```
cout << "List of students : " << endl;
```

البته دستکاری کننده های متعدد دیگری نیز ازجمله موارد مربوط به تعیین طول میدان و نحوه چاپ مقادیر وجود دارد که توضیح آنها نیاز به یک مبحث مستقل دارد.

توابع کتابخانه ای

همانطور که قبلا نیز گفته شد، زبان C از زیر برنامه ها نیز حمایت می کند. هر زیر برنامه در C یک تابع نامیده می شود که آن را بطور مفصل در فصول بعدی بررسی خواهیم کرد. تا کنون با توابعی همچون main و یا printf و scanf آشنا شده ایم.

معمولا عرضه کنندگان کامپایلرها و یا سایر فروشندگان نرم افزار، برخی از توابع عمومی را که ممکن است مورد نیاز جمع کثیری از برنامه نویسان مختلف باشد را در قالب کتابخانه ای از توابع در اختیار برنامه نویسان می گذارند (این کتابخانه ها ممکن است مجانی باشند و یا نیاز به پرداخت مبلغی داشته باشند). بعضی از این

توابع کتابخانه ای مانند printf و scanf بصورت استاندارد درآمده و توسط عرضه کنندگان مختلف ارائه می شوند.

در کامپایلر عرضه شده توسط شرکت بورلند (Borland C++ 3.1) نیز کتابخانه های متعددی از توابع برای شما عرضه شده اند که بتدریج با آنها و کاربردشان آشنا خواهید شد. نکته مهم آنستکه برای استفاده از این توابع ابتدا باید فایل سرآمد مربوط به آنها را نیز در ابتدای برنامه خود اضافه نمایید (با استفاده از #include). هر فایل سرآمد شامل تعاریف اولیه گروهی از توابع مرتبط با هم و داده های مربوط به آنها بوده و در استاندارد قدیمی تر دارای پسوند .h می باشد(در استاندارد جدید پسوند این فایلها حذف شده است). برخی از این فایلهای سرآمد عبارتند از:

stdio.h : توابع ورودی و خروجی استاندارد
math.h : توابع ریاضی
graphics.h : توابع مربوط به عملیات گرافیکی
string.h : توابع مربوط به کار با رشته ها

البته تعداد این فایلها و توابع مربوطه بسیار زیاد است و در آینده با هریک از آنها بنا به کاربرد، آشنا خواهید شد.

ساختارهای کنترلی

در فصل ششم اشاره کردیم که در برنامه نویسی ساختیافته، هر برنامه از ۳ ساختار کنترلی بنام: ساختار ترتیب، ساختار انتخاب و ساختار تکرار تشکیل می گردد. از آنجا که این ۳ ساختار، نحوه و ترتیب اجرای برنامه را کنترل می کنند، به آنها ساختارهای کنترلی گفته می شود. تا کنون فقط با برنامه هایی سروکار داشته ایم که از ساختار ترتیب استفاده می کرده اند، چرا که دستورهای زبان C در حالت عادی به همان ترتیبی که نوشته شده اند، یکی پس از دیگری اجرا می شوند.

اما زبان C دارای ۳ نوع ساختار انتخاب می باشد که عبا رتند از : ساختار if یا ساختار تک انتخابی، ساختار if / else یا ساختار دو انتخابی و ساختار switch یا ساختار چند انتخابی. علاوه براین، این زبان دارای ۳ نوع ساختار تکرار بنامهای while، for و do / while نیز می باشد که هریک را بطور کامل شرح خواهیم داد.

قرارداد: توجه کنید که در هنگام تشریح یک دستور، خود دستور با رنگ آبی و عملگرهای آن مانند () با رنگ قرمز نشان داده می شوند. قسمتهایی که در داخل <> قرار می گیرند، عبارت یا دستوری هستند که باید در هنگام استفاده جایگزین گردند.

ساختار انتخاب if

این دستور به شکل زیر استفاده می شود:

```
if (<expresion>) <statement>;
```

نحوه کار بدینصورت است که ابتدا عبارت موجود در قسمت <expression> ارزیابی می شود. در صورتیکه درست ارزیابی گردد، دستور قسمت <statement> اجرا خواهد شد و در صورتیکه نادرست باشد، بدون اینکه دستور قسمت <statement> را اجرا کند به دستور بعدی خواهد رفت. این دستور می تواند بصورت زیر نیز استفاده گردد:

```
if (<expresion>) <statement 1>;
else <statement 2>;
```

در اینصورت ابتدا عبارت موجود در قسمت <expression> ارزیابی می شود. در صورتیکه درست ارزیابی گردد، دستور قسمت <statement 1> اجرا خواهد شد، و در صورتیکه نادرست باشد، دستور قسمت <statement 2> اجرا خواهد شد. در هر حال فقط یکی از این دو قسمت اجرا خواهد گردید.

دقت کنید که پرانتز استفاده شده پس از دستور if برخلاف برخی زبانهای دیگر، اجباری است. علاوه براین، در این دستور نیازی به استفاده از then نمی باشد .

بعنوان مثال چنانچه متغیر grade حاوی نمره دانشجو باشد و بخواهیم بر مبنای نمره وی، پیغام مناسبی چاپ کنیم، می توانیم از دستور زیر استفاده کنیم:

```
if (grade> = 10) printf("Passed !");
else printf("Failed!");
```

در حالت عادی دستور if منتظر یک دستور در بدنه خود می باشد، اما چنانچه می خواهید چندین دستور را در بدنه یک دستور if دهید، باید آنها را در داخل آکولاد باز وبسته { } قرار دهید. این مجموع دستورات را یک دستور مرکب می گویند. بطور کلی در زبان C هرجا که می توان یک دستور قرار داد، می توان از یک دستور

مرکب نیز استفاده کرد. به یک دستور مرکب، بلوک نیز گفته می شود. بنابراین صورت کلی دستور if به شکل زیر است:

```
if (<expression>) {
<statement 1> ;
<statement 2> ;
....
<statement n> ;
}
else {
<statement 1> ;
<statement 2> ;
.....
<statement m> ;
}
```

توجه کنید که وجود قسمت else اختیاری است و در ضمن ممکن است یکی از دو قسمت دارای دستور ساده و دیگری دارای دستور مرکب باشد. بعنوان یک مثال کاملتر به برنامه زیر توجه کنید:

برنامه ۱) برنامه ای بنویسید که ضرایب یک معادله درجه ۲ را دریافت و ریشه های آن را محاسبه و چاپ نماید.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
void main() {
int a, b, c;
float x1, x2, delta;
clrscr();
printf("Please enter a, b and c : ");
scanf("%d %d %d", &a, &b, &c);
if (a==0) {
printf("wrong equation!");
exit(1) ;
}
```

```
delta = b*b – 4*a*c;
if (delta <0)
printf("No answer !");
else if (delta == 0) {
x1 = -b / (2*a);
printf("There is one answer, x = %f",x1);
}
else {
delta = sqrt(delta);
x1 = (-b+delta) / (2*a);
x2 = (-b-delta) / (2*a);
printf("There are two answers, x1= %4.2f and x2= %4.2f", x1,
x2);
}
}
Please enter a, b and c : 3 -7 2
There is two answers, x1 = 2.00 and x2 = 0.33
```

چندین نکته درمورد برنامه بالا قابل ذکر است.

۱- در این برنامه از ۲ تابع جدید استفاده شده است.اولی تابع sqrt که یک عدد را به عنوان ورودی دریافت و جذر آن را باز می گرداند. این تابع در فایل math.h تعریف شده است. و دیگری تابع exit که باعث می شود اجرای برنامه خاتمه یابد. این تابع نیز در فایل stdlib.h تعریف شده است.

۲- به نحوه دندانه گذاری در برنامه دقت کنید، هرجا که بلوک جدیدی ایجاد شده است، دستورات آن حدود ۳ کاراکتر جلوتر نوشته شده اند. اینکار باعث می شود که خوانایی برنامه افزایش یابد.

۳- توجه کنید که همانطور که قبلا نیز گفته شد، خروجی یک عبارت مقایسه ای، یک عدد است که ۰ نشانه نادرست و هر عدد دیگر نشانه درست است. بنابراین در قسمت شرط یک دستور if می توان بجای یک عبارت مقایسه ای، هر عبارت دیگری که یک مقدار عددی بازگرداند نیز قرار داد! بعنوان مثال می توان بجای دستور

```
if (a==0) ...
```

از دستور زیر استفاده کرد :

```
if (!a) ...
```

دراینصورت چنانچه a برابر صفر باشد، نادرست تلقی خواهد شد و در نتیجه نقیض آن یعنی ! ، درست محسوب می گردد.
یک روش متداول استفاده از دستور if، استفاده از if های تودرتو می باشد که در مثال بالا نیز دیده می شود. دراینحالت مجموعه ای از عبارات if-else-if بصورت متداخل قرار داده می شوند. بعنوان نمونه به مثال زیر توجه کنید:

```
if (grade> = 18) printf("good!");
else if (grade> = 15) printf("medium!");
else if (grade> = 12) printf("rather weak!");
else if (grade> = 10) printf("weak");
else printf("failed!");
```

در چنین دستوری، کلیه شرطها بترتیب از بالا به پایین بررسی شده و به محض اینکه یکی از آنها درست باشد، دستور مربوط به آن اجرا شده و از بقیه دستورات صرفنظر می گردد. درصورتیکه هیچ یک از شرطها درست نباشد، دستور مربوط به آخرین else اجرا می گردد. درچنین حالتی توصیه می گردد که شرطهای نادر را که امکان وقوع آنها کم است، در انتهای کار بررسی نمایید، تا تعداد مقایسه کمتری صورت پذیرد.

مشکلی که در مورد if های تودرتو پیش می آید، مسئله تعیین if مربوط به هر else است. بعنوان مثال در مورد دستور زیر، else به کدام if تعلق دارد؟

```
if (a <b)
if (c <d) <statement1>;
else <statement2>;
```

همانطور که از دندانه گذاری نیز مشخص است، else مربوط به دومین if می باشد. یعنی statement > 2> درصورتی اجرا خواهد شد که b>a درست باشد، اما d>c درست نباشد. بطور کلی طبق قوانین گرامری، هر else مربوط به نزدیکترین if قبل از خود می باشد. اما سوال این است که اگر بخواهیم else به if اول بازگردد از چه روشی استفاده نماییم. دراینصورت می توان از یکی از دو روش زیر استفاده کرد:

```
if (a < b) if (a <b) {
if (c <d) <statement 1> ; if (c <d) <statement1>;
else ; }
else <statement2>; else <statement2>;
```

بعنوان یک مثال دیگر، به نمونه زیر دقت کنید:

```
if (<c1>) {
if (<c2>)
if (<c3>) <statement2>;
else <statement 3>; // this refer to if (<c3>)
}
else <statement 4>;//this refer to if <c1>
```

برنامه ۲) برنامه ای بنویسید که ۳ عدد را دریافت و حداکثر آنها را چاپ کند.

```
#include <stdio.h>
void main() {
int a, b, c, max;
printf("Please enter 3 numbers :");
scanf("%d %d %d",&a, &b, &c);
if (a> b)
if (a> c) max = a;
else max= c;
else if (b> c) max = b;
else max = c;
printf("Maximum is %d",max);
}
```

ساختار تکرار while

همانطور که در بخش الگوریتمها نیز گفته شد، یک ساختار تکرار باعث می شود تا زمانیکه شرط خاصی برقرار است، عملیات مشخصی تکرار گردد. دستور while نیز باعث ایجاد یک حلقه تکرار به شکل زیر می گردد:

```
while (<expression>) <statement>;
```

این دستور باعث می شود تا زمانیکه شرط موجود در قسمت <expression> درست است، دستور قسمت <statement> تکرار شود، و به محض اینکه شرط نادرست گردد، کنترل اجرا به دستور بعد از حلقه می رود .

بازهم دستور موجود در قسمت <statement> می تواند یک دستور مرکب باشد، دراینصورت دستور بصورت زیر درخواهد آمد:

```
while (<expression>) {
<statement1>;
<statement2>;
.....
<statementn>;
}
```

برنامه ۳) برنامه ای بنویسید که یک عدد را دریافت و فاکتوریال آن را محاسبه و چاپ نماید.

```
#include <stdio.h>
void main() {
int i,number;
long int factorial;
printf("Please enter number :");
scanf("%d",&number);
factorial = 1;
i = 1;
while (i <= number) {
factorial *= i;
i ++;
}
printf("Factorial of %d is %ld"number,factorial);
}
```

برنامه ۴) برنامه ای بنویسید که یک متن را از کاربر دریافت و آن را با حروف بزرگ چاپ کند.

```
#include <stdio.h>
void main() {
char ch;
ch = getch() ;
while (ch != 13) {
if (ch>= 'a' && ch <= 'z')
ch -= 32;
putch(ch);
ch = getch();
```

```
}
}
```

ساختار تکرار for

همانگونه که در مثال مربوط به حل مسئله فاکتوریال دیده می شود، گاهی نیاز به حلقه تکراری داریم که به تعداد دفعات مشخصی تکرار گردد. در چنین مواقعی با استفاده از یک متغیر شمارنده، تعداد تکرارها را تا رسیدن به مقدار مورد نظر می شماریم و سپس به حلقه پایان می دهیم. به چنین حلقه هایی، تکرار تحت کنترل شمارنده یا تکرار معین می گوییم، چرا که تعداد تکرار ها از قبل مشخص است. چنین حلقه ای دارای ۳ جزء اصلی می باشد:

۱- مقداردهی اولیه به متغیر شمارنده حلقه
۲- شرط پایان حلقه (پایان شمارش)
۳- نحوه افزایش متغیر شمارنده

از آنجا که در تمام حلقه هایی که تکرار معین دارند، همین ساختار استفاده می شود؛ در اکثر زبانهای برنامه سازی یک ساختار تکرار ویژه، بنام حلقه for، برای اینکار در نظر گرفته شده است. اما در این بین، حلقه تکرار for در زبان C دارای ویژگیهای خاصی است که آنرا از سایر زبانها متمایز کرده و به آن قدرت بسیار بالایی داده است. شکل کلی این دستور بصورت زیر است:

```
for (<exp1> ; <exp2> ; <exp3> ) <statement>;
```

وظایف عبارات فوق بشرح زیر است :

۱- <exp1>، مقداردهی اولیه به متغیر حلقه

۲- <exp2>، شرط اجرای حلقه

۳- <exp3>،نحوه افزایش متغیر حلقه

البته همانندموارد قبل بازهم قسمت <statement> می تواند یک دستور مرکب باشد. نحوه کار حلقه بدینصورت است که در ابتدای شروع حلقه <exp1> فقط برای یکبار اجرا می شود. سپس عبارت <exp2> بررسی می گردد و در صورتیکه درست ارزیابی شود (≠ ۰)، آنگاه دستور <statement> اجرا شده و سپس به ابتدای حلقه باز می گردد. از اجرای دوم به بعد، ابتدا عبارت <exp3> اجرا می گردد (یعنی متغیر حلقه افزایش می یابد) و سپس عبارت <exp2> بررسی می گردد و مجددا درصورت درست بودن،

حلقه تکرار می شود. اینکار تا زمانیکه مقدار عبارت <exp2> نادرست (= ۰) ارزیابی شود، تکرار می گردد. به محض اینکه این اتفاق بیفتد، کنترل اجرا به دستور پس از حلقه انتقال می یابد.
درحقیقت هر حلقه for معادل با حلقه while زیر است:

```
<exp1> ;
while (<exp2>) {
<statement>;
<exp3>;
}
```

بعنوان یک مثال ساده، تکه برنامه زیر اعداد بین ۰ تا ۱۰۰ را چاپ می نماید:

```
int count;
for (count = 0; count <= 100; count ++)
printf("%d ",count);
```

اگر بخواهیم تنها مضارب ۵ چاپ شوند، حلقه را به شکل زیر تغییر می دهیم:

```
for (count = 0; count <= 100; count += 5)
```

حتی می توان مضارب ۵ را از آخر به اول چاپ کرد:

```
for (count = 100; count> = 0; count -= 5)
```

قسمت شرط می تواند یک شرط مرکب نیز باشد، بعنوان مثال :

```
for (count = 0; count <100 && sw==1; count ++)
```

که در اینصورت در هربار اجرای حلقه، علاوه بر مقدار شمارنده، مقدار متغیر SW نیز بررسی می گردد.

نکته آخر اینکه قسمت مقدار دهی اولیه و افزایش متغیر نیز می توانند شامل چند عبارت باشند که در اینصورت با کاما از یکدیگر جدا می شوند. بعنوان مثال:

```
for (a = 0, b = 100; b – a> 50; a++, b--)
```

در ادامه یک مثال کاملتر برای تشریح این حلقه آمده است:

برنامه ۵) برنامه ای بنویسید که تعدادی عدد را از کاربر دریافت و ۲ عدد بزرگتر و مجموع آنها را محاسبه و چاپ نماید.

```
#include <stdio.h>
void main() {
int i, n, number;
int sum, max1, max2;
printf("please enter n : ");
scanf("%d",&n);
sum = 0;
max1 = max2 = -1;
for (i=1 ; i<n ; i++) {
printf("enter number : ");
scanf("%d",&number);
sum += number;
if (number> max1) {
max2 = max1;
max1 = number;
}
else if (number> max2)
max2 = number;
} //end for
printf("Sum = %d, Maximum 1=%d, Maximum 2= d", sum, max1,
max2);
}
```

نکته جالب در مورد حلقه for آنستکه می توان هریک از ۳ عبارت فوق را حذف کرد. به مثالهای زیر توجه کنید:

```
for (;i<100; i++)
```

مقداردهی اولیه حذف شده است. این در صورتی است که بنحوی قبل از شروع حلقه، مقدار متغیر i تعیین شده باشد.

```
for (i= 0;i<100;)
```

نحوه افزایش متغیر حلقه حذف شده است. این مورد زمانی بکار می رود که نحوه افزایش متغیر حلقه پیچیده بوده و توسط برنامه نویس در داخل حلقه انجام شود.

```
for (;i<100;)
```

هر دو مورد فوق حذف شده است.

```
for (i=0; ;i++)
```

شرط ادامه حلقه حذف شده است. این مورد زیاد متداول نیست و ممکن است باعث شود حلقه برای همیشه اجرا شود. در چنین مواردی حتما باید در داخل حلقه با استفاده از دستور break (که در قسمتهای بعدی توضیح داده خواهد شد)، راهی برای خروج از حلقه قرار داده شود.

حلقه do / while

همانگونه که در قسمت الگوریتمها توضیح داده شد، در بعضی مواقع لازم است که شرط ، در انتهای حلقه بررسی گردد. دستور do / while از نوع حلقه هایی است که ابتدا دستورات را اجرا کرده و سپس شرط ادامه حلقه را بررسی می نماید. شکل کلی این دستور بصورت زیر است:

```
do
<statement> ;
while (<expression>) ;
```

نحوه کار این حلقه به این صورت است که ابتدا دستور <statement> اجرا می گردد، سپس شرط حلقه بررسی شده و درصورتیکه درست بود، به ابتدای حلقه باز گشته و آن را مجددا اجرا می کند. البته دستور <statement> می تواند یک دستور مرکب باشد.

بعنوان یک مثال کوچک، ممکن است شما از کاربر خواسته اید که اعلام کند آیا مایل به ادامه هست یا خیر؟ وی باید پاسخ y یا n بدهد، اما ممکن است یک حرف اشتباه (مانند m) وارد کند. قصد داریم تکه برنامه ای بنویسیم که عمل دریافت پاسخ را تا زمانیکه یک حرف درست وارد شود، تکرار کند. مسلم است که باید ابتدا یک پاسخ وارد شود و سپس درستی آن بررسی گردد.

```
char answer;
do {
printf("Do you want to continue (y/n) ?");
```

```
answer = getch();
} while (answer != 'y' && answer != 'n') ;
```

بعنوان یک نمونه کاملتر به مثال زیر توجه کنید.

برنامه ۶) فرض کنید نمرات یک گروه از دانشجویان بصورت درجه بندی (A, B, C and D) آماده شده است. برنامه ای بنویسید که نمرات دانشجویان را دریافت و در پایان درصد هریک از نمرات را محاسبه و چاپ نماید. در ضمن از آنجا که تعداد دانشجویان از قبل مشخص نیست، کاربر در انتهای نمرات، حرف Q (مخفف Quit) را وارد می نماید.

```
#include <stdio.h>
void main() {
int aCount, bCount, cCount, dCount, n;
char grade;
aCount = bCount = cCount = dCount = n = 0;
do {
printf("Enter grade (Q for Quit) : ");
grade = getch() ;
n ++;
if (grade == 'A') aCount ++;
else if (grade == 'B') bCount ++;
else if (grade == 'C') cCount ++;
else if (grade == 'D') dCount ++;
else if (grade == 'Q') n --;
else {
printf("Wrong grade, try again.\n");
n --;
}
} while (grade != 'Q' ) ;
printf("Statistics :\n");
printf("Grade A : %f percent\n", float(aCount)/float(n));
printf("Grade B : %f percent\n", float(bCount)/float(n));
printf("Grade C : %f percent\n", float(cCount)/float(n));
printf("Grade D : %f percent\n", float(dCount)/float(n));
```

```
} // end main
```

ساختار switch / case

اگر مجددا به برنامه ۶ دقت کنید، خواهید دید که در بعضی موارد قصد داریم برحسب مقادیر مختلف یک عبارت (در اینجا مقدار متغیر grade)، عملیات متفاوتی را انجام دهیم. گرچه اینکار با استفاده از دستورات if / else متداخل قابل انجام است، اما ساختار مناسبتری نیز برای اینکار وجود دارد، که به آن ساختار چندانتخابی می گوییم. شکل کلی ان ساختار بصورت زیر است:

```
switch (<expression>) {
case <exp1> : <statement 1> ;
<statement 2> ;
...
<statementn>;
case <exp2> : <statement 1> ;
<statement 2> ;
...
<statementn>;
...
default : <statement 1> ;
<statement 2> ;
...
<statementn>;
}
```

برنامه ۷) برنامه ۶ را با استفاده از دستور switch / case بازنویسی نمایید. برنامه را بگونه ای بنویسید که حروف بزرگ و کوچک هردو مورد قبول واقع شود.

```
#include <stdio.h>
void main() {
int aCount, bCount, cCount, dCount, n;
char grade;
aCount = bCount = cCount = dCount = n = 0;
do {
printf("Enter grade (Q for Quit) : ");
grade = getch() ;
```

```
n ++;
switch (grade) {
case 'A' :
case 'a' : aCount ++; break ;
case 'B' :
case 'b' : bCount ++; break ;
case 'C' :
case 'c' : cCount ++; break ;
case 'D' :
case 'd' : dCount ++; break ;
case 'Q' :
case 'q' : n--; break ;
default : printf("Wrong grade, try again.\n");
n --;
} //end switch
} while (grade != 'Q' ) ;
printf("Statistics :\n");
printf("Grade A : %f percent\n", float(aCount)/float(n));
printf("Grade B : %f percent\n", float(bCount)/float(n));
printf("Grade C : %f percent\n", float(cCount)/float(n));
printf("Grade D : %f percent\n", float(dCount)/float(n));
} // end main
```

برنامه ۸) برنامه ای بنویسید که یک عدد، یک عملگر و یک عدد دیگر را از کاربر دریافت و پس از اعمال عملگر برروی دو عدد، حاصل را چاپ نماید.

```
#include <stdio.h>
void main() {
int number2, number2, result;
char op ;
printf("Please enter number1 operator number2 : ");
scanf("%d %c %d",&number1, &number2, &op, &number3);
result = 0;
switch (op) {
case '+' : result = number1 + number2 ; break;
case '-' : result = number1 - number2 ; break;
```

```
case '*' : result = number1 * number2 ; break;
case '/' : if (number2 != 0) result = number1 / number2 ;
else printf("There is no answer!\n");
break;
case '%' : if (number2 != 0) result = number1 % number2 ;
else printf("There is no answer!\n");
break;
default : printf("invalid operator!\n");
}
printf("Result = %d",&result);
}
```

دستورات **break** و **continue**

این دستورات قادرند مسیر اجرای برنامه را در یک حلقه تکرار تغییر دهند. البته این تغییر مسیر بصورت کنترل شده بوده و همانند دستور goto نمی توان به هرجای دلخواه پرش کرد.

چنانچه دستور break در یک ساختار while، for، do/while ویا switch بکار رود، باعث می شود که بلافاصله کنترل اجرای برنامه از ساختار خارج شده و به اولین دستور پس از ساختار برود. قبلا کاربرد این دستور را در switch دیده اید. در اینجا به مثال زیر دقت کنید:

برنامه ۹) برنامه ۵ را بگونه ای تغییر دهید که فقط اعداد مثبت را بپذیرد، و درصورتیکه عدد منفی وارد شد، بلافاصله به عملیات خاتمه داده و نتایج تا همین نقطه را چاپ نماید.

```
#include <stdio.h>
void main() {
int i, n, number;
int sum, max1, max2;
printf("please enter n : ");
scanf("%d",&n);
sum = 0;
max1 = max2 = -1;
for (i=1 ; i<n ; i++) {
printf("enter number : ");
scanf("%d",&number);
if (number <0) break; // this is the difference
```

```
sum += number;
if (number> max1) {
max2 = max1;
max1 = number;
}
else if (number> max2)
max2 = number;
} //end for
printf("Sum = %d, Maximum 1=%d, Maximum 2= d", sum, max1,
max2);
}
```

اما دستور continue فقط در حلقه های while، for و do/while بکار می رود. نحوه عمل آن بدین صورت است که به محض آنکه کنترل اجرا به این دستور برسد، بلافاصله از باقیمانده حلقه صرفنظر کرده و مجددا به ابتدای حلقه باز می گردد و اجرای آن را از سر می گیرد. در مورد حلقه for، پس از بازگشت به ابتدای حلقه، عمل افزایش مقدار متغیر حلقه نیز صورت می پذیرد.

بعنوان مثال، چنانچه بخواهیم برنامه ۹ را بگونه ای تغییر دهیم که از اعداد منفی صرفنظر کند و آنها را در محاسبات لحاظ نکند، کافیست دستور

```
if (number <0) break;
```

را به دستور زیر تبدیل کنیم :

```
if (number <0) continue;
```

دراینصورت، چنانچه عدد منفی باشد، بدون اینکه محاسبات بعدی انجام شوند، کنترل به ابتدای حلقه بازگشته و عدد بعدی را دریافت می کند.

آرایه در C عبارتست از مجموعه ای از داده های همنوع که تحت یک نام مشترک و در خانه های متوالی حافظه ذخیره می گردند. برای دسترسی به عناصر آرایه، باید از نام آرایه بعلاوه اندیس استفاده کرد. در قسمتهای بعدی، نحوه تعریف و استفاده از آرایه ها را تشریح خواهیم کرد.

آرایه های یک بعدی

پیش از آنکه بتوان از یک آرایه یک بعدی استفاده کرد، باید آن را اعلان کرد. اعلان آرایه ها بصورت زیر انجام می گردد:

```
<type> <var-name>[<size>] ;
```

بعنوان مثال:

```
int A[10];
```

خط بالا یک آرایه ۱۰ تایی از اعداد صحیح بنام A ایجاد می نماید. هر کدام از عناصر این آرایه می توانند بعنوان یک متغیر مستقل مورد استفاده قرار گیرد. برای دسترسی به عناصر این آرایه باید از اندیس استفاده نمود. در زبان C اندیسها در داخل کروشه [] قرار می گیرند. نکته بسیار مهمی که باید بدان توجه کرد آنستکه در C اندیس یک عدد صحیح است که از ۰ آغاز می گردد. به مثال زیر توجه نمایید:

```
int A[10] ;
A[2] = 8;
```

و یا چنانچه بخواهیم مقدار خانه سوم را بر ۲ تقسیم و در متغیر x بریزیم، داریم:

```
x = A[3] / 2;
```

اکنون به یک مثال دقت کنید.

برنامه ۱) برنامه ای بنویسید که شماره دانشجویی و معدل تعدادی دانشجو را دریافت، و سپس چنانچه معدل دانشجو از میانگین کلاس :

- بیش از یک نمره بیشتر باشد، چاپ کند : عالی
- حداکثر یک نمره بیشتر یا کمتر باشد، چاپ کند : خوب
- بیش از یک نمره کمتر باشد، چاپ کند : ضعیف

```
#include <stdio.h>
void main() {
float average[100] ;
long int id[100] ;
int i, n ;
float totalAverage;
printf("enter number of students : ");
```

```
scanf("%d", &n);
for (i = 0; i < n ; i ++) {
printf("enter id and average : ");
scanf("%ld %f", &id[i], &average[i]);
totalAverage += average[i];
}
totalAverage /= n;
for (i = 0; i < n ; i ++) {
if (average[i] > = totalAverage + 1)
printf("%ld : excellent !\n", id[i]);
else if (average[i] > = totalAverage – 1)
printf("%ld : good !\n", id[i]);
else printf("%ld : weak !\n", id[i]);
}
}
```

چند نکته مهم راجع به آرایه در C وجود دارد که حتما باید به آنها دقت کنید:

۱- اندازه آرایه ها در C ثابت بوده و حتما باید توسط یک مقدار ثابت صحیح تعیین گردد. بعنوان مثال اعلان زیر خطای نحوی محسوب می گردد:

```
int n ;
n=100 ;
int A[n];
```

اما می توان با استفاده از متغیر های ثابت (ثابتهای دارای نام)، اندازه آرایه را تعیین کرد، که در قسمتهای بعدی به آن اشاره خواهد شد.

۲- اندیس آرایه ها در C عدد صحیح بوده و همیشه از ۰ شروع می شود. لذا به تفاوت "عنصر چهار آرایه" یعنی A[4] و "چهارمین عنصر آرایه" یعنی A[3] دقت کنید. این مسئله معمولا باعث بروز خطاهای منطقی می گردد.

۳- در C مرز آرایه ها بررسی نمی گردد. بدین معنا که چنانچه اندیسی خارج از محدوده مجاز یک آرایه استفاده شود، باعث ایجاد خطا توسط کامپایلر نمی گردد، اما مسلما برنامه را دچار یک خطای منطقی خواهد کرد. بعنوان مثال:

```
int A[10] ;
A[12] = 20 ; //this is not a syntax error but a logical error
```

لذا بررسی مرزهای آرایه بعهده خود برنامه نویس است و باید از درستی برنامه خود و خارج نشدن از محدوده مجاز مطمئن گردد.

۴- مقداردهی اولیه به آرایه های یک بعدی بصورت زیر انجام می پذیرد:

```
int A[3] = {5, 2, 8};
```

که در اینجا A[0] برابر ۵ ، A[1] برابر ۲ و A[2] برابر ۸ خواهد شد.
علاوه براین می توان فقط به تعدادی از عناصر آرایه مقدار داد، دراینصورت مقدار عناصر باقیمانده آرایه اتوماتیک ۰ خواهد شد.

```
int B[10] = {5, 8} ;
```

در اینجا عناصر B[2] به بعد مقدار ۰ خواهند گرفت. بنابراین می توان برای ۰ کردن کلیه عناصر یک آرایه به شکل زیر عمل کرد :

```
int C[10] = {0};
```

چنانچه به آرایه مقدار دهی اولیه کرده باشیم، می توان تعداد عناصر آرایه را نیز ذکر نکرد، دراینصورت اندازه آرایه بطور اتوماتیک برابر تعداد مقادیر مشخص شده خواهد شد.

```
int C[] = {10, 15, 20};
```

در مثال فوق آرایه C با ۳ عضو درنظر گرفته می شود.

آرایه ها در برنامه نویسی C (متغیرهای ثابت)

متغیرهای ثابت

همانطور که در قسمت قبل گفته شد، گرچه اندازه یک آرایه باید ثابت صحیح باشد؛ اما می توان از متغیرهای ثابت نیز استفاده کرد. یک متغیر ثابت، متغیری است که فقط می تواند در هنگام اعلان مقدار اولیه بگیرد و این مقدار دیگر قابل تغییر نیست.

برای اعلان متغیرهای ثابت، از کلمه کلیدی const قبل از نوع متغیر استفاده می گردد. بعنوان مثال:

```
const int k = 10;
```

اکنون هرگونه تلاش برای تغییر مقدار k، باعث ایجاد یک خطای نحوی توسط کامپایلر خواهد شد. به این نوع متغیرها، ثابتهای نام دار نیز گفته می شود.

این متغیرها در تعریف مقادیر ثابتی که مقدار آنها در طول برنامه تغییر نمی کند، بکار می روند. بعنوان مثال :

```
const float pi = 3.14;
```

این کار نه تنها خوانایی برنامه را بالا می برد (بدلیل استفاده از کلمه pi که برای همه شناخته شده است)، بلکه باعث می شود تغییر پذیری برنامه نیز بالا برود. بدین معنا که در صورتیکه برنامه نویس تصمیم گرفت مقدار ثابت را عوض کند، نیازی به تغییر کل برنامه نیست و فقط کافی است مقدار اولیه متغیر را عوض نماید. بعنوان مثال اگر برنامه نویس بخواهد عدد pi را با ۴ رقم اعشار در محاسبات شرکت دهد، فقط باید در تعریف اولیه آن، مقدار را عوض کرده و از ۴ رقم اعشار استفاده نماید.

از این مسئله می توان در تعریف آرایه ها نیز استفاده کرد. بدین صورت که بجای آنکه اندازه آرایه را با یک ثابت صحیح مشخص نماییم، آن را با یک متغیر ثابت تعریف می کنیم. با اینکار، درصورتیکه نیازی به تغییر اندازه آرایه (یا آرایه ها) گردد، فقط کافی است مقدار اولیه متغیر ثابت خود را تغییر دهیم. برای نمونه به مثال زیر دقت کنید:

برنامه ۲) برنامه ای بنویسید که سال ورود تعدادی دانشجو را دریافت و سپس تعداد ورودی های سالهای ۷۵ تا ۸۴ را محاسبه و چاپ نماید.

```
void main() {
const int startYear = 75;
const int yearNo = 10;
int count[yearNo] = {0};
int i, n, year;
printf("enter student no :");
```

```
scanf("%d",&n);
for (i= 0;i<n; i++) {
printf("enter entrance year :");
scanf("%d",&year);
count [year – startYear] ++;
}
for (i= 0; i<yearNo ; i++) {
printf("year = %d count = %d \n",startYear + i , count[i]);
}
```

آرایه ها می توانند دارای ابعاد بیشتری نیز باشند. در زبان C نیز می توان یک آرایه چند بعدی را بصورت زیر اعلان کرد:

```
<type> <var-name> [<size 1>][<size 2>]  ...  [<size n>] ;
```

بعنوان مثال، اعلان زیر یک آرایه دوبعدی را معرفی می نماید:

```
int A[5][8] ;
```

برای دسترسی به هر عنصر از این آرایه باید از دو علامت [] استفاده کرد.

توجه کنید که اندیس سطرها و ستونها هر دو از ۰ آغاز می گردند. بعنوان مثال:

```
int A[5][8] ;
A[3][1] = 24;
```

البته در مورد آرایه های با ابعاد بالاتر نیز به شکل مشابهی عمل می گردد. بعنوان مثال به نحوه استفاده از یک آرایه سه بعدی در مثال زیر دقت کنید:

```
int B[5][8][6] ;
B[2][4][0] = 12;
```

برنامه ۳) برنامه ای بنویسید که نمرات تعدادی دانشجو را برای ۵ درس دریافت و آنها را بهمراه معدل دانشجو ذخیره نماید، سپس برای هر دانشجو معدل وی را چاپ نماید.

```
void main() {
const int maxStudent = 100;
float grades[maxStudent][5] ;
float average[maxStudent];
int i, j, n;
printf("enter student number:");
scanf("%d",&n);
for (i=0 ; i< n ; i++) {
printf("student no %d:\n",i+1);
sum = 0;
for (j= 0 ;j<5 ; j++) {
printf("enter grade : ");
scanf("%f", &grades[i][j]);
sum += grades[i][j] ;
}
average[i] = sum / 5 ;
}
for (i=0 ; i< n ; i++)
printf("average of student %d is %f \n",i+1, average[i]);
}
```

و نکته آخر اینکه مقداردهی اولیه به آرایه های چندبعدی امکان پذیر است و بصورت زیر انجام می پذیرد:

```
int A[3][4] = { {12, 5, 3, 8} , {-3, 7, -9, 2}, {4, 22, 18, 6} };
```

یعنی یک علامت {} برای کل مقداردهی قرار می گیرد، سپس هر ردیف از آرایه در داخل یک {} مجزا قرار می گیرد. برای ابعاد بالاتر نیز به روش مشابهی عمل می گردد. بعنوان مثال برای آرایه های سه بعدی داریم:

```
int A[2][3][4] = { { {12, 5, 3, 8} , {-3, 7, -9, 2}, {4, 22, 18, 6} } , {
{8, 1, -3, 4} , {-2, 8, 11, 21} , {7, 3, -15, -8} } };
```

آرایه ها را نیز همچون سایر نوع داده ها می توان به یک تابع ارسال کرد. برای اینکار ابتدا باید تابع را بگونه ای تعریف کنیم که یک پارامتر از نوع آرایه را دریافت کند. فرض کنید تابعی بنام sumArray داریم که یک آرایه یک بعدی از اعداد صحیح را بعنوان ورودی دریافت می نماید و مجموع عناصر آن را باز می گرداند.

تعریف این تابع بصورت زیر است:

```
int sumArray(int A[], int size) {
int i , sum = 0;
for (i=0; i< size; i++)
sum += A[i];
return(sum) ;
}
```

در تابع فوق، پارامتر A بعنوان یک آرایه از اعداد صحیح معرفی شده است. همانطور که می بینید، اندازه آرایه مشخص نشده است و این یک نکته مثبت است؛ چرا که تابع sumArray می تواند هر آرایه صحیحی را با هر اندازه ای دریافت نماید. درواقع حتی اگر اندازه آرایه را نیز مشخص نمایید، کامپایلر از آن صرفنظر خواهد کرد. دومین پارامتر، اندازه واقعی آرایه A را مشخص می نماید. معمولا توابع بگونه ای نوشته می شوند که هنگام ارسال یک آرایه به یک تابع، اندازه آن نیز بعنوان یک پارامتر ارسال گردد. درغیراینصورت مجبوریم در تابع اندازه مشخصی را برای آرایه در نظر بگیریم که باعث ایجاد محدودیت در ارسال آرایه های با اندازه دلخواه می گردد.

در هنگام فراخوانی تابع sumArray، برای ارسال آرایه موردنظر کافی است که تنها نام آرایه را بدون کروشه استفاده نماییم. البته اندازه واقعی آرایه نیز باید بعنوان دومین آرگومان به تابع ارسال شود.

```
void main() {
int data1[3] = {5, 10, 15};
int data2[5] = {1, 6, 4, 12, 5} ;
int sum1, sum2;
sum1 = sumArray(data1, 3);
sum2 = sumArray(data2, 5);
printf("sum1 = %d\n",sum1);
printf("sum2 = %d\n",sum2);
}
```

```
sum1 = 30
sum2 = 28
```

همانطور که در مثال فوق دیده می شود، تابع sumArray دوبار فراخوانی شده است. در بار اول یک آرایه با اندازه ۳، و در دفعه دوم یک آرایه با اندازه ۵ به آن ارسال شده است و تابع در هر دو مورد بدون هیچ مشکلی مجموع عناصر آرایه را باز گردندانده است.

نکته بسیار مهم، نحوه ارسال آرایه ها به توابع است. زبانC آرایه ها را توسط ارجاع به تابع ارسال می نماید (برخلاف انواع دیگر داده ها که درحالت عادی توسط مقدار به توابع ارسال می شدند). بدین معنا که در هنگام ارسال یک آرایه به تابع، بجای یک کپی از آرایه، خود آرایه ارسال می شود. در حقیقت در فصلهای بعدی خواهید دید که برای ارسال یک آرایه، آدرس اولین عنصر آن ارسال می گردد. لذا تابع می تواند از طریق این آدرس، به کلیه داده های آرایه اصلی دسترسی پیدا کند. اما چرا C در مورد آرایه ها به روش متفاوتی عمل می نماید؟ دلیل این مسئله آن است که معمولا یک آرایه حافظه بسیار زیادی را اشغال می کند، لذا تهیه یک کپی کردن از آن، نه تنها باعث اشغال حافظه می شود بلکه زمان زیادی را نیز صرف خواهد کرد. با ارسال آرایه ها توسط ارجاع در زمان و حافظه صرفه جویی زیادی صورت می گیرد.

اما آیا می توان یک آرایه را توسط مقدار به یک تابع ارسال کرد؟ متاسفانه خیر. اما اگر نگران تغییر سهوی آرایه ارسالی به یک تابع هستید می توانید آن را بگونه ای به تابع ارسال نمایید که تغییر آن در تابع ممکن نباشد. زبان C یک نحوه دیگر ارسال داده ها به توابع بنام ارسال توسط ارجاع ثابت می باشد. چنانچه در هنگام تعریف یک پارامتر از یک تابع، از کلمه کلیدی const استفاده شود، کامپایلر اجازه تغییر مقادیر آن پارامتر را در حین اجرای تابع نخواهد داد. با این ارسال آرایه ها بصورت ارجاع ثابت، می توانیم مانع از انجام تغییرات ناخواسته در آرایه شویم. مثال زیر نحوه انجام این کار را نشان می دهد.

برنامه) برنامه ای بنویسید که با استفاده از یک تابع، اشتراک دو مجموعه را محاسبه و چاپ نماید.

```
void intersection(const int A[], int na, const int B[], int nb, int C[],
int &nc) {
k = 0;
for (i=0; i< na; i++) {
sw = 1;
for (j=0; j< nb && sw; j++)
if (A[i] == B[j]) {
C[k] = A[i] ;
k ++;
```

```
sw = 0;
}
}
nc = k;
}
void printSet(int set[], int size) {
int i;
printf("{ ") ;
for (i=0; i< size; i++)
printf("%d ",set[i]) ;
printf("}\n");
}

void main() {
int set1[5] = {5, 8, 3, 12, 20};
int set2[3] = {12, 16, 8} ;
int result[3] , resultSize ;
intersection(set1, 5, set2, 3, result, resultSize);
printf("set 1 = ");
printSet(set1) ;
printf("set 2 = ");
printSet(set2) ;
printf("intersection = ");
printSet(result) ;
}

set1 = { 5 8 3 12 20 }
set2 = { 12 16 8 }
intersection = { 8 12 }
```

همانگونه که در مثال بالا دیده می شود، تابع intersection، دو مجموعه را بعنوان ورودی دریافت و اشتراک آنها را بعنوان خروجی باز می گرداند. آرایه های A و B بعنوان پارامترهای ورودی هستند که نماینده دو مجموعه اولیه هستند. از آنجا که لزومی ندارد مقادیر این دو آرایه در تابع تغییر نماید، بعنوان پارامتر ثابت (const) به تابع ارسال شده اند. اندازه این دو آرایه نیز به ترتیب در قالب پارامترهای na و nb ارسال شده است. اما آرایه C پارامتر خروجی است که اشتراک دو مجموعه را باز می گرداند، به همین دلیل بصورت ثابت

تعریف نشده است. تابع intersection، اشتراک دو مجموعه را محاسبه و مجموعه حاصل را در پارامتر C و اندازه آن را در پارامتر nc قرار می دهد. از آنجا که هم پارامتر C (بدلیل اینکه یک آرایه است) و هم پارامتر nc (بدلیل استفاده از عملگر &) توسط ارجاع به تابع ارسال شده اند، تغییرات انجام شده در آنها (یعنی حاصل نهایی) به تابع فراخواننده منتقل خواهد شد.

در تابع اصلی ابتدا دو مجموعه بنامهای set1 و set2 تعریف شده و مقدار اولیه گرفته اند. سپس با استفاده از تابع intersection، اشتراک آنها محاسبه و حاصل در آرایه result و اندازه آن نیز در متغیر resultSize قرار گرفته است. سرانجام دو مجموعه اولیه و اشتراک آنها با فراخوانی تابع printSet چاپ شده اند.

نکته مهم دیگر آنکه خروجی یک تابع نمی تواند یک آرایه باشد. برای بازگرداندن یک آرایه از تابع، باید آن را بصورت یک پارامتر خروجی به تابع ارسال نمود.

در این قسمت، ابتدا به نحوه ارسال آرایه های دو بعدی به توابع می پردازیم و سپس آرایه های با ابعاد بالاتر می پردازیم.

شاید تصور کنید که برای تعریف یک آرایه دوبعدی بعنوان پارامتری از یک تابع، تنها قرار دادن دو علامت [] کافی است و نیازی به ذکر ابعاد آن نیست. اما متاسفانه اینگونه نیست، بلکه برنامه نویس باید تعداد ستونهای آرایه دوبعدی را صریحا مشخص نماید، اما نیازی به تعیین تعداد ردیفهای آن نیست. بعنوان مثال فرض کنید تابعی مانند test داریم که بعنوان ورودی یک آرایه دو بعدی و تعدادی پارامتر دیگر دریافت می کند.

تعریف تابع بصورت زیر اشتباه است:

```
void test(int A[][], ...) {
```

تعریف درست، تعریفی مانند زیر است:

```
void test(int A[][10] , ...) {
```

همانطور که می بینید تعداد ردیفها مشخص نشده است، اما تعداد ستونها برابر ۱۰ تعیین شده است. در هنگام فراخوانی تابع test، می توان هر آرایه دوبعدی ۱۰ ستونی را به آن ارسال کرد. آرایه ارسالی به تابع می تواند ۵×۱۰ و یا ۲۰×۱۰ باشد، اما نمی تواند مثلا ۵×۲۰ باشد.

اکنون به ۲ برنامه نمونه دقت کنید.

مثال) تابعی بنویسید که میزان فروش تعدادی شرکت در ۱۲ ماه سال را بعنوان ورودی دریافت، و میانگین فروش شرکتی را که بیشترین میانگین فروش را داشته است، بازگرداند.

حل) با توجه به صورت مسئله مسلم است که این تابع باید یک آرایه دو بعدی را بعنوان ورودی دریافت نماید. ردیفهای این آرایه به تعداد شرکتهای مورد نظر و ستونهای آن برابر ۱۲ (یک ستون برای هر ماه از سال) می باشد. بنابراین کافیست تابع را بگونه ای تعریف نماییم که یک آرایه ۱۲ ستونی را بعنوان ورودی دریافت نماید. حل دقیق بصورت زیر است:

```
float maxSales(const long int sales[][12], int companyNo) {
int i,j;
float average , max;

max = 0.0;
for (i=0 ;i< companyNo; i++) {
average = 0;
for (j= 0;j<12; j++)
average += sales[i][j] ;

if (average > max)
max = average ;
}
return(max);
}
```

مثال) برنامه ای بنویسید که حاصلضرب دو ماتریس را با استفاده از یک تابع محاسبه نماید.

حل) برای حل این مسئله ابتدا باید تابعی بنویسیم که دو ماتریس را بعنوان ورودی دریافت و حاصلضرب آنها را بازگرداند. مسلما بهترین روش برای ذخیره هر ماتریس، استفاده از یک آرایه دوبعدی است. اما مشکل اینجا

است که طبق تعاریف گفته شده، تعداد ستونهای آرایه های ورودی باید مشخص گردد و این باعث می شود که تابع نوشته شده محدود به ضرب ماتریسهای با تعداد ستونهای مشخصی گردد.

درچنین مواردی برنامه نویسان یک حد بالا برای تعداد ستونهای آرایه دوبعدی تعیین می کنند و در تعریف پارامترهای تابع از آن حد بالا استفاده می نمایند. گرچه در هنگام فراخوانی تابع، آرایه دوبعدی حتما باید دارای تعداد ستونهای مشخص شده باشد، اما لزومی ندارد همه آنها دارای داده های معتبر باشند. ممکن است فقط تعدادی از این ستونها حاوی داده های واقعی باشند.

معمولا تعداد ستونهای واقعی آرایه دوبعدی بعنوان یک پارامتر مجزا به تابع ارسال می شود. به برنامه زیر دقت کنید:

```
const int maxCol = 10;

void multiply(const int A[][maxCol], const int B[][maxCol], int m,int
p, int n,
int C[][maxCol] ) {

for (i=0; i< m; i++)
for (j=0; j< n; j++) {
sum = 0;
for (k= 0;k < p; k++)
sum += A[i][k] * B[k][j] ;

C[i][j] = sum;
}
}

void printMatrix(int matrix[][maxCol], int row,int col) {
int i,j;

for (i=0; i< row; i++) {
for (j=0; j< col ;j++)
printf("%d ",matrix[i][j]);
printf("\n");
}
```

```
}

void main() {
int matrix1[2][maxCol] = { {7 ,3 , 2} , {-2, 6, 1} };
int matrix2[3][maxCol] = { {2 , 7 , -4, -1} , { 3 ,-3, 5, -8} , {6, -7,
2, 3} };
int result[2][maxCol] ;

multiply(matrix1, matrix2, 2, 3, 4, result);

printf("matrix1 is :\n");
printMatrix(matrix1,2,3) ;
printf("\nmatrix2 is :\n");
printMatrix(matrix2,3,4) ;
printf("\nmultiply of matrix1 and matrix2 is :\n");
printMatrix(result,2,4) ;
}
matrix1 is :
 7 3 2
-2 6 1
matrix2 is :
2  7 -4 -1
3 -3  5 -8
6 -7  2  3
multiply of matrix1 and matrix 2 is :
35  26 -9  -25
20 -39 40 -43
```

اگر به تابع multiply دقت کنید، ابتدا دو ماتریس را بصورت ثابت دریافت می کند (چرا که مقادیر ماتریسهای اولیه نباید عوض شود). سپس اندازه واقعی ماتریسها را در قالب سه پارامتر m، p و n دریافت می نماید، بدینصورت که ماتریس اول $m \times p$ و ماتریس دوم $p \times n$ فرض شده است. مسلما ستونهای ماتریس اول یعنی p و ستونهای ماتریس دوم یعنی n، باید کوچکتر از حداکثر تعداد ستونها یعنی maxCol باشند. آخرین پارامتر نیز ماتریس حاصلضرب است که خروجی تابع است (و بهمین دلیل بصورت ثابت تعریف نشده است).

مسلم است که اندازه ماتریس خروجی m × n خواهدبود و نیازی به بازگردندان ابعاد آن نیست. نحوه انجام عملیات ضرب نیز قبلا و در مبحث الگوریتمها تشریح شده است. و اما در تابع اصلی، ابتدا دو ماتریس matrix1 و matrix2 تعریف شده و مقدار اولیه گرفته اند. توجه کنید که گرچه ابعاد اصلی matrix1 برابر ۲×۳ است، اما از آنجا که تابع multiply فقط ماتریسهایی را می پذیرد که تعداد ستونهای آنها برابر maxCol یعنی ۱۰ باشد، مجبور شده ایم آن را بصورت ۲×۱۰ تعریف کنیم ولی از ۷ ستون آخر استفاده نکنیم. بهمین دلیل matrix2 نیز که در حقیقت ۳×۴ بوده است، بصورت ۳×۱۰ تعریف شده است و ماتریس حاصلضرب یعنی result نیز که باید ۲×۴ باشد، بصورت ۲×۱۰ تعریف شده است.

با این تعاریف، می توان بدون هیچ مشکلی تابع multiply را فراخوانی کرد. در پایان نیز با استفاده از تابع printMatrix، ماتریسهای اولیه و حاصلضرب آنها چاپ شده است. دقت کنید که تابع printMatrix نیز یک ماتریس با maxCol ستون را به همراه تعداد واقعی سطر و ستونهای آن دریافت و آن را چاپ می نماید.

و اما ارسال آرایه های با ابعاد بالاتر به توابع نیز مشابه آرایه های دوبعدی است. به این صورت که در هنگام تعریف یک پارامتر از تابع بعنوان یک آرایه چندبعدی، مشخص کردن بعد اول لزومی ندارد، اما اندازه کلیه ابعاد بعدی باید حتما مشخص گردد. بعنوان مثال تابع زیر:

```
void test(int A[][5][10], ... ) {
```

بعنوان ورودی یک آرایه سه بعدی دریافت می نماید که حتما باید بعد دوم آن ۵ و بعد سوم آن ۱۰ باشند، اما اندازه بعد اول هر مقداری می تواند باشد.

خوب این آموزش کامل دوزبان برنامه نویسی C++ و C از خانوداه سی بود. در مقاله دوم آموزش کامل زبان C# از خانوداه سی را خواهید دید ، لطفا منتظر بمانید.

منبع : کتاب آموزش گام به گام خانوداه سی

.....

از سایت ما بازدید کنید:

http://www.hadi-vista.blogfa.com

به زودی:

http://www.hadi-vista.tk

بزرگترین وبلاگ آموزشی. در زمینه های :
دانلود موزیک . دانلود نرم افزار . معرفی بهترین وبسایتها . معرفی بهترین بازیهای دنیا .
آموزش نرم افزارهای گوناگون . آموزش ترفندهای ویندوز . آموزش ترفندهای رجیستری .
آموزش هک و ویروس نویسی . آموزش برنامه نویسی . پذیرش انواع تبلیغات در سایت با
قیمت مناسب . فروش الکترونیکی محصولات جدید . و... فقط فقط در هادی – ویستا.

آی دی (ID)
hadivista@hotmail.com