ویرایش دوم کتاب آموزش مقدماتی MFC را به علاقه مندان به برنامه نویسی تقدیم میکنم.
در این نسخه در کنار افزودن مطالب جدید بسیاری‌ازایرادات‌ویرایش‌قبلی برطرف شده است

Behzad Jenab's

Beginning MFC in Visual Studio 2008 Second Edition

Mahtab

jenab.persiangig.com

آموزش مقدماتی MFC

بهزاد جناب

مقدمه

دنیای هر انسانی به اندازه وسعت فکر اوست

سلام

از آنجایی که کتاب الکترونیک خوبی به زبان فارسی در زمینه ویژوال سی پلاس پلاس در دست نیست تصمیم گرفتم تا در کنار یادگیری این زبان آموخته های خودم را به صورت یک کتاب درآمورم تا علاقه مندان به برنامه نویسی مانند خودم بتوانند از آن برای یادگیری این زبان استفاده کنند. در تمامی آموزش های این کتاب از محیط **Visual Studio 2008** استفاده شده است.

این روزها زیاد هستند افرادی که می خواهند با یادگیری زبان **C++** اقدام به نوشتن ویروس نمایند تا مثلا سطح مهارت خود را به رخ دیگران بکشند و حس غرور درون خود را به اینصورت ارضا نمایند ، در حالی که تفاوتی بین خط انداختن روی یک ماشین با تخریب اطلاعات کامپیوتر یک شخص ، یا سرقت از کیف پول یا موبایل یک شخص با سرقت اطلاعات شخصی از کامپیوتر فرد وجود ندارد و نام همه آنها مردم آزاری یا دزدی است که باعث مدیونی و عواقب مادی و معنوی شما خواهد شد.
شما می توانید سطح مهارت خود را با نوشتن برنامه های مفید برای دیگران نیز افزایش دهید.دانشمندان و کسانی که از علمی استفاده میکنند باید به تعهدات اخلاقی و وجدانی پایبند باشند. متاسفانه به دلیل بی توجهی به این قضیه دراین زمان مثلا پزشکانی میبینیم که در اتاق عمل کلیه بیمار را میدزدند یا بیمار را بدون نیاز به جراحی و فقط برای دریافت پول عمل می نمایند ، حالا نظر شما درباره این افرادچیست؟ این هم نوع دیگری سوء استفاده از علم است.

با تشکر
بهزاد جناب
مهر ۱۳۸۸

فهرست مطالب مهم کتاب

فصل چهارم

استفاده از ماوس و کی بورد
نقاشی با ماوس
استفاده از AND و OR باینری
پرچمهای باینری(Flags)
بدام انداختن رویدادهای کی بورد
تغییر دادن کرسر

فصل پنجم

ساختن آیکون در سیستم ترای ویندوز (آیکون های بغل ساعت ویندوز)
حذف آیکون سیستم ترای در هنگام خروج یا اجرای برنامه
تشخیص کلیک شدن ماوس بر روی آیکون سیستم ترای برنامه
نمایش داده نشدن پنجره برنامه هنگام شروع
مخفی شدن پنجره پروژه هنگام انتخاب کلید Minimize
نمایش منو در صورت راست کلیک کردن بر روی آیکون سیستم ترای
تغییر خواص و مشخصات سیستم ترای برنامه مانند آیکون و متن راهنما به هنگام اجرا
نمایش بالون در سیستم ترای
محو کردن بالون به نمایش در آمده در سیستم ترای

فصل ششم

اطلاعات اولیه درباره رجیستری ویندوز
تهیه پشتیبان از رجیستری و بازیابی دوباره آن
نوشتن یک رشته از نوع CString در رجیستری ویندوز
خواندن یک رشته از نوع CString از یک کلید در رجیستری
نوشتن یک عدد از نوع long در رجیستری
خواندن یک عدد از نوع long از رجیستری
حذف داده از یک کلید در رجیستری
حذف کلید از رجیستری
شمارش کلیدهای یک مسیر از رجیستری
شمارش داده های یک مسیر از رجیستری

فصل هفتم

افزودن تایمر به برنامه
غیر فعال کردن تایمر

فصل هشتم (مباحث متفرقه)

الگوریتم تبدیل تاریخ میلادی به تاریخ شمسی

اجرای یک پنجره دیالوگ دیگر در هنگام اجرای برنامه

شیشه ای کردن پنجره دیالوگ

تغییر عکس میز کار (Desktop)

مشخص کردن تعداد درایوهای متصل به سیستم

خواندن نام فایلها و پوشه های یک مسیر

تغییر خواص یک فایل یا پوشه

حذف فایل

چرا از C++ استفاده می کنیم؟

زبان برنامه نویسی **C** اوایل دهه هفتاد اختراع شد و به سرعت به یکی از محبوبترین زبانهای برنامه نویسی حرفه ای تبدیل گردید. سی پلاس پلاس زبان مورد انتخاب برای ساخت نرم افزارهایی با کارآیی بالا است که به طور مستقیم به منابع یا تجهیزات و ابزار های ویندوز دسترسی دارند. پس شما با آن می توانید به قابلیت های سطح پایین سیستم دسترسی داشته باشید و البته به خاطر اینکه این زبان به شما قدرت بسیار بالایی می دهد ، لذا در مقایسه با سایر زبان ها مثل **visual Basic** یا **C#** شما بایستی از جزییات بیشتری اطلاع داشته باشید. بنابراین از جمله کاربرد های بدون شک این زبان برنامه هایی می باشد که نیاز به دقت بالا ، تاخیر زمانی کوتاه **(Low-latency)** و استفاده مستقیم از سخت افزار دارند ، نرم افزار هایی همچون:

نرم افزارهای گرافیگی و طراحی ۲ بعدی و ۳ بعدی ، محیط ها و موتورهای توسعه بازی های کامپیوتری ، نرم افزارهای صوتی / تصویری ، پیشرفته سیستمی و غیره **...** که چیزی فراتر از طراحی واسط کاربر باشد .

در ذیل چند دلیل موفقیت C++ را آورده ام:

- **C++** فایلهای متکی به خود می سازد. همین که برنامه تان را کامپایل و لینک کردید دیگر می توانید فایل **exe** را بدون هیچ دغدغه ای به دیگران بدهید.(البته استفاده از این روش ایراداتی و محدودیتهایی دارد که در بعضی از پروژه ها نمایان می شود و روش مطمئن تر و بهتر آن است که برنامه را به صورت نصب شونده ایجاد نمایید نه قابل حمل)
- سرعت اجرای فایل های اجرایی **C++** بسیار خوب است.
- فایل های **exe** تولید شده توسط **C++** کوچک هستند.
- سرعت کامپایل و لینک شدن برنامه های **C++** بسیار زیاد است.
- **C++** زبانی مطمئن، ساده و قدرتمند است.

به طور کلی دو نوع شیوه های برنامه نویسی برای سی پلاس پلاس وجود دارد.

بومی **(Native)** و مدیریت شده **(Managed)**

۱- در نوع **Native** که قدرتمند ترین نوع برنامه نویسی می باشد (مدیریت نشده) ، برنامه شما به طور مستقیم توسط پردازنده مرکزی **(CPU)** اجرا می شود و می تواند بر روی نسخه های مختلف سیستم عامل ویندوز اجرا شود این مورد شامل ویندوز **CE** و ویندوز **mobile** برای تلفن های همراه نیز می شود .لذا برنامه های **Native** دسترسی مستقیم به سیستم عامل و سخت افزار دارند و این به شما قدرت و کارآیی **(Performance)** بسیار بالایی می دهد. اما نکته ای که باید توجه کنید این است که قدرت بالا ، به دقت ، مسئولیت پذیری و تمرین بیشتری نیاز دارد تا موارد به درستی انجام شود .

شیوه **Native** نیز به دو نوع**MFC** و **win32** که هر دو مختص پلتفرم ویندوز هستند تقسیم میشود.

در **Win32** که قلب ویندوز شناخته می شود و شیوه برنامه نویسی سطح پایین می باشد (پشتیبانی از ۱۶ بیت تا ۶۴ بیت) ، سرعت توسعه نسبت به سایر زبان ها مانند **VB** و **C#** کمتر است و زمان و انرژی بیشتری صرف خواهد شد، اما در عوض همه چیز در اختیار شما قرار دارد ، از کنترل دقیق حافظه تا کنترل تمامی منابع سیستم و البته با نهایت کارآیی.

اکثر نرم افزار های تجاری شرکت های بزرگ و متوسط سراسر دنیا که در منازل از آن ها استفاده می کنید (و نیازی به نام بردن آن ها نیست)و تقریبا هسته اصلی تمامی بازی ها در نسخه ویندوز آن ها از این **API** ها به طور مستقیم استفاده می کنند.

MFC یا همان **Microsoft Foundation Class** ، یک**framework** می باشد که **API** های **win32** را در قالب کلاس هایی برای برنامه نویسان **C++** ارائه می کند ، تا زمان توسعه را کاهش دهد ، کار با پایگاه داده را آسان تر می کند و با وجودی که تقریبا تمامی قابلیت های سایر زبان ها را در اختیار شما قرار می دهد ، و جدا از اینکه کارآیی در مقایسه با شیوه قبل کمی کاهش می یابد ، تمامی ناحیه های **win32**را نیز در بر نخواهد گرفت و لذا نیاز به آشنایی با خود **API** های ویندوز نیز می باشد .

ضمنا **MFC** در کشورمان کاربران بیشتری دارد.

از جمله محصولات شرکت **Nero** و همین طور ابزار های همراه آن که در سال های اخیر عرضه شده اند ، همچون کپی **CD/DVD** ، پخش فیلم ، ویرایش موسیقی و غیره ... از **MFC** استفاده می کنند.

- مزیت – کارآیی بالا
- مزیت – کمترین میزان نیاز به منابع سخت افزاری مانند حافظه **Ram** و فضای دیسک و ...
- نقص – پیچیدگی بیشتر و دارای زمانبری بیشتر برای کارکردن و نوشتن با آن
- نقص – وابسته به پلتفرم ویندوز

۲- در نوع **Managed** که یک محیط **runtime** به نام **CLR** برای شما فراهم می کند ، شما را از این پیچیدگی کار و قرار دادن تحت سیستم عامل و سخت افزار جدا می کند و برنامه نویسی را بسیار سریع تر و آسان تر می کند. اما در هر حال این جدایی ، انعطاف پذیری **(flexibility)** و به احتمال غریب به یقین کارآیی **(Performance)** کمتری دارد ، که البته این موارد بستگی به پروژه مورد نظر دارد که آیا کارآیی ، مورد اهمیت می باشد یا خیر .ضمنا این مورد نیاز به نصب **Net.** در سیستم مورد نظر دارد.

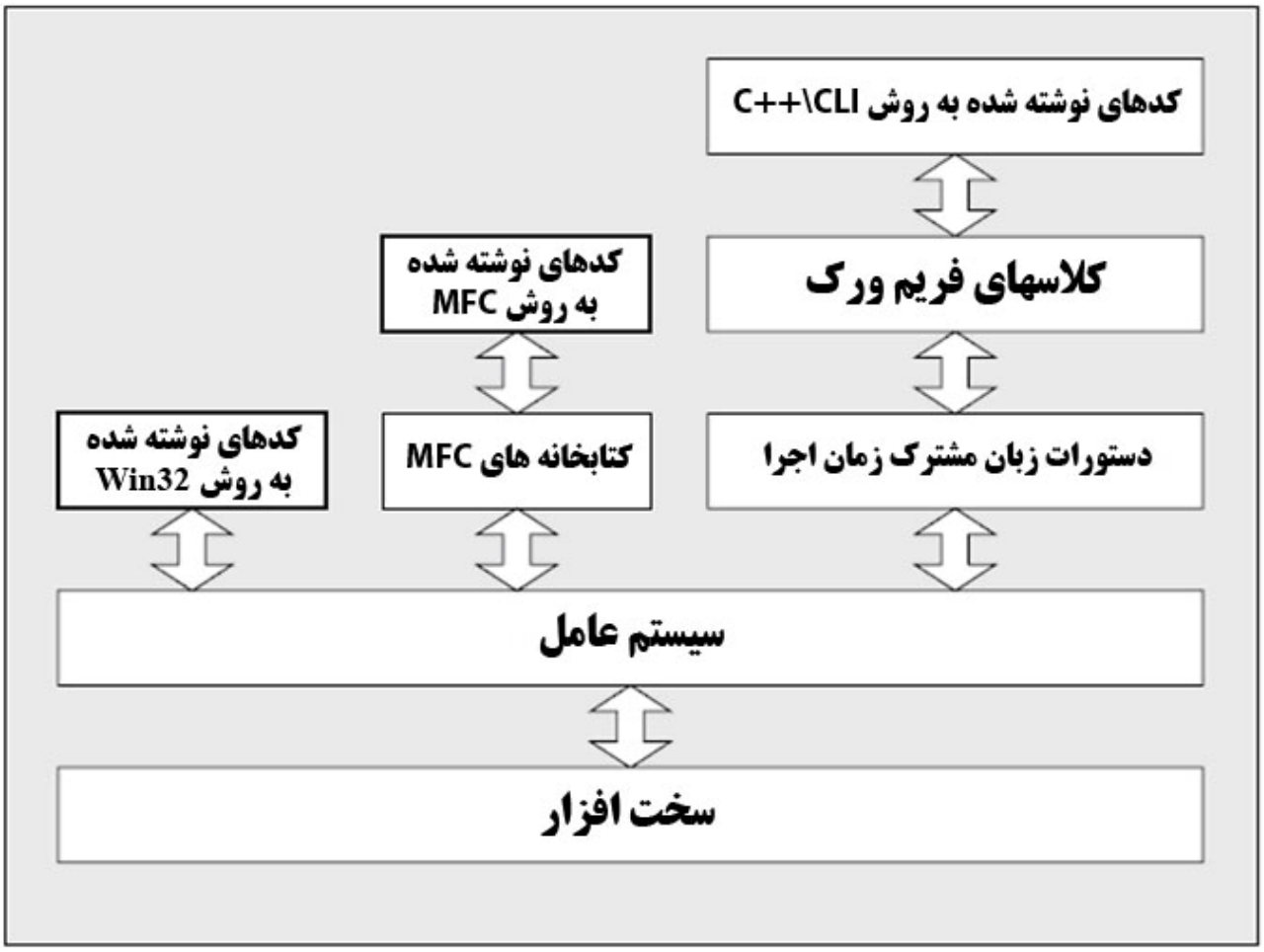
- مزیت – قابلیت حمل (قابل استفاده در هر سیستم دارای **CLR**)
- مزیت – سهولت در این نوع برنامه نویسی
- نقص – احتمال اجرای کند تر
- نقص – امکان نیاز بیشتر به منابع سیستمی ، حافظه و فضای دیسک و ...

طبق گفته های مکرر تیم طراحی **Visual C++** ، طبق سیاست جدید مایکروسافت ، تصمیم بر آن گرفته شده که برای راضی نگه داشتن مشتریان ، توسعه بخش **Native** توسط **C/C++** ، به سرعت و همراه با پیشرفت تکنولوژی انجام شود (و با ارائه بیش از ۷۰۰۰ **API** جدید برای ویندوز **vista** تمامی شک های احتمالی بر طرف شد و امید پشتیبانی بخش **native** برای سال های آینده به واقعیت قطعی تبدیل شد و

دیگر اجباری در کوچ کردن به دات نت نمی باشد)، که این کار باعث ایجاد تاخیر در توسعه بخش دات نت برای **C++/CLI** می شود و بیان شده است که:

کاربران انتظار نداشته باشند کاری که قبلا توسط سایر زبان ها انجام پذیرفته است حتما و فورا برای **C++/CLI** نیز انجام شود و به کسانی که می خواهند که با این روش **Managed** برنامه نویسی کنند توسیه می شود از **C#** (سی شارپ) استفاده کنند.

همانطور که در عکس زیر ترتیب اجرای یک فرمان در روشهای مختلف برنامه نویسی در **C++** را مشاهده میکنید، روش **Native** که شامل **MFC** و **Win32** می شود بسیار به سخت افزار نزدیکتر بوده تا روش **Managed** و بنابراین سرعت آن نیز بالاتر است.



بنابراین طبق گفته های بالا

- ✓ برنامه نویسی در محیط ویژوال سی پلاس پلاس با استفاده از روش مدیریت شده(CLR) برای نوشتن یک پروژه پیشنهاد نمی شود (بجز استفاده از آن به صورت ترکیبی در برنامه های **MFC** ، چنانچه قصد استفاده از این روش را دارید پیشنهاد می شود از زبان **C#** استفاده کنید.
- ✓ برای نوشتن برنامه هایی مانند حسابداری ، بانک اطلاعاتی ، ویرایشگر تصاویر و متن که اصولا عامل سرعت در آنها احمیت زیادی ندارند از روش **MFC** استفاده کنید.
- ✓ برای نوشتن برنامه های نیازمند به سرعت پردازش بالا مانند بازی ها از روش **Win32** استفاده کنید.

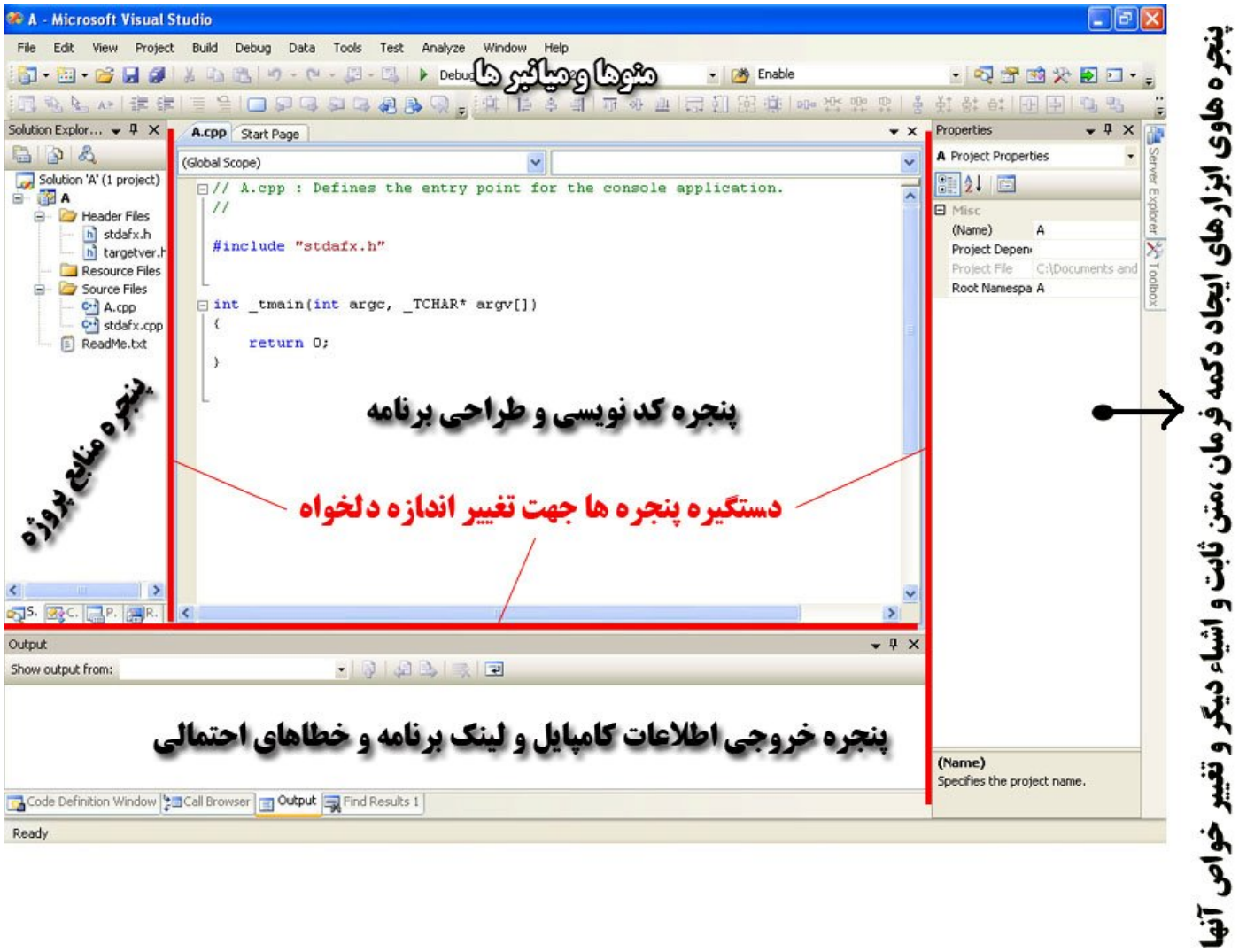
آموزش مقدماتی MFC

بهزاد جناب

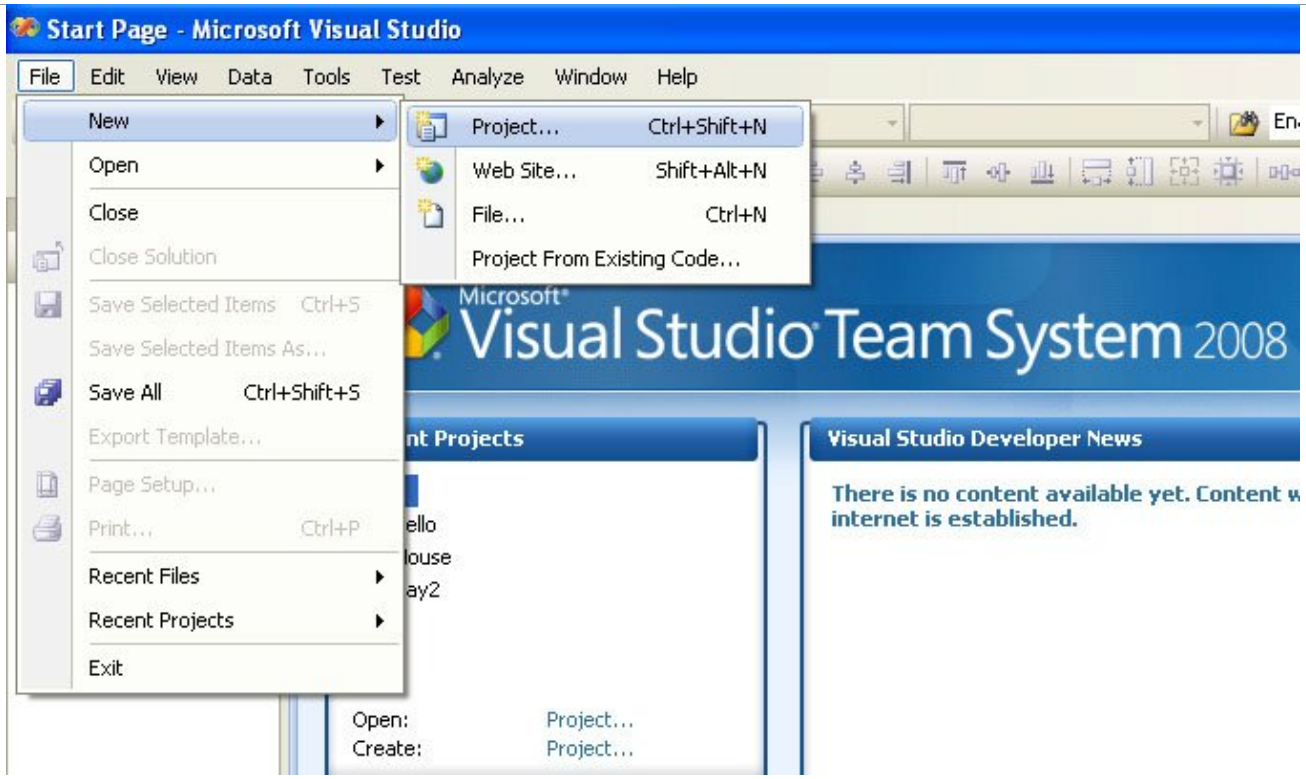
فصل اول

آشنایی با کلیات و مفاهیم زبان سی پلاس پلاس

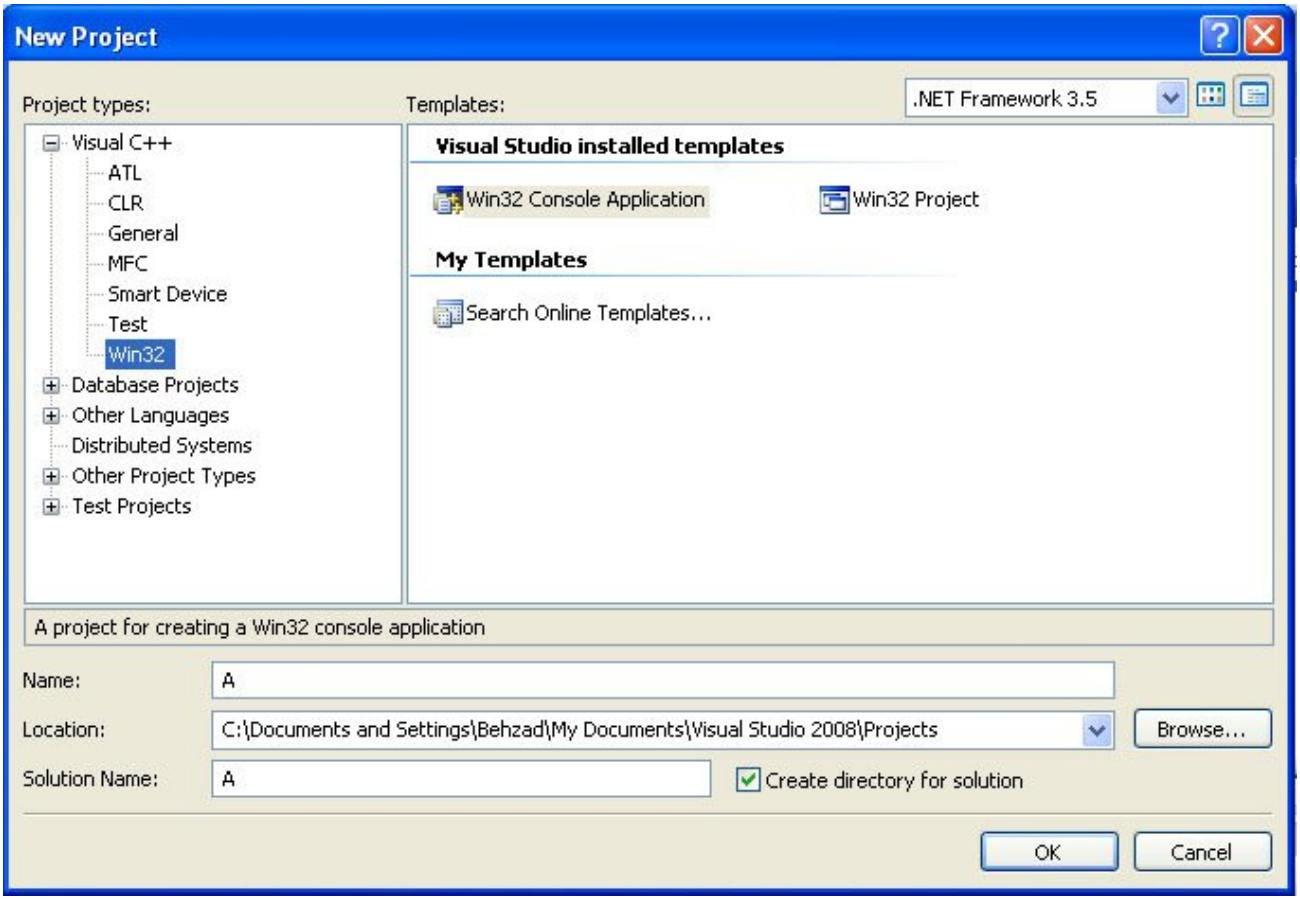
قبل از اینکه شروع به برنامه نویسی و دادن توضیحاتی مربوط به کد ها نماییم ابتدا با محیط Visual Studio 2008 بوسیله عکس زیر کمی آشنا میشویم.در قسمت بالایی منوها و میانبرها و در پائین چهار پنجره به هم چسپیده ، محیط ویژوال استدیو را تشگیل میدهند که بوسیله دستگیره مشخص شده در شکل قابلیت تغییر اندازه به صورت دلخواه ما را دارا هستند.هر کدام از این پنجره ها نیز دارای قسمتهای(Tab) مختلفی هستند.



برای آشنایی با اصول اولیه برنامه نویسی با سی پلاس پلاس یک برنامه ساده DOS می نویسیم. ابتدا از منویFile گزینه New و سپس Project را انتخاب کنید.

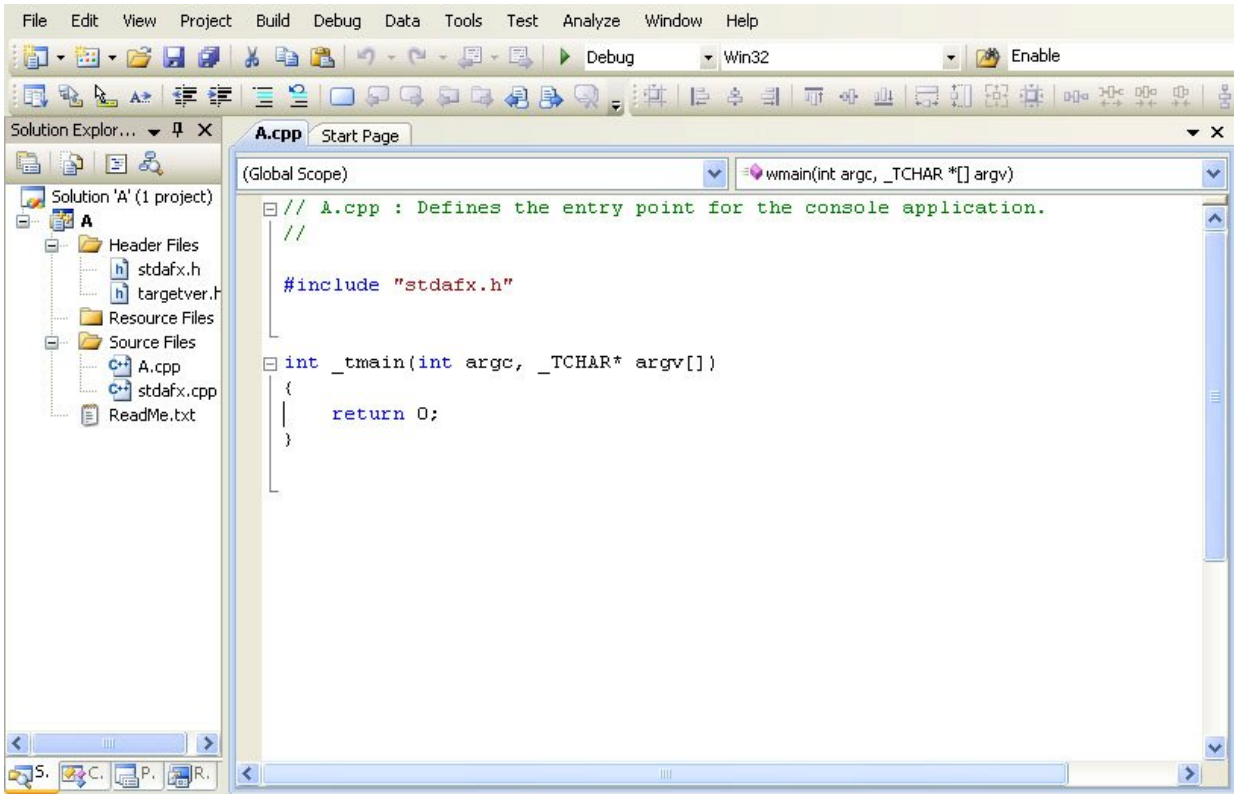


سپس **Win32 Console Application** را انتخاب کرده و نام آن را **A** بگذارید و بر روی **OK** کلیک کنید.



سپس در پنجره دوم به نمایش درآمده بر روی **Finish** کلیک کرده تا پروژه ساخته شود.

در این مرحله شما متن درون فایل A.cpp را مشاهده میکنید.(پسوند cpp به معنای CPlusPlus یا همان C++ است که در اینجا مشاهده میکنید)



اگر تا به حال برنامه ای به زبان C++ ندیده باشید این کدها برایتان بسیار نامفهوم خواهد بود اما نگران نباشید با تمای آنها آشنا خواهید شد.
در دو خط اول از علامت // استفاده شده است که این علامتها به معنای شروع توضیحاتی هستند که برای خوانا شدن برنامه نوشته میشود و هر عبارتی که جلوی آنها نوشته شود هنگام تبدیل پروژه به یک فایل اجرائی توسط کامپایلر نادیده گرفته میشوند.
حال سئوال این است که فایده استفاده این توضیحات در چیست؟ وقتی یک برنامه نوشته میشود دارای بخشهای زیادی است که حتی خود نویسنده هم بعد از مدتی برای اصلاح و تغییر و بررسی برنامه بدون این توضیحات گیج می شود و کلا برای پیدا کردن خطا ها و بخشهای مختلف و ویرایش مجدد یک برنامه بسیار مفید هستند.
حال قصد داریم توضیحاتی به برنامه اضافه کنیم، خط زیر را به برنامه اضافه نمایید.

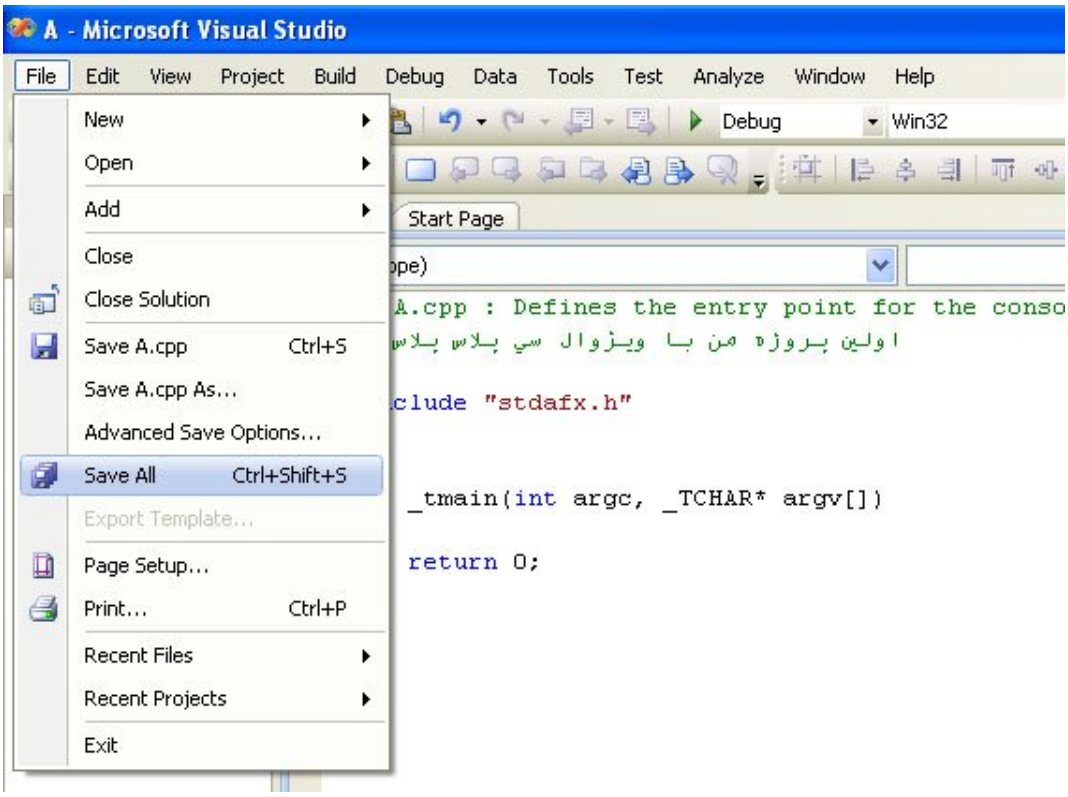
```
// اولین پروژه من با ویژوال سی پلاس پلاس
```

در روشی دیگر برای نوشتن توضیحات در C++ توضیحات با /* شروع و با */ پایان می یابد .
به طور کلی برای نوشتن توضیحات در برنامه از دو روش زیر استفاده می شود.

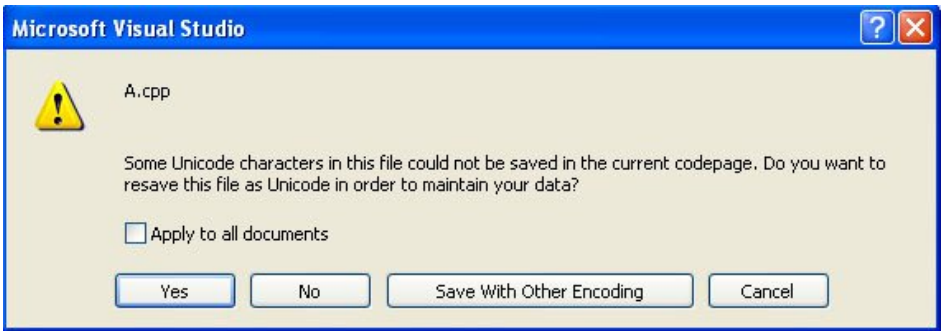
```
/* اولین پروژه من با ویژوال سی پلاس پلاس */

/*****************************
*                بهزاد جناب          *
*    اولین پروژه من با ویژوال سی پلاس پلاس    *
*****************************/
```

حالا می خواهیم برنامه را قبل از اجرا ذخیره نمایید. برای ذخیر کل پروژه می توانید به دو روش زیر اقدام نمایید.

۱. از منوی File گزینه Save ALL را انتخاب.
۲. کلید های Ctrl+Shift+S را فشار دهید.

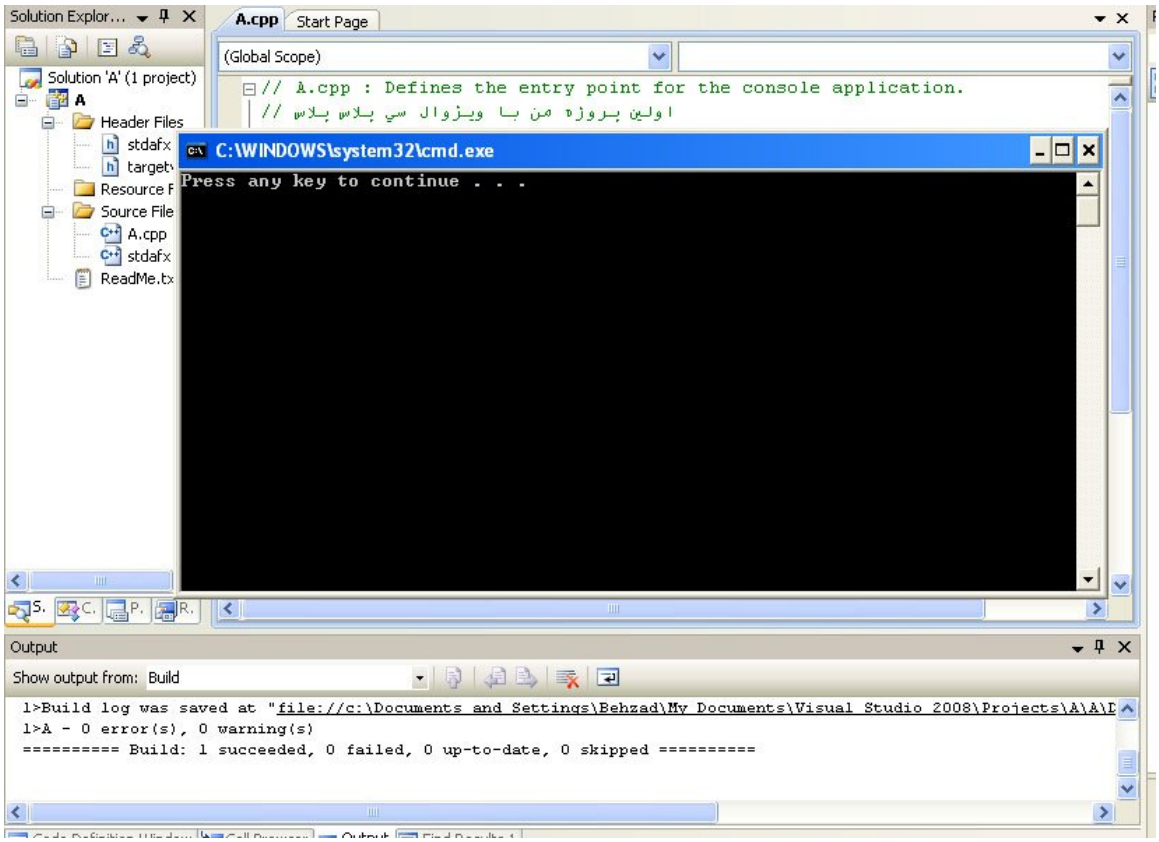


بعد از اقدام ما برای ذخیره پروژه به خاطر استفاده از زبان فارسی که کارکترهای آن به صورت یونی کد (Unicode) است هشداری از ویژوال استدیو دریافت میکنیم که پروژه ما باید به صورت یونی کد ذخیره شود که شما باید گزینه Yes را انتخاب کنید تا نوشته های فارسی شما به درستی ذخیره شده و در مراجعات بعدی به صورت علامتهای ؟ نشان داده نشوند.



حالا برای تبدیل این برنامه به یک فایل اجرایی (exe) و مشاهده خروجی برنامه از کلید Ctrl+F5 استفاده می کنیم.در پنجره خروجی در قسمت پایین شما شاهد اطلاعات لینک و کامپایل این برنامه هستید که به معنای نبود وجود خطا و هشداری به هنگام کامپایل برنامه شماست و همچنین موفقیت در تبدیل این پروژه به فایل اجرائی(exe).

از آنجایی که هنوز در این برنامه هیچ کدی ننوشته ایم هیچ کاری هم توسط این برنامه انجام نمیشود و در پنجره بوجود آمده توسط برنامه هیچ نوشته ای مشاهده نمیکنیم به غیر از عبارت "Press any key to continue . . ." که توسط خود کامپایلر نوشته شده است نه بوسیله برنامه شما ، آن هم به خاطر اینکه برنامه نویس قبل از بسته شدن پنجره شاهد خروجی برنامه خود باشد،چون در حالت عادی به محز اتمام برنامه این پنجره نیز بسته میشود.



در خطهای بعدی بدنه اصلی برنامه که تابع **main** است را مشاهده می کنید.ولی این تابع به صورت **_tmain** نوشته شده که هیچ تفوتی ندارد و فقط برای حل مشگل استفاده از حروف یونی کد(**Unicode**) مانند زبان فارسی است.

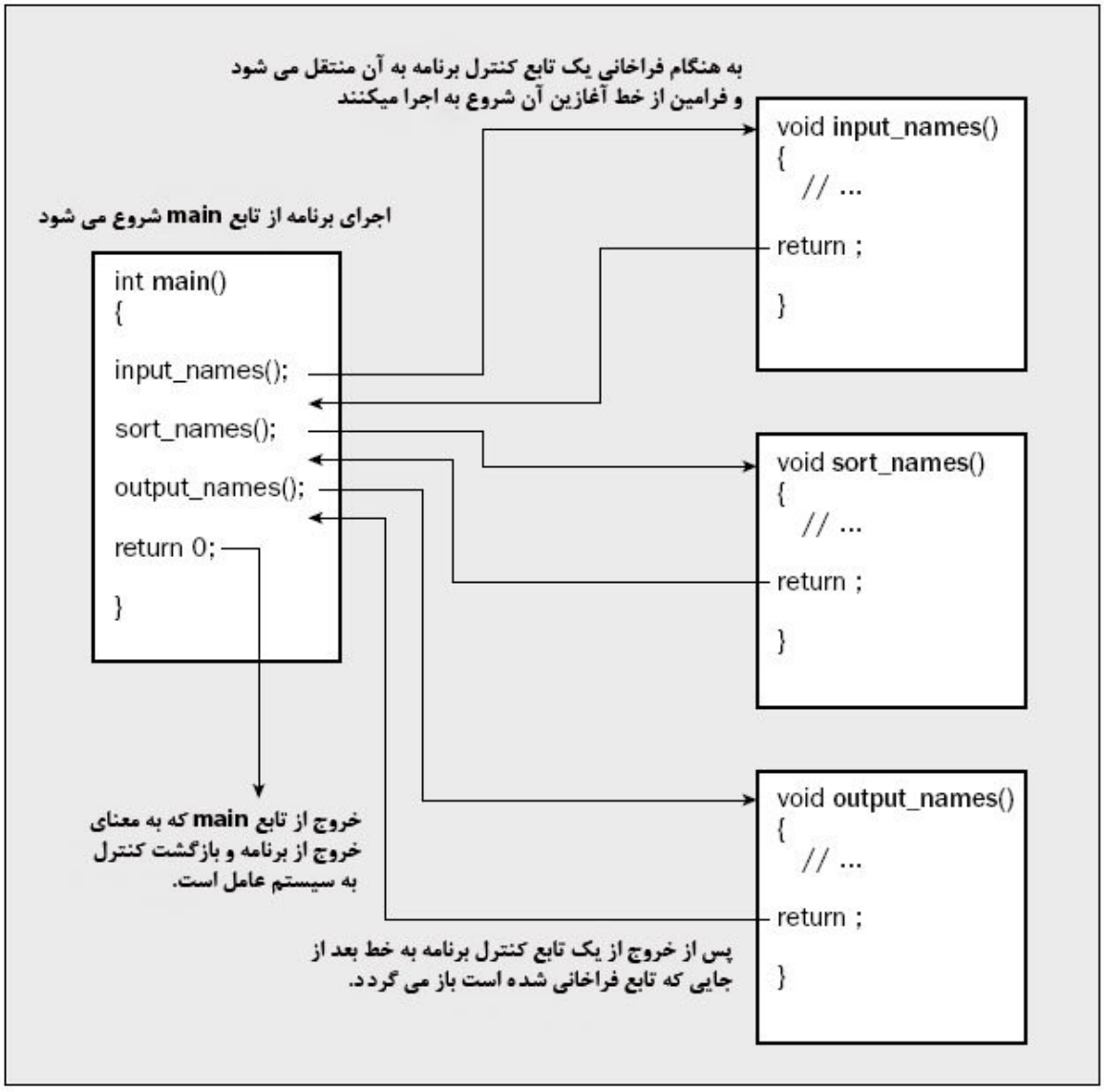
```
int _tmain(int argc, _TCHAR* argv[])
{
      return 0;
}
```

تابع یک بلوک کد است که بین دو{} قرار می گیرد.نمونه ذیل هم یک تابع است:

```
void MyFunction();
{
   کد تابع در این قسمت قرار می گیرد //
}
```

در برنامه های C++ حتما باید یک تابع **()main** وجود داشته باشد. تابع **()main** اولین تابعی است که در یک برنامه C++ اجرا می شود.
با اجرای این برنامه دستورات خط به خط بعد از علامت { تابع **main** شروع به اجرا و با رسیدن به عبارت **return 0;** خاتمه پیدا میکند و از برنامه خارج میشود.

شکل زیر نحوه اجرای توابع و در برنامه های C++ را نشان می دهد.



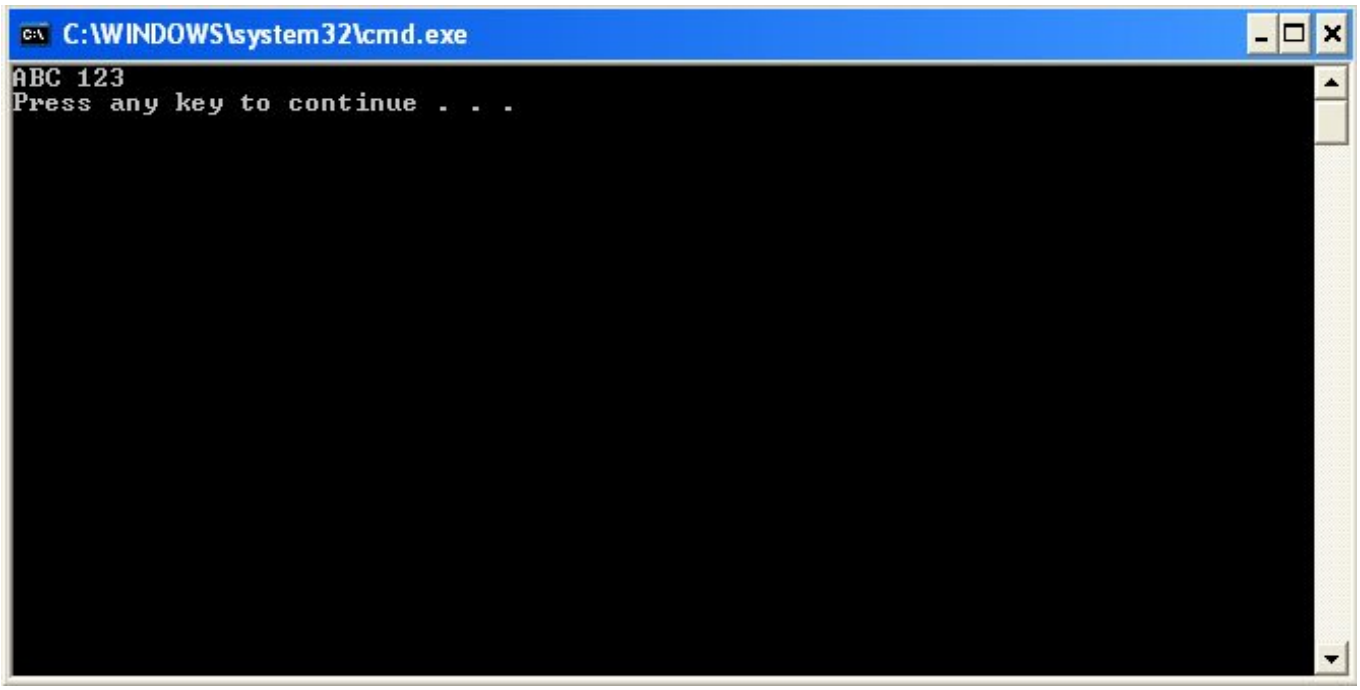
حالا دو خط جدید به این برنامه اضافه می کنیم و به شکل زیر تغییر میکند

```
// A.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
	std::cout << "ABC 123 \n";
	return 0;
}
```

دستور #include <iostream> باعث میشود که محتویات فایلی به نام iostream.h به برنامه ما اضافه شود چون برای چاپ نوشته مان نیاز به تابع std::cout داریم که در درون فایل ذکر شده قرار دارد، دستور دوم هم که عبارت std::cout << "ABC 123 \n"; است که باعث چاپ عبارت داخل گیومه در پنجره خروجی برنامه می شود، دستور cout به کامپیوتر می گوید که هر چه بعد از << وجود دارد نمایش دهد.کاراکتر \n که چاپ نشده است در واقع به معنای برگشت به سرخط است و کار فشرده شدن کلید اینتر(Enter) را انجام می دهد.

نکته مهم این است که در پایان هر خط کد نوشته شده در C++ باید یک علامت ; قرار گیرد.

برنامه را با Ctrl+F5 کامپایل و اجرا نمایید، خروجی به شکل زیر است.

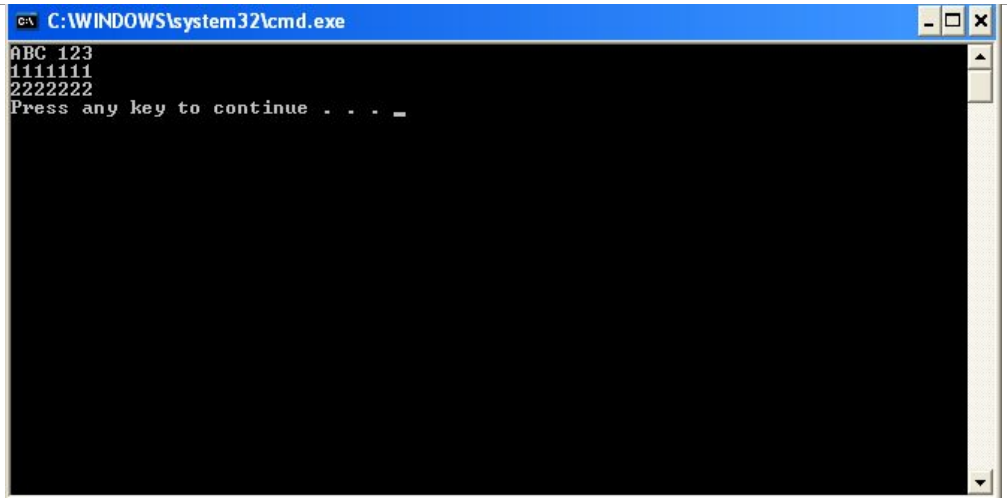


حالا دوخط دیگر نیز به برنامه اضافه میکنیم.

```
std::cout << "1111111" << std::endl;
std::cout << "2222222" << std::endl;
```

تنها قسمت جدید عبارت std::endl است که در واقع همان کار عبارت \n داخل گیومه("") که رفتن به خط جدید است را انجام میدهد.

دوباره برنامه را با Ctrl+F5 کامپایل و اجرا نمایید، خروجی به شکل زیر خواهد بود.



جمع کردن در برنامه A.CPP:

در این قسمت کاری می کنیم تا برنامه بتواند دو عدد را با هم جمع کند.

۱-محتوی فایل را به صورت ذیل تغییر دهید:

```
#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
	int a;
	int b;
	int c;

	a = 2;
	b = 3;
	c = a + b;

	std::cout << "a=" << a << std::endl;
	std::cout << "b=" << b << std::endl;
	std::cout << "a+b=" << c << std::endl;

	return 0;
}
```

می بینید که تمام کدهای قبلی را حذف کرده ایم، در اولین دستور یک متغییر بنام **a** تعریف شده است. در C++ قبل از استفاده از هر متغییری باید آن را تعریف(**declare**) کرد. متغییرها می توانند داده ها را در خود نگهدارند. هنگام تعریف یک متغییر باید نام و نوع آن را مشخص کنید، به هر حال کامپایلرها باید بدانند که چه نوع داده ای قرار است در این متغییر قرار گیرد.کلمه **int** به کامپایلر می گوید که این متغییر داده ای از نوع عدد صحیح را خواهد گرفت.دو متغییر دیگر هم به نامهای **b** و **c** تعریف کرده ایم سپس به متغییر های **a** و **b** مقدار داده شده است.

در سی پلاس پلاس انواع مختلفی از متغییر وجود دارد که هر کدام برای ذخیره نوع خاصی از داده ساخته شده است. در زیر لیستی از انواع آنها مشاهده می کنید.در نامگزاری متغییر در C++ فقط می توان از حروف انگلیسی ، عدد و علامت _ استفاده کرد.ولی نباید اولین حرف تشگیل دهنده نام متغییر عدد باشد مثلا متغییر **1A** خطا است ولی **A12h** صحیح می باشد، و یک نکته بسیار مهم اینکه حروف کوچک و بزرگ در C++ متفاوت هستند یعنی دو متغییر به نامهای **A** و **a** با هم متفاوت هستند و یا دو متغییر با نامهای **Behzad** و **behzad** نیز باهم متفاوتند.

تعریف متغییر از فرمول زیر انجام پذیر است:

نوع متغییر نام متغییر;

مثلا برای تعریف متغییری به اسم a برای ذخیره عدد صحیح از دستور int a; و برای ذخیره اعداد اعشاری در متغییری به نام b از دستور double b; استفاده می کنیم.از روی جدول زیر می توانید انواع داده در C++ مشاهده نمایید.

نوع داده	کاربرد	بایت مصرفی	محدوده
bool	ذخیره درست و غلط	۱	مقدار صفر یا یک true or false
char	ذخیره یک کارکتر	۱	–128 to +127
signed char	ذخیره یک کارکتر	۱	–128 to +127
unsigned char	ذخیره یک کارکتر	۱	0 to 255
wchar_t	ذخیره یک کارکتر یونیکد	۲	0 to 65,535
short	ذخیره عدد صحیح کوچک	۲	–32,768 to +32,767
unsigned short	ذخیره عدد مثبت صحیح کوچک	۲	0 to 65,535
int	ذخیره عدد صحیح متوسط	۴	–2,147,483,648 to 2,147,483,647
unsigned int	ذخیره عدد مثبت صحیح متوسط	۴	0 to 4,294,967,295
long	ذخیره عدد صحیح بلند	۴	–2,147,483,648 to 2,147,483,647
unsigned long	ذخیره عدد مثبت صحیح بلند	۴	0 to 4,294,967,295
float	ذخیره اعداد اعشاری	۴	با ۷ رقم دقت در اعشار ±3.4·10±38
double	ذخیره اعداد اعشاری بلند	۸	با ۱۵ رقم دقت در اعشار ±1.7·10±308
long double	ذخیره اعداد اعشاری بلند	۸	با ۱۵ رقم دقت در اعشار ±1.7·10±308

اما چه نوع داده هایی را می توان در این نوع متغیرها ذخیره نمود، در جدول زیر مثالهایی برای آشنایی شما آورده ام.

نوع داده	مثال هایی از انواع ورودی های قابل قبول برای ذخیره در این نوع داده
char, signed char و unsigned char	‘A’, ‘Z’, ‘8’, ‘*’
wchar_t	L’A’, L’Z’, L’8’, L’*’
int	-77, 65, 12345, 0x9FE
unsigned int	10U, 64000u
long	-77L, 65L, 12345l
unsigned long	5UL, 999999999UL, 25ul, 35Ul
float	3.14f, 34.506F
double	1.414, 2.71828
long double	1.414L, 2.71828l
bool	true, false

می توان هنگام تعریف متغییر به دو روش زیر مقدار اولیه ای نیز به آنها اختصاص داد.

```
int a = 0;
int b = 10;
int abc = 5;
```

یا

```
int a(0);
int b(10);
int abc(5);
```

نکته: در C++ فاصله های بین دستورات به کلی نادیده گرفته می شود ، به عبارتی دیگر هر چهار دستور ذیل معادل یکدیگر و صحیح هستند.

```
a                    = 2;
a =                  2;
a  =
                   2;
```

البته یک استثناء وجود دارد و آن عبارت بین گیومه(" ") است، دستور ذیل خطا است:

```
std::cout << " I am
                Behzad";
```

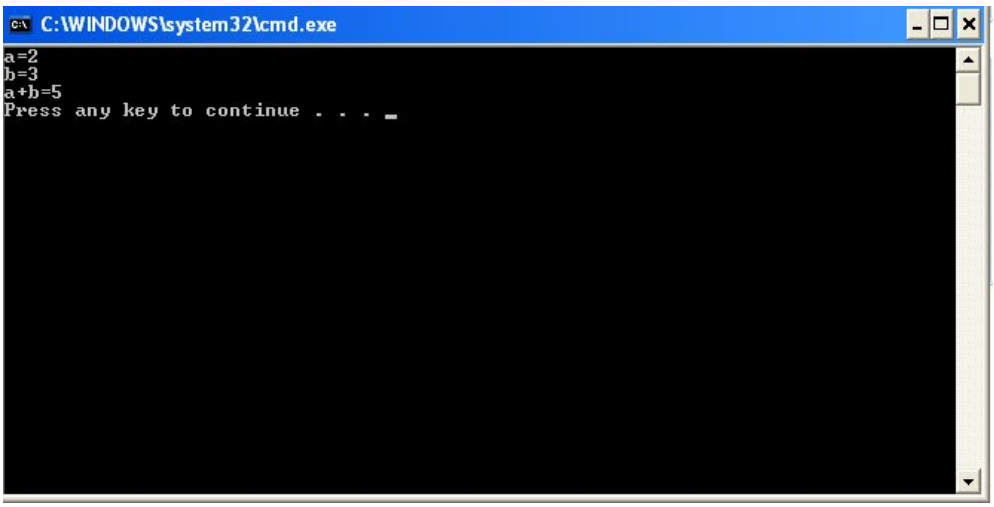
نکته: در C++ پایان یک خط را علامت ; مشخص می کند یعنی شما در یک خط مانند زیر می توانید از چندین دستور استفاده کنید و این دستورات هرکدام یک خط جداگانه به حساب می آیند.

```
int a,b,c;   a=20;    b=30; c=a+b;
```

جمع دو عدد با دستور c=a+b انجام می شود.سپس حاصل جمع با دستورات:

```
std::cout << "a=" << a << std::endl;
std::cout << "b=" << b << std::endl;
std::cout << "a+b=" << c << std::endl;
```

نمایش داده می شود. چون عبارات a،b و c درون " " قرار ندارند مقدار آن نمایش داده می شود نه کلمه a،b و یا c.
پس از اجرای برنامه نتیجه کار به شکل زیر خواهد بود.



عملگرهای ریاضی در زبان C++ به صورت زیر تعریف میشوند

علامت عملگر در C++	نام عملگر
+	جمع
-	تفریق
*	ضرب
/	تقسیم
%	باقیمانده تقسیم
++	افزایش یک واحد به متغییر مورد نظر
--	کاهش یک واحد از متغییر مورد نظر

در مورد دو عملگر آخر یعنی ++ و -- روش استفاده به شکل زیر است و باعث کاهش و یا افزایش یک واحد از متغییر می شود. مثلا دستور a++ معادل با دستور a=a+1 است و دستور b-- معادل دستور b=b-1 است، و به شکل زیر مورد استفاده قرار می گیرد.

```
a++;
b--;
```

توابع در C++

برنامه ما فعلا یک تابع به نام main() دارد حالا می خواهیم یک تابع دیگر هم اضافه کنیم ، این تابع را Add() نامیده ایم.

۱-برنامه را چنین تغییر دهید:

```
#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
	int a,b,c;

	a = 2;
	b = 3;
	c = Add(a,b);

	std::cout << "a=" << a << std::endl;
	std::cout << "b=" << b << std::endl;
	std::cout << "a+b=" << c << std::endl;

	return 0;
}
```

نکته: در خط اول تابع(int a,b,c;) اینبار یکجا سه متغییر از نوع int (عدد صحیح) تعریف کرده ایم که هیچ تفاوتی با روش قبلی ندارد، فقط جهت آشنایی شما با این روش است که به کد نویسی کمتری نیاز دارد.

در اینجا فقط به جای دستور c=a+b دستور c=Add(a,b) را نوشته ایم تابع Add() را بعدا می نویسیم، فقط توجه کنید که این تابع دو پارامتر میگیرد آنها را با هم جمع کرده و سپس حاصل جمع را به متغییری که با آن برابر نهاده شده می دهد.
اگر برنامه را در این مرحله کامپایل کنید با خطا مواجه خواهیم شد چرا؟ چون C++ هیچ چیز درباره تابع Add() نمی دادند و این وظیفه شماست که این را به او بفهمانید.

تعریف پیش اعلام تابع Add()

وقتی کامپایلر به دستوری که Add() در آن است می رسد باید بداند که این تابع است که دو پارامتر ورودی می گیرد(پارامتر هایی از نوع int) و یک عدد صحیح بر می گرداند. این کار با تعریف پیش اعلام(Prototype) انجام خواهد شد:

```
int Add(int , int);
```

حالا تعریف پیش اعلام تابع Add() را به برنامه اضافه کنید:

```
#include "stdafx.h"
#include <iostream>

int Add(int , int);

int _tmain(int argc, _TCHAR* argv[])
{

	int a;
	int b;
	int c;

	a = 2;
	b = 3;
	c = Add(a,b);

	std::cout << "a=" << a << std::endl;
	std::cout << "b=" << b << std::endl;
	std::cout << "a+b=" << c << std::endl;

	return 0;
}
```

حالا دیگر کامپایلر درباره تابع Add() کاملا چشم و گوش بسته نیست. در قسمت آینده خود این تابع را هم تعریف خواهیم کرد.(تنها تابعی که به پیش اعلام نیاز ندارد همان تابع main() است، اما سایر توابع حتما باید پیش اعلام داشته باشند.)

نوشتن کد تابع Add()

کد ذیل که مربوط به تابع Add() است را به برنامه اضافه کنید:

```
int Add(int num1,int num2)
{
	int addnum;
	addnum = num1 + num2;
	return addnum;
}
```

این کد بسیار شبیه تعریف پیش اعلام آن است با این تفاوت که در انتهای خط تعریف تابع ; ندارد. در پیش اعلام نام پارامترها قید نشده و فقط نوع آنها مشخص شده است، اما در این جا نام ها هم قید شده اند.
و در نهایت کد برنامه به صورت ذیل خواهد شد:

```
#include "stdafx.h"
#include <iostream>

int Add(int , int);

int _tmain(int argc, _TCHAR* argv[])
{

	int a;
	int b;
	int c;

	a = 2;
	b = 3;
	c = Add(a,b);

	std::cout << "a=" << a << std::endl;
	std::cout << "b=" << b << std::endl;
	std::cout << "a+b=" << c << std::endl;

	return 0;
}

int Add(int num1,int num2)
{
	int addnum;
	addnum = num1 + num2;
	return addnum;
}
```

با اجرای این برنامه دقیقا همان خروجی را مشاهده میکنید ولی این بار جمع دو متغییر با استفاده از فراخانی یک تابع انجام می شود.

ارسال پارامتر به یک تابع

در تابع main() دیدید که پارامترهای ارسالی به تابع Add() دو عدد صحیح a و b بودند اما در اینجا نام آنهاnum1 و num2 است. همچنین دقت کنید که مقدار برگشتی این تابع addnum است در حالی که در تابع main() مقدار برگشتی را به c نسبت دادیم. در کل باید گفت که کامپایلر هنگام برخورد به تابع Add() پارامتر ها را به همان ترتیب گفته شده به آن تابع می فرستد و مقدار برگشتی را همانطور که از آن خواسته شده به یک متغییر نسبت می دهد. یعنی در اینجا مقدار a با num1 ،مقدار b با num2 و مقدار c با addnum برابر هستند.
برنامه را ساخته و اجرا کنید. خواهید دید که همانطور که انتظار داریم کار خواهد کرد.

میدان دید متغییر ها

یکی از مهمترین مفاهیم C++، میدان دید(Scope) متغیرهاست. هر متغییر را تنها در تابعی می توان به کار برد که در آن تعریف شده است و به این نوع متغییر ها محلی (Local) گفته می شود و اگر متغییری با نام a در تابع main() تعریف شده باشد فقط درون این تابع قابل استفاده است

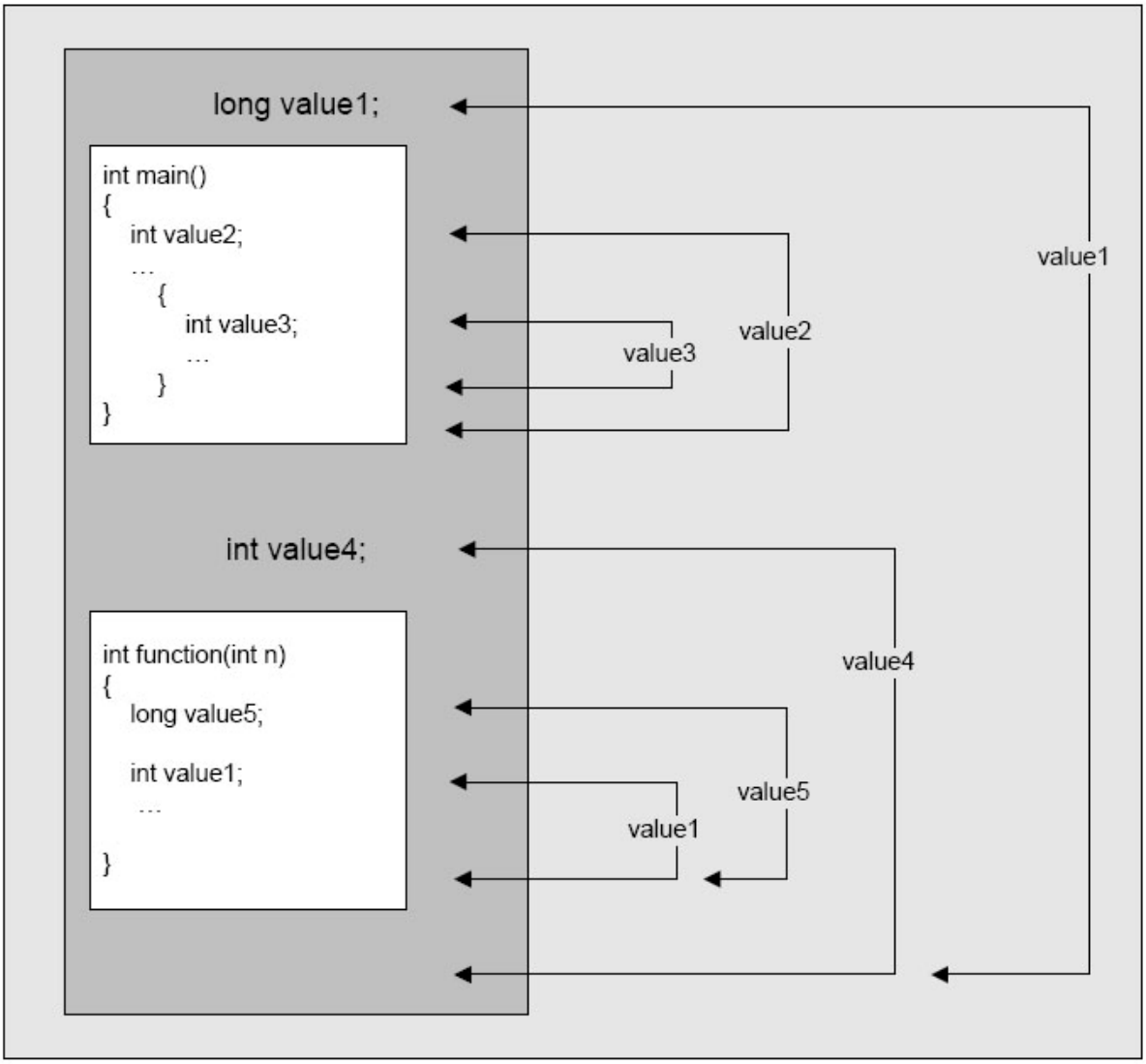
و اگر خارج از این تابع که تعریف شده به کار گرفته شود در یک تابع دیگر، کامپایلر تولید خطا خواهد کرد، یعنی متغییر a تعریف شده در دو تابع main() و Add() هرکدام مجزا هستند و با اجرای تابع تولید و با اتمام و خروج از آن ،از بین میروند.

نوع دیگری از متغییر ها با عنوان متغییر های همگانی (Global) وجود دارند که از تمام توابع و نقاط مختلف برنامه می شود آنها را تغییر داد و نیازی به تعریف مجدد در توابع ندارند، که محل تعریف آنها خارج از تابع main() و بالای آن است.

در مثال زیر متغییر a به صورت همگانی تعریف می شود سپس مقدار 1 در آن قرار گرفته و نمایش داده می شود.

```
#include "stdafx.h"
#include <iostream>

int a;

int _tmain(int argc, _TCHAR* argv[])
{
       a=1;
       std::cout << a << std::endl;
      return 0;
}
```

در عکس زیر محدوده دسترسی و میدان دید چند متغییر که در جاهای مختلف برنامه تعریف شده اند را مشاهده میکنید.



دستور if

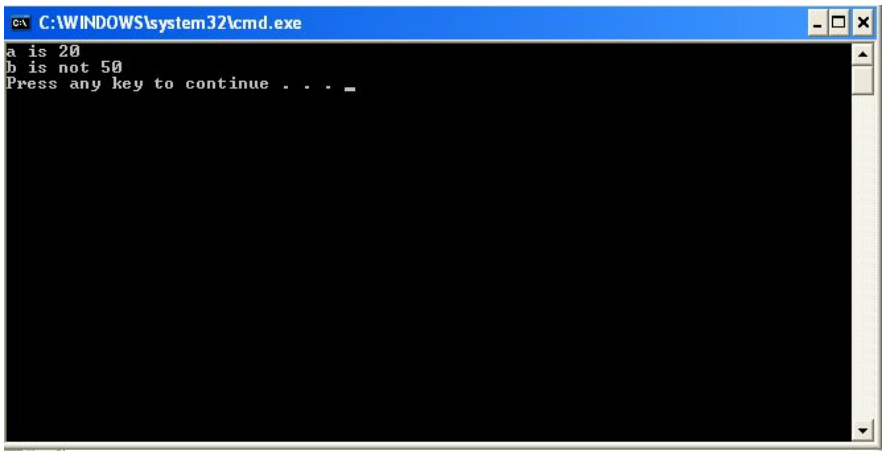
در این قسمت به دستور **if** در C++ می پردازیم. دستورات **if** مقدار یک متغییر را سنجیده و بر اساس آن کارهای مختلفی را انجام می دهد. برای آشنا شدن با **if** به مثال زیر توجه کنید.

۱-تابع **main()** را مانند زیر اصلاح کنید:

```
#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;

	a=20;
	b=30;

	if(a == 20)
	{
		std::cout << "a is 20" << std::endl;
	}

	if(b == 50)
	{
		std::cout << "b is 50" << std::endl;
	}
	else
	{
		std::cout << "b is not 50" << std::endl;
	}

	return 0;
}
```

در اینجا دو متغییر تعریف شده و سپس به آنها مقدار داده شده است. اولین دستور **if** مقدار متغییر **a** را می سنجد و اگر مقدار آن ۲۰ بود (که هست) پیام "a is 20 " را می دهد. دستور **if** دوم متغییر دیگر یعنی **b** را می سنجد و اگر مقدار آن ۵۰ باشد (که نیست) پیام "b is 50 " را نشان می دهد و در غیر این صورت که چنین است پیام "b is not 50 " را نشان خواهد داد(توجه کنید که در دستور **if** برای سنجیدن تساوی از == استفاده کرده ایم نه یک =).

۲-کارتان را ذخیره کرده و برنامه را اجرا کنید، نتیجه اجرای برنامه به شکل زیر خواهد بود.



نکته:درک تفاوت = و == در C++ اهمیت زیادی دارد. علامت = برای مقداردهی بکار می رود یعنی مقدار عبارت سمت راست علامت = در متغییر سمت چپ علامت = قرار داده می شود. ولی علامت == برای مقایسه دو مقدار بکار می رود یعنی هنگام تست شرط ها باید از == استفاده کنید.

نکته: دستورات مقایسه ای در C++ به صورت زیر است.

کوچکتر یا مساوی با	<=	کوچکتر از	<
بزرگتر یا مساوی با	>=	بزرکتر از	>
نا مساوی با	!=	مساوی با	==

استفاده از AND و OR منطقی در برنامه

در برنامه ممکن است در جایی بخواهیم در صورت برابری دوشرط با هم کدی را اجرا نماییم، مثلا در صورت برابری متغیر **a** با **b** و متغیر **c** با **d** دستوراتی را اجرا نماییم.خوب این دستور را می توان با دو دستور **if** تو در تو نوشت مانند کد زیر:

```
if(a == b)
{
     if(c == d)
     {
        std::cout << "Ok" << std::endl;
     }
}
```

اما راه حل بهتر که باعث خواناتر شدن کد برنامه می شود استفاده از دستور **AND** منطقی است،این دستور به صورت علام && بین دو دستور مقایسه ای قرار می گیرد و کارایی آن نیز به این صورت است که در صورت برابری دو مقدار مقایسه ای در دو طرف آن شرط اجرا می شود. حالا دستور بالا را بوسیله استفاده از **AND** منطقی و با یک دستور **if** می نویسیم:

```
if(a == b && c == d)
{
     std::cout << "Ok"  << std::endl;
}
```

حال فرض کنید که می خواهیم کدی بنویسیم که در صورت برابری **a** با **b** و یا **c** با **d** کاری را انجام دهد. در حالت معمول برای انجام این کار باید به صورت زیر عمل کنید:

```
if(a == b)
{
     std::cout << "Ok" << std::endl;
}
else
{
     if(c == d)
     {
          std::cout << "Ok" << std::endl;
     }
}
```

دستور **OR** منطقی در صورت برابری هر کدام و یا دو طرف شرط باعث اجرای آن می شود ، و شکل دستوری آن علامت || است.حال با استفاده از دستور **OR** منطقی این کد را بازنویسی می کنیم.

```
if(a == b || c == d)
{
	std::cout << "Ok" << std::endl;
}
```

می بینید که کد جدید نوشته شده چقدر ساده تر و خواناتر است.

دستور using

دستور **using namespace std;** مشخص میکند از فضای نام **std** استفاده شود، وگرنه باید در ابتدای هر نامی که از این فضای نام استفاده میکنید، یک **std::** بگذارید، مثلاً بجای **cout** باید بگذارید **std::cout** و این خسته کننده خواهد بود، اما اگر زمانی تداخل نام پیش آید مجبور به استفاده از این پیشوند خواهید بود(مثلاً وقتی یک متغیر با همین نام دارید).
برای راحتی کار و کدنویسی کمتر بوسیله دستور **using** که به صورت زیر استفاده می شود نام توابع را ساده کرده و دیگر نیازی به استفاده از نام کامل آن در برنامه نمی باشد، در مثال زیر دیگر نیازی به استفاده از عبارت **std::** در دستورات **cout** و **endl** نیست.

```
#include "stdafx.h"
#include <iostream>

using std::cout;
using std::endl;

int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;

	a=20;
	b=30;

	cout << "a = " << a << endl;
	cout << "b = " << b << endl;

	return 0;
}
```

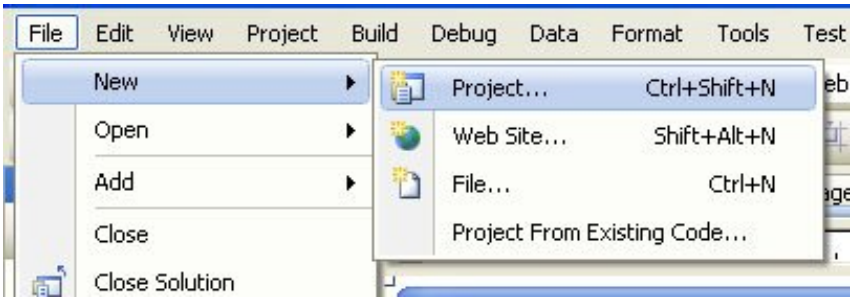
آموزش مقدماتی MFC

بهزاد جناب

فصل دوم

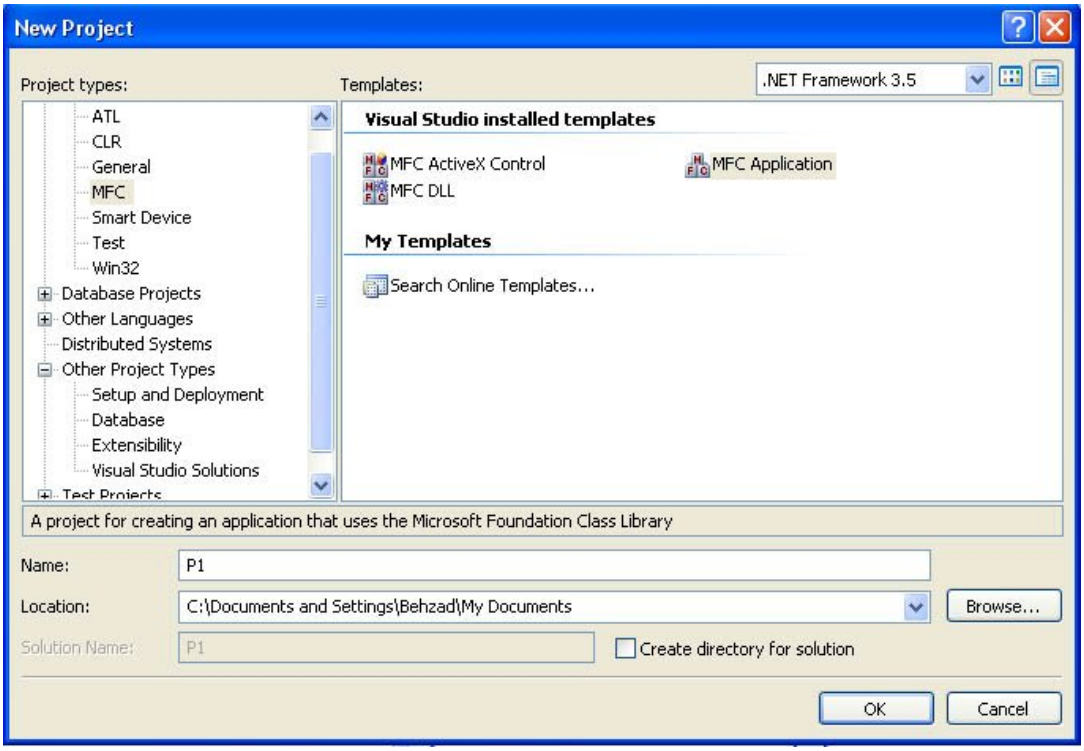
اولین پروژه با کلاسهای بنیادی مایکروسافت (MFC)

اولین برنامه **MFC** که با **C++** می نویسیم یک برنامه ساده است که دارای یک دکمه برای بستن برنامه می باشد. برای ایجاد پروژه از منوی **File** گزینه **Project** را انتخاب کنید.

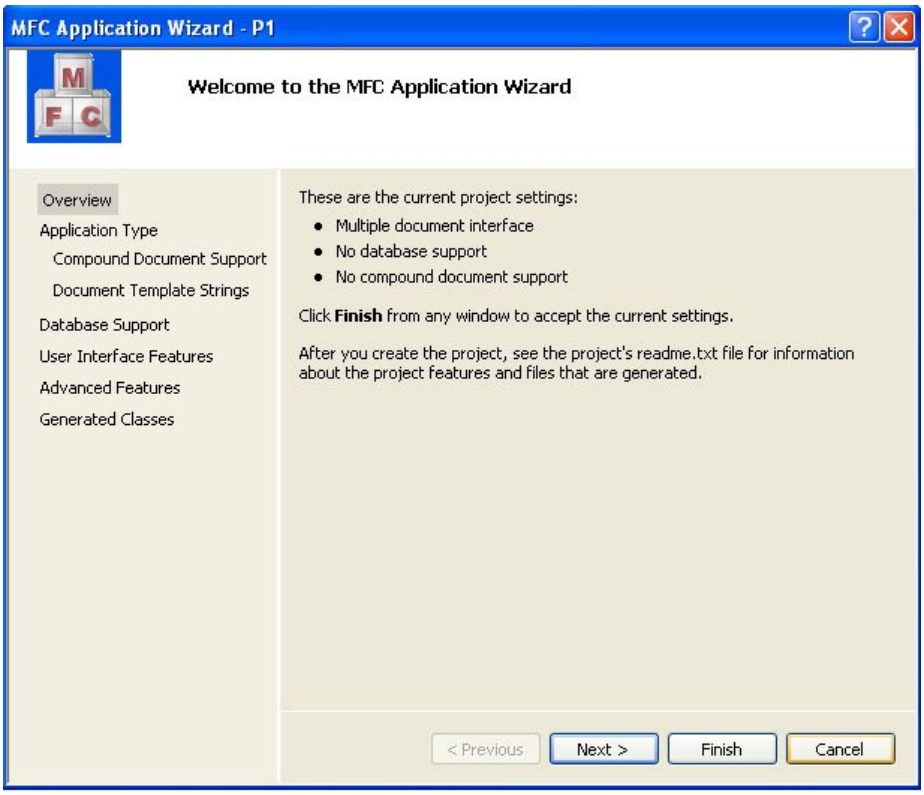


در پنجره به نمایش در آمده در سمت چپ آن گزینه **MFC** را انتخاب و سپس در سمت راست بر روی گزینه **MFC Application** کلیک نمایید، نام برنامه را **P1** بگزارید و بر روی **Ok** کلیک کنید.

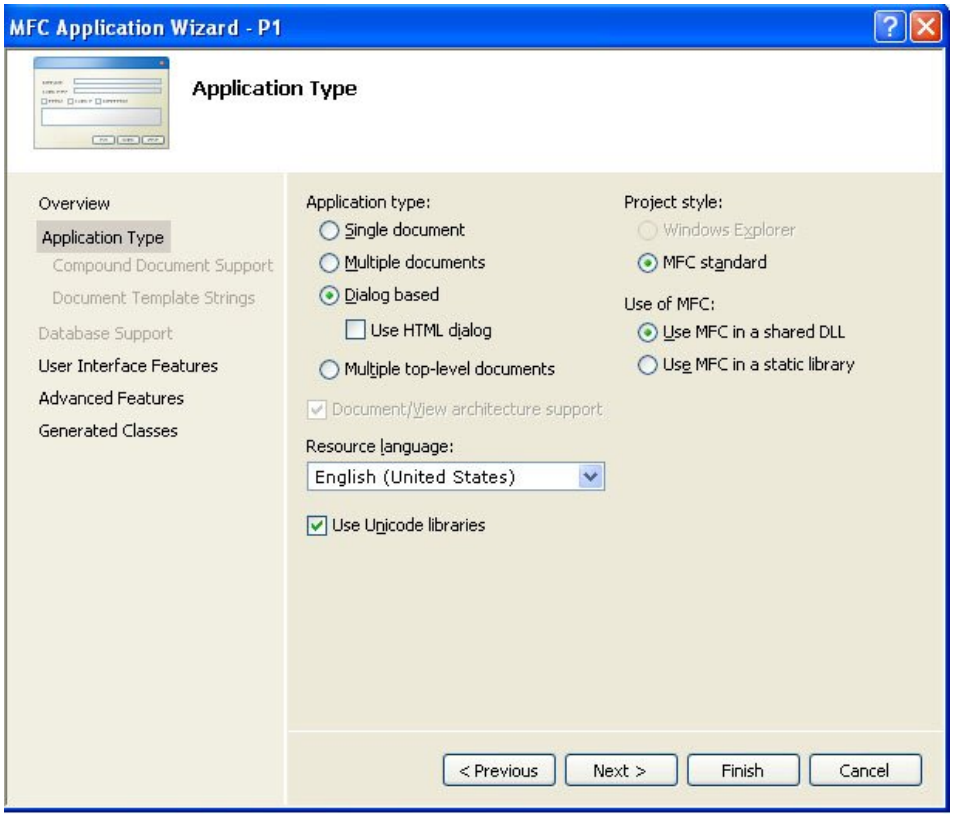
نکته: اگر چک باکس **Create directory for solution** را انتخاب کنید، یک فولدر برای راه حل در مسیری که برای فولدر پروژه مشخص کرده اید ساخته میشود، سپس فولدر پروژه در آن ساخته خواهد شد. در اینجا این چک باکس را انتخاب نکنید، چرا که یک پروژه بیشتر نخواهیم ساخت.



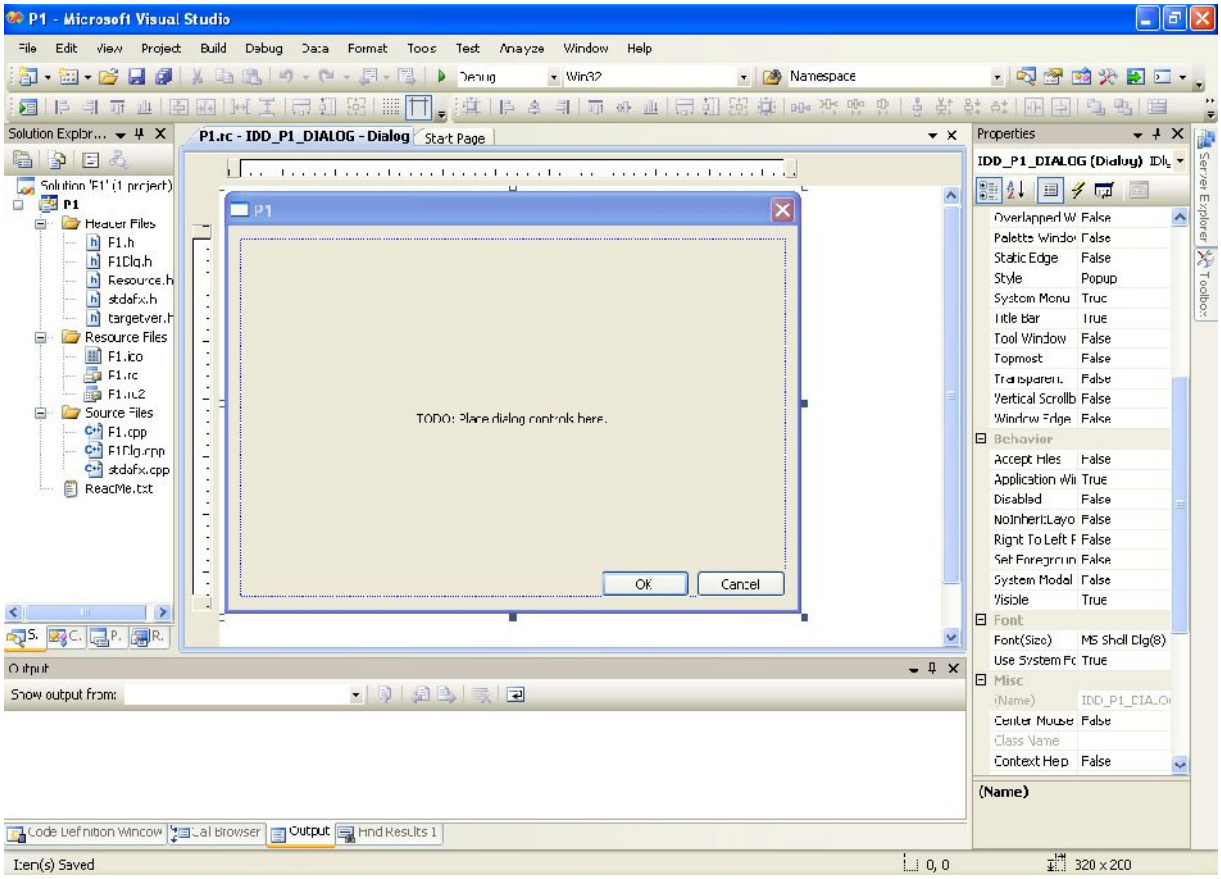
در بنجره بعدی چون قصد داریم تغییراتی در نوع تنظیمات پروژه اعمال کنیم بر روی گزینه **Next>** کلیک می نماییم.



حالا عنوان **Application Type** برنامه را به **Dialog based** تغییر داده و بر روی کزینه **Finish** کلیک می کنیم تا برنامه ساخته شود.



بعد از پایان کار به محیط ویژوال استدیو باز می گردید و پنجره برنامه شما نشان داده می شود(شکل زیر).



توضیحاتی در مورد پنجره های پروژه

حالا پروژه شما ساخته شده است. قبل از اینکه بخواهید به برنامه نویسی دست بزنید باید با محیط کار با پروژه آشنا شوید. چند پنجره مهم مربوط به پروژه در محیط ویژوال استدیو وجود دارند که توضیحات آن در زیر آمده است. چنانچه هر کدام از این پنجره ها را در محیط ویژوال نمی بینید می توان از منوی **View** آنها را به برنامه اضافه نمایید.

پنجره های سمت چپ

۱. پنجره **Solution Explorer** که فایل های اضافه شده به پروژه را در آن مشاهده میکنید. البته برای دسته بندی کردن، **VS** چند پوشه را برایتان ایجاد کرده است که در زیر نام پروژه مشاهده میکنید (این پوشه ها مجازی هستند و روی دیسک سخت قرار ندارند). **VS** به این پوشه ها فیلتر میگوید، چرا که انواع فایل را فیلتر میکند.
۲. پنجره **Class View** که توابع و متغییر های عضو کلاسهای برنامه در آن قرار دارد و توسط آن می توان آنها را مدیریت کرد.
۳. پنجره **Add Resource** که منابع پروژه نظیر منوها، آیکونها، دیالوگها و ... در آن قرار دارد.

پنجره های سمت راست

۱. پنجره **Properties** که با انتخاب هر اشیائی برنامه مانند منابع سیستم(منوها، پنجره ها و ...) می توانید خواص آنها را در این قسمت مشاهده کرده و یا تغییر دهید.
۲. پنجره **Toolbox** که اشیائی مانند دکمه فرمان، متن سابت و ... برای ایجاد یک پنجره دیالوگ در آن قرار دارد.

نکته: چنانچه برای برنامه نویسی نیاز به فضای بیشتری دارید می توانید از خاصیت پنهان شدن خودکار پنجره ها استفاده کنید تا در موارد بیکاری مخفی بمانند و به این ترتیب فضای کاری بیشتری در اختیارتان قرار خواهد گرفت. برای فعال کردن این خواصیت بر روی سر تیتر پنجره ها راست-کلیک کرده و گزینه **Auto Hide** را تیک بزنید.

شروع برنامه نویسی در محیط ویژوال استدیو به روش MFC

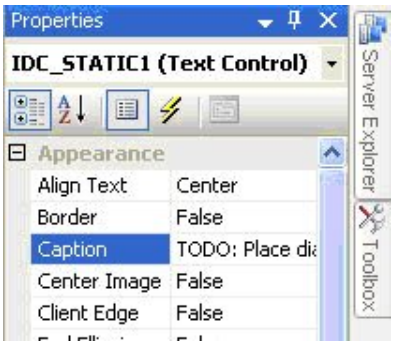
در پنجره برنامه به طور پیشفرض یک پیام **TODO** و دو دکمه فرمان وجود دارد ، حالا برنامه را با استفاده از انتخاب گزینه **Start Without Debugging** از منوی **Debug** و یا با فشردن کلیدهای میانبر **Ctrl+F5** اجرا نمایید.خروجی برنامه به شکل زیر خواهد بود که با فشردن هر کدام از دکمه های **Ok** یا **Cancel** توسط کابر از برنامه خارج می شود.

حالا به محیط ویژوال استدیو بر می گردیم و می خواهیم تغییراتی در شکل ظاهری پنجره برنامه اعمال کنیم.

۱. دکمه فرمان **Ok** را از روی پنجره حذف نمایید. برای اینکار با کلیک بر روی شکل دکمه **Ok** آن را انتخاب کرده و سپس کلید **Delete** را از روی کی بورد می فشاریم تا حذف شود و یا با راست کلیک کردن بر روی آن و انتخاب گزینه **Delete** از منوی باز شده نیز می توان این کار را انجام داد.

۲. پیام **TODO** را که در وسط پنجره مشاهده می کنید به بالای آن انتقال دهید. برای این کار کافی است آن را نتخاب کنید و در حالی که دکمه چپ ماوس را پایین نگه داشته اید آن را به بالا یا هر جای دیگر دلخاه انتقال دهید.

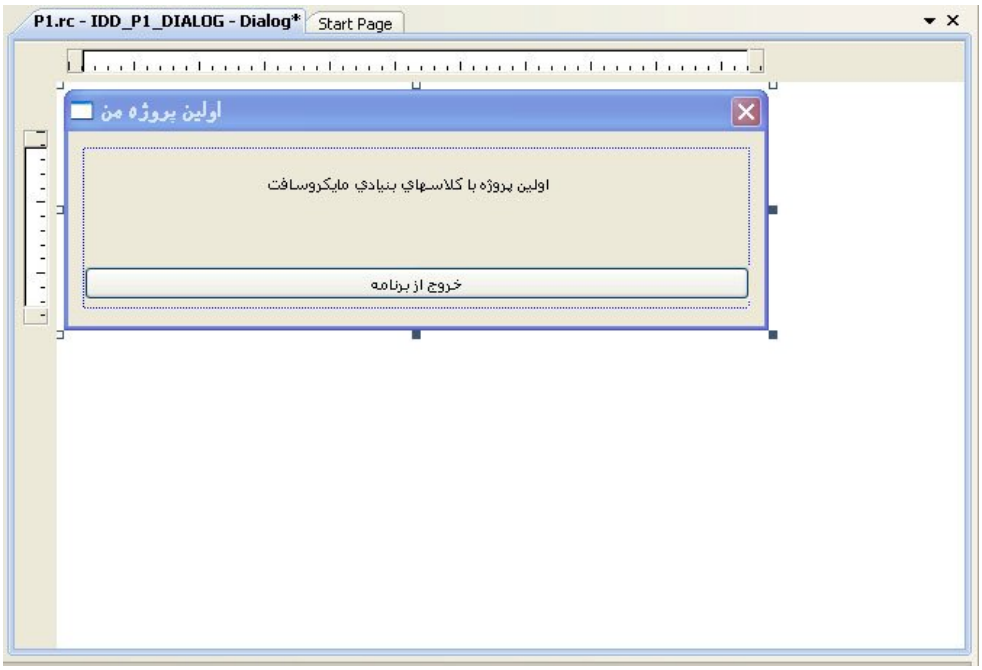
۳. متن پیام **TODO** را تغییر دهید. برای تغییر متن پیام ابتدا آن را انتخاب کرده سپس در پنجره **Properties** سمت راست محیط ویژوال استدیو اطلاعات جلوی فیلد **Caption** را به متن مورد نظر یعنی اولین پروژه با کلاسهای بنیادی مایکروسافت تغییر دهید.



۴. متن بالای پنجره برنامه که **P1** نوشته شده را تغییر دهید. برای تغییر متن بالای پنجره شبیه بالا ابتدا آن را انتخاب کرده سپس در پنجره **Properties** سمت راست محیط ویژوال استدیو اطلاعات جلوی گزینه **Caption** را به متن مورد نظر یعنی اولین پروژه من تغییر دهید.

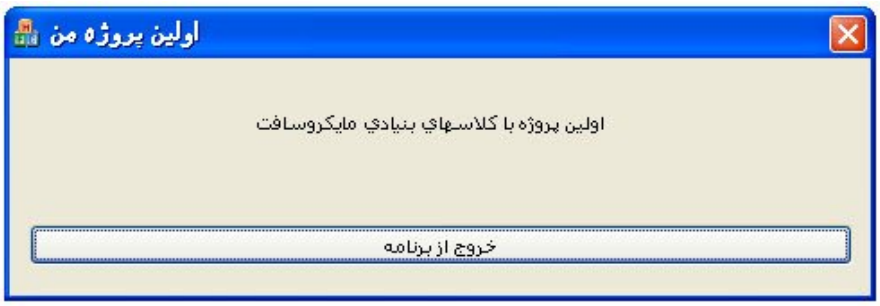
۵. اندازه پنجره برنامه را کوچک کنید. با انتخاب پنجره برنامه دستگیره های تغییر اندازه آن فعال می شود، سپس با استفاده از آن دستگیره ها پنجره را کوچک نمایید.

۶. دکمه **Cancel** موجود در روی پنجره را کشیده تر کنید و نام آن را نیز با استفاده از عنوان **Caption** به خروج از برنامه تغییر دهید. در نهایت پنجره برنامه چیزی شبیه به شکل زیر خواهد شد.



حالا برنامه را ذخیره و سپس اجرا نمایید.پس از اجرا خواهید دید که تمامی نوشته های فارسی شما به علامت ؟ تبدیل شده اند اما دلیل آن چیست؟ برای استفاده از زبان فارسی در این پنجره باید تنظیمات زبان فارسی را برای آن انتخاب نمایید.برای انجام این کار از پنجره سمت چپ محیط ویژوال استدیو سربرگ **Resource View** را انتخاب کنید، سپس با کلیک بر روی علامتهای مثبت موجود که به صورت درختی هستند با جستجوی محتوای آن به دنبال عبارت **IDD_P1_DIALOG** بگردید و آن را انتخاب کنید. (مواضب باشید که به هنگام انتخاب آن بر روی آن دابل کلیک نکنید). سپس در فیلد **Language** در قسمت **Properties** زبان را **Farsi** قرار دهید.

حالا دو باره برنامه را اجرا کنید(اگر برنامه تان را نبسته اید حتما قبل از اجرای مجدد آن را ببندید)، می بینید که مشگل حل شده است.



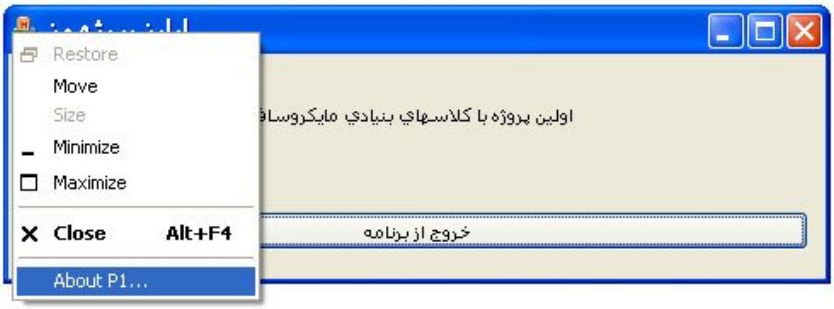
برنامه را بوسیله گزینه **Save All** از منوی **File** ذخیره کنید و از محیط ویژوال استدیو خارج شوید ، حالا ویژوال استدیو را دوباره اجرا نمایید و برنامه تان را بوسیله گزینه **Open** از منوی **File** باز کنید. اما حالا پنجره ای که طراحی کرده اید را نمی بینید، برای مشاهده پنجره برنامه باید از پنجره سمت چپ محیط ویژوال استدیو از سر برگ **Resource View** بر روی عبارت **IDD_P1_DIALOG[Farsi]** دابل کلیک نمایید.

اضافه کردن دکمه های حداقل و حداکثر

برای اضافه کردن دکمه های حداقل و حداکثر به میله عنوان پنجره برنامه ابتدا پنجره دیالوگ برنامه را انتخاب کنید، سپس در پنجره **Properties** محتوای دو فیلد **Maximize Box** و **Minimize Box** را از **False** به **True** تقییر دهید. عبارت **False** به معنای نادرستی و **True** به معنای درستی است.

پنجره توضیحات برنامه

در برنامه شما پنجره دیگری نیز به نام **About** وجود دارد که با راست کلیک کردن بر روی عنوان پنجره و انتخاب گزینه **About P1...** نمایش داده می شود.



در این پنجره معمولا توضیحاتی درباره نویسنده ، سال تولید و نصخه برنامه قرار داده می شود.

شما یا باید این پنجره را از برنامه خود حذف نمایید و یا به شکلی صحیح آنرا ویرایش کنید. برای انتخاب آن از پنجره سمت چپ در سر برگ **Resource View** عبارت **IDD_ABOUTBOX** را انتخاب کنید(دابل کلیک نکنید). برای فارسی سازی این پنجره شبیه بالا عمل می کنیم ،از پنجره **Properties** در سمت راست محیط ویژوال استدیو و از فیلد **Language** زبان را **Farsi** قرار دهید.در این مرحله با دابل کلیک کردن بر روی عبارت **IDD_ABOUTBOX** این پنجره نماین می گردد، حالا آن را به شکل دلخواه ویراش کنید.

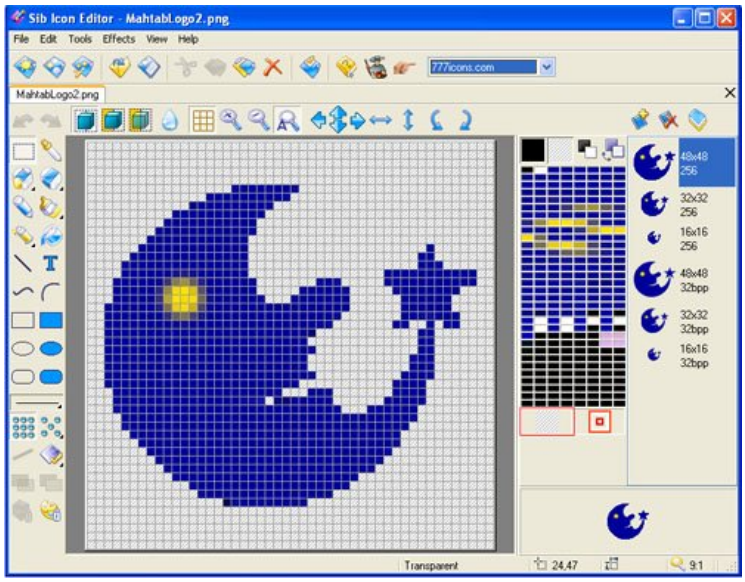
آیکونهای پروژه

اگر به آیکون برنامه که در کوشه چپ، بالای پنجره برنامه است نگاه کنید خواهید دید که این آیکون از سه مکعب با حروف F،M و C تسگیل شده است. این سه حرف معرف کلاسهای بنیادی مایکروسافت (Microsoft Foundation Classes) هستند و میگویند که این برنامه با استفاده از کتابخانه های C++ ساخته شده است، ولی شما که میل ندارید این مطلب را همه جا جار بزنید؟ پس باید آیکون برنامه را عوض کنیم. برای تغییر شکل آیکونهای برنامه باید از پنجره سمت چپ ویژوال استدیو سربرگ Solution Explorer را انتخاب کرده ،سپس بر روی فایل P1.ico دابل کلیک نمایید تا پنجره آیکونهای برنامه نمایش داده شود. حالا می توانید آنها را ویرایش کنید. اما در طراحی آیکون برنامه اندازه های مختلفی نظیر ۴۸*۴۸ ، ۳۲*۳۲ و ۱۶*۱۶ با چند نوع رنگ بندی مانند ۴ ، ۸ و ۲۴ بیت وجود دارد. اما شاید بپرسید دلیل وجود این همه آیکون در یک برنامه چیست؟ در واقع که هرکدام از این آیکونها بسته به شرایط در محل خواص خودش در برنامه مورد استفاده قرار می گیرد مثلا سایز ۱۶*۱۶ در سیستم ترای ویندوز یا بالای تیتر پنجره استفاده می شود و سایز ۳۲*۳۲ در پنجره About برنامه، همچنین حالات مختلف رنگبندی که مثلا وقتی ویندوز در حالت Safe Mode بالا می آید چون سیستم با حداقل امکانات راه اندازی می شود معمولا عمق رنگبندی صفحه پایین است و برنامه از آیکونهایی با رنگبندی های پایینتر مثل ۴ و ۸ بیت استفاده می کند. اما شما که قصد دارید یک برنامه حرفه ای تولید کنید باید برای تمامی اندازه ها و انواع رنگبندی، آیکونهایی یک شکل طراحی کنید تا در تمامی حالات برنامه ما یکنواخت و زیبا باشد.

تولید آیکونهای برنامه توسط نرم افزار Sib Icon Editor

اما طراحی آیکونهای برنامه توسط ادیتور محیط ویرایش کار زمانبر و سختی خواهد بود و در نهایت هم نمی توان آیکونهای یک دست و زیبایی را ایجاد کرد. بنابراین پیشنهاد من این است که طرح اصلی آیکون برنامه تان را در نرم افزار هایی مانند فتوشاپ یا AAA logo طراحی نمایید و خروجی کارتان را به صورت یک عکس با فرمت PNG و با ابعاد ۲۵۶*۲۵۶ ذخیره نمایید، سپس بوسیله نرم افزار Sib Icon Editor به یک آیکون تبدیل کنید.

عکس محیط نرم افزار



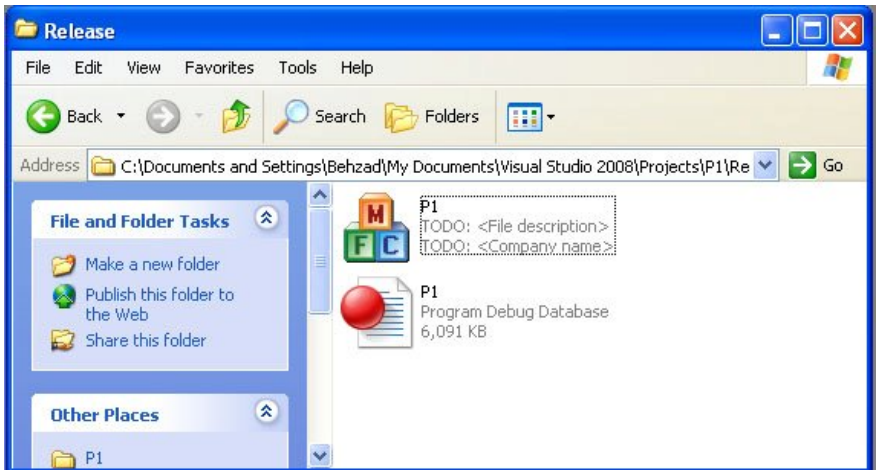
پس از دانلود نرم افزارSib Icon Editor از آدرس http://sibcode.com/downloads/icon-editor.exe با حجم تقریبی ۲ مگ، آنرا اجرا کرده و گزینه Create a New Icon From Image File را انتخاب نمایید، سپس عکس PNG با ابعاد ۲۵۶*۲۵۶ را که قبلا طراحی کرده اید باز نمایید و پس از اعمال تغییرات دلخواه در ابعاد مختلف آیکونهای با کلیک کردن بر روی Ok پروژه را ساخته و در نهایت آیکون را با انتخاب گزینه Save as... با فرمت ico ذخیره نمایید، خروجی برنامه ما یک مجموعه آیکون با ابعاد و رنگبندی مختلف است که در یک فایل ذخیره شده است. برای تغییر آیکون پروژه به شاخه منابع پروژه با نام res بروید و نام آیکون پروژه (که همان نام پروژه است) را در Clipboard کپی

کرده و نام فایل آیکونی که ساخته ایم با دستور **Rename** و سپس **Paste** به همان نام تغییر دهید، سپس آیکون جدید را که همنام با آیکون قدیمی است در محل آن کپی نمایید تا جایگزین آن شود.

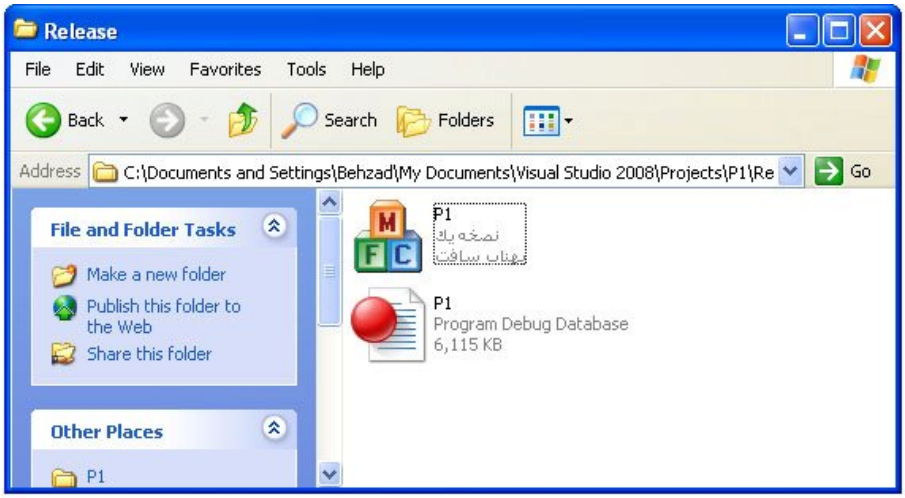
هشدار: در هنگام جایگزین کردن آیکون جدید پروژه حتما پروژه بسته باشد و برای اطمینان بیشتر از محیط ویژوال خارج شوید در غیر اینصورت ممکن است به فایل ذخیره منابع پروژه شما آسیبی وارد شود و پروژ کلا از دست برود.

تغییر مشخصات فایل P1.exe :

بعد از کامپایل برنامه اگر فایل اجرایی پروژه تان را بوسیله **My Computer** ویندوز تماشا کنید خواهید دید که مشخصاتی که با کلمه **TODO** شروع می شود در زیر نام فایل وجود دارد، این مشخصات در واقع باید نشان دهنده شرکت تولید کننده و شماره نصخه برنامه شما و یا از این قبیل باشد ولی چگونه می شود آنها را به شکلی زیبا تغییر داد و ویرایش نمود تا مشخصات شما و برنامه تان را نشان دهد؟



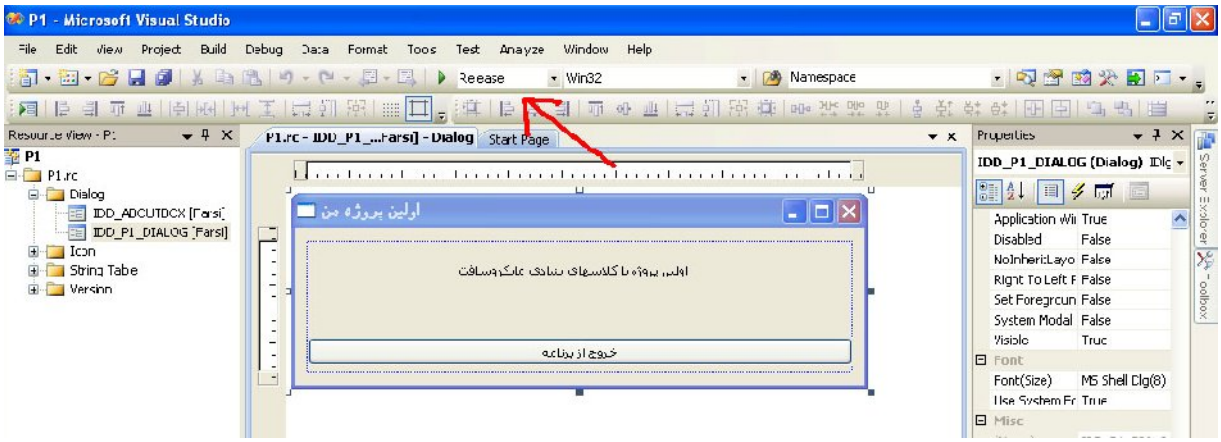
برای این کار از پنجره سمت چپ ، سر برگ **Resource View** را انتخاب کرده و بر روی عبارت **VS_VERSION_INFO** دابل کلیک کنید، سپس اطلاعات این بخش نمایش داده می شود و شما می توانید آنها را به طور دلخواه ویرایش نمایید.چنانچه قصد دارید مشخصات فایل خود را به زبان فارسی ویرایس نمایید یک بار بر روی عبارت **VS_VERSION_INFO** کلیک نمایید سپس از پنجره **Properties** فیلد **Language** را به **Farsi** تغییر دهید.حالا مشخصات شما در پنجره **My Computer** نشان داده می شود.



کامپایل نهایی

حالا فرض کنید که قصد دارید این برنامه را به عنوان اولین برنامه ای که با C++ نوشته این به خانه دوستتان برده و به او نشان دهید. به طور پیش فرض هر برنامه ای در C++ بر روی حالت **Debug** کامپایل می شود که برای اشکال زدایی برنامه در هنگام تولید از آن استفاده می شود و برنامه ای که در این حالت کامپایل شود غیر از بر روی کامپیوتر خودتان بر روی هیچ کامپیوتر دیگری اجرا نخواهد شد، اما برای تهیه پروژه نهایی ابتدا باید روش کامپایل برنامه را بر روی حالت **Release** قرار دهید و سپس آنرا کامپایل نمایید.

(به عکس زیر توجه کنید)



اما برنامه شما هنوز هم یک برنامه مستقل نیست، چون در این برنامه از توابعی استفاده شده است که در کتابخانه های(**DLL**) مختلفی وجود دارند و شما می بایست آن کتابخانه ها را در پوشه ای که برنامه وجود دارد و یا پوشه سیستم ویندوز کپی نمایید تا برنامه به توابع آن که توسط برنامه فراخانی می شود دسترسی داشته باشند. اما در این پروژه کتابخانه ها مورد نیاز به صورت اشتراکی(**Shared DLL**) استفاده می شوند و باید برای اجرای مستقل آن به این صورت یک برنامه نصب بسازیم. اما فعلا برای راحتی کار و حل این مشگل می توانید کتابخانه های مورد نیاز برنامه را به آن پیوست کنید که در این حالت کتابخانه ها به همراه برنامه کامپایل می شود(**Static Library**)، در این صورت کمی به حجم برنامه اضافه می شود ولی برنامه شما دیگر به صورت یک فایل مستقل در هر کامپیوتری بدون نیاز به نصب (**Portable**) اجرا خواهد شد.

نکته: البته لینک کردن کتابخانه ها به شکل **static** معایبی هم دارد که باعث ایجاد محدودیت هایی می شود و معمولا از این روش در پروژه های حرفه ای استفاده نمی شود.

پروژه های MFC نیازمند دو دسته کتابخانه هستند:

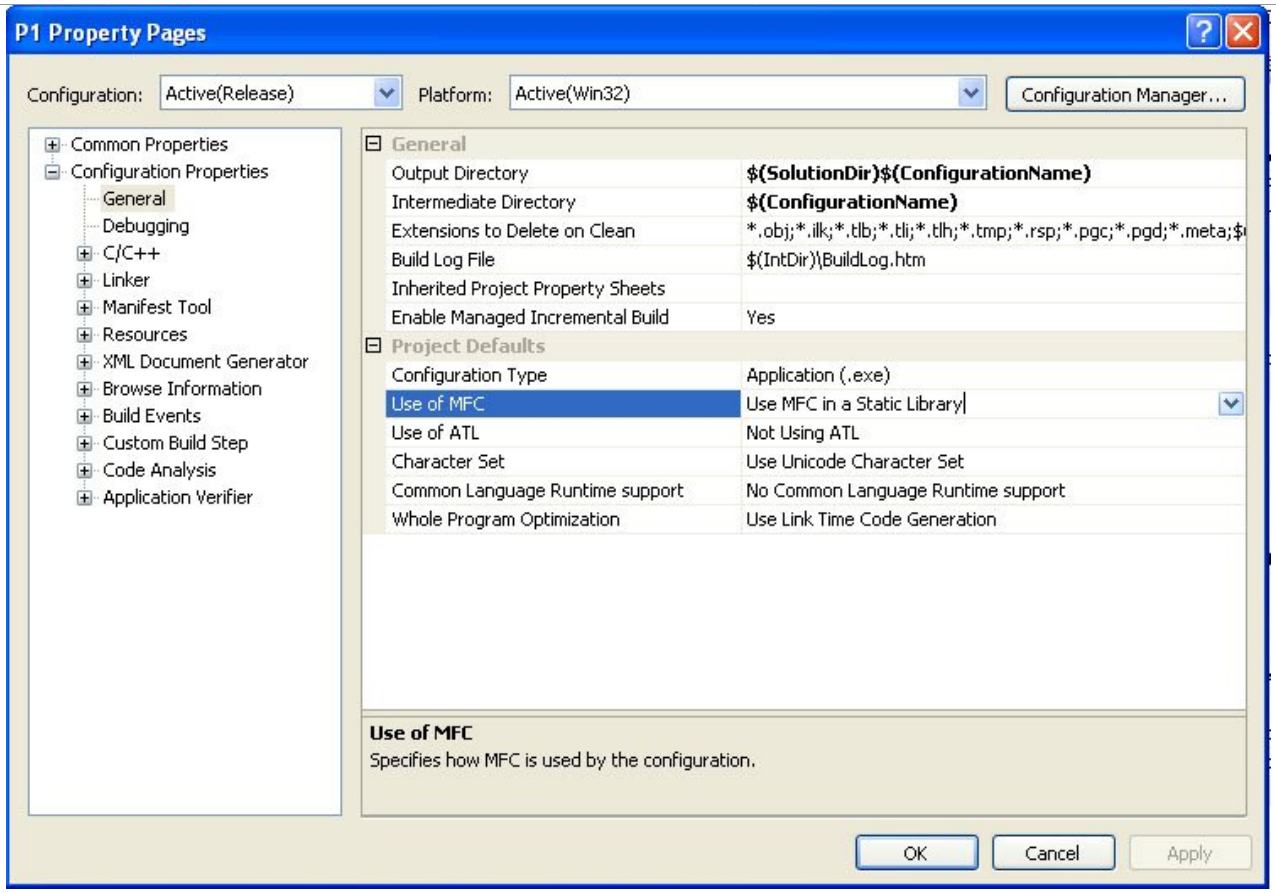
۱- کتابخانه های **MFC**

۲- کتابخانه های **C run-time**

اگر پروژه ای قرار هست کلا بدون نیازمندی به **dll** ای و به صورت قابل حمل(**Portable**)ساخته شود باید هر دو مورد فوق به صورت **static** لینک شوند.

لینک کردن کتابخانه های MFC

برای انجام این کار از منوی **Project** گزینه **P1 Properties...** را انتخاب کنید. سپس از پنجره نمایش داده شده ، سمت چپ از عبارت **Configuration Properties** گزینه **General** را انتخاب نمایید و از تنظیمات آن که در سمت راست نمایش داده شده، فیلد **Use of MFC** را از **Use MFC in a Shared DLL** به **Use MFC in a static library** تغییر دهید و بر روی **Ok** کلیک کنید.



لینک کردن کتابخانه های C run-time

از منوی **Project** گزینه **Properties** را انتخاب کنید و در سمت چپ پنجره باز شده اطلاعات فیلد موجود در مسیر

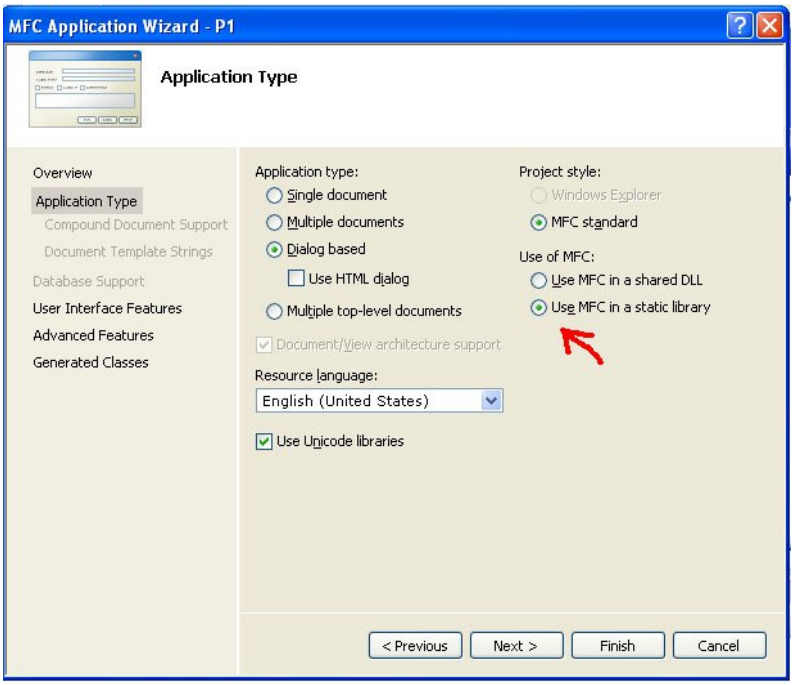
Configuration Properties -> C/C++ -> Code Generation -> Runtime library

را به **/MT** تغییر داده و بر روی دکمه **Ok** کلیک کنید.

برای اطمینان بیشتر از اعمال تغییرات در کامپایل برنامه و ایجاد یک پروژه کاملا مستقل و قابل حمل(**Portable**)، پروژه قبلی را بوسیله گزینه **Clean Solution** از منوی **Build** پاکسازی کرده و دوباره برنامه را کامپایل(**Ctrl+F5**) می نمایید. اگر به دنبال فایل اجرایی برنامه تان هستید نسخه نهایی و اجرایی پروژه شما با نام **P1.exe** در پوشه **My Documents** ویندوز در مسیر **Visual Studio 2008\Projects\P1\Release** است. حالا می توانید با خیال راحت برنامه را برای اجرا به یک کامپیوتر دیگر منتقل کنید.

نکته: اما روش دیگری نیز برای پیوست کردن توابع **MFC** به برنامه در هنگام ساخت پروژه جدید وجود دارد .در این روش هنگام ساخت یک پروژه جدید تنظیمات برنامه را بر روی گزینه **Use MFC in a static library** قرار میدهیم تا هنگام کامپایل و اجرایی شدن برنامه کتابخانه های **MFC** مورد نیاز به آن پیوست شوند.

به عکس زیر دقت کنید.



راه حل مناسب برای استفاده از پروژه

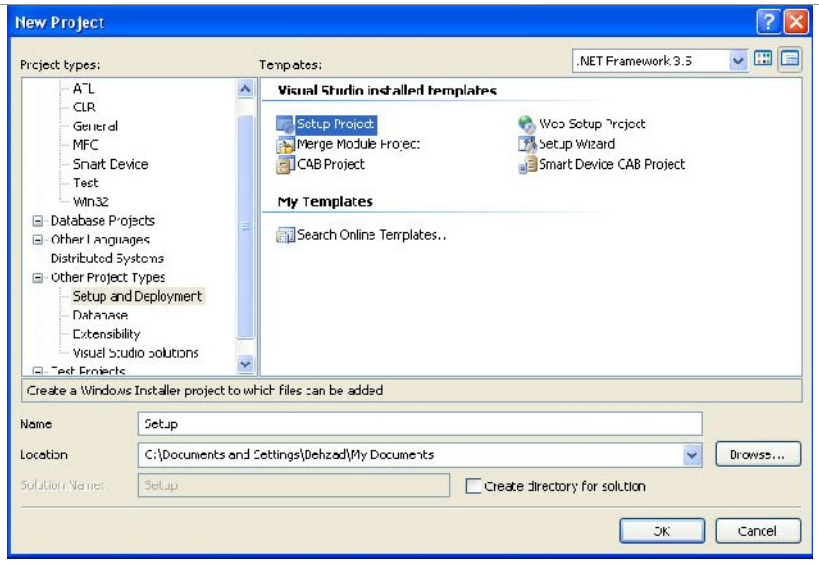
همان طوری که در بالا گفته شد چسباندن کتابخانه های مورد نیاز برنامه به آنها باعث ایجاد محدودیت هایی در پروژ می شود که این روش در پروژه های سنگین و با حجم بالا کار صحیحی نیست. ولی یک نکته در اینجا وجود دارد و آن هم حجم فایل **Setup** است، مثلا فایل اجرایی یکی از پروژه های من در حالت **Use MFC in a Shared DLL** حجمی حدود ۳۴۴ کیلوبایت داشت و وقتی توسط ویژوال استدیو یک برنامه نصب برای آن ساختم حجم فایل نصبی تقریبا ۵/۵ مگ شد، همچنین حجم فایل نصب برنامه با نرم افزار **InstallShield 2009** بیشتر از ۹/۵ مگ شد، در حالی که وقتی پروژه را به صورت **Use MFC in a static library** در آوردم حجم فایل اجرای به ۴۳۸ کیلو بایت تغییر یافت و حجم فایل نصبی با ویژوال استدیو نیز فقط ۳۵۹ کیلوبایت شد(چون در این حالت نیازی افزودن مرج مودولهای نیست).

پس استفاده از روش دوم برای برنامه های کوچک به دلیل افزایش بسیار زیاد حجم فایل نصبی توصیه نمی شود. اما روش دوم که همان ساخت یک برنامه نصب (**Setup**) است را به شما آموزش می دهیم. در این روش خود برنامه نصب پروژه کتابخانه های مورد نیاز برنامه را در سیستم هدف کپی میکند.

نکته: قبل از تولید **Setup** پس از اشکال زدایی نهایی پروژه و اطمینان از صحت عملکرد برنامه تان آنرا در حالت **Release** کامپایل نمایید تا فایل اجرایی(**exe**) پروژه برای ساختن **Setup** تولید شود.

تولید یک برنامه نصب برای پروژه بوسیله ویژوال استدیو (Setup Project)

برای ساخت برنامه نصب باید یک پروژه از این نوع بسازیم بنابراین از منوی**File** گزینه **New** و سپس **Project** را انتخاب کنید و در پنجره باز شده از قسمت **Other Project Types -> Setup and Deployment** پروژه ای از نوع **Setup Project** را انتخاب نمایید و پس از انتخاب نام برای این پروژه بر روی **Ok** کلیک کنید تا ساخته شود.



سپس برای ساختن برنامه نصب مراحل زیر را دنبال کنید

۱. برای اضافه کردن فایل اجرایی(exe) به پروژه از منوی **Project** گزینه **Add** و سپس **File** را انتخاب کنید. سپس فایل **exe** را از شاخه **Release** پروژه انتخاب کرده تا به آن اضافه شود.

۲. در مرحله بعدی برای افزودن کتابخانه های مورد نیاز برنامه از منوی **Project** گزینه **Add** و سپس **Merge Module** را انتخاب کنید و دو فایل با نامهای **Microsoft_VC90_CRT_x86.msm** و **Microsoft_VC90_MFC_x86.msm** را انتخاب کرده و بر روی **Open** کلیک کنید.

۳. برای تغییر مسیر پیش فرض برای نصب برنامه در سیستم هدف بر روی **Application Folder** کلیک کرده و از پنجره **Properties** گزینه **DefaultLocation** را تغییر دهید. مثلا چنانچه قصد دارید برنامه شما در مسیر **C:\Program Files\Mahtab** نصب شود باید اطلاعات این فیلد را به **[ProgramFilesFolder]\Mahtab** تغییر دهید.

۴. برای اینکه برنامه ما یک آیکون میانبر در صفحه اصلی داشته باشد پوشه **Application Folder** را انتخاب کرده و از سمت راست پنجره بر روی نام برنامه اجرایی راست-کلیک کرده و از منوی باز شده گزینه **Create Shortcut to...** را انتخاب می کنیم و پس از انتخاب نام مناسب برای آیکون میانبر با کشیدن آیکون آن آنرا به درون پوشه **User's Desktop** انتقال می دهیم. تا اینجا میانبر را ساخته ایم ولی شکل آیکون آنرا مشخص نکرده ایم.

۵. در این مرحله برای انتخاب شکل آیکون این میانبر آنرا انتخاب کرده و از پنجره **Properties** بر روی گزینه **Icon** کلیک کرده و **Browse** را انتخاب کنید. حال یک پنجره جدید با عنوان **Icon** باز می شود، برای تایین آیکون میانبر از آیکون خود فایل اجرایی(exe) استفاده می کنیم بنابراین بر روی دکمه **Browse** کلیک کنید و حالت نمایش فایلها را بر روی *.* بگذارید تا بتوانید فایل اجرایی پروژه را ببینید، حالا از پوشه **Application Folder** فایل اجرایی را باز و آیکون آنرا انتخاب نمایید.

۶. برای قرار دادن آیکون در قسمت **All Programs** ویندوز در قسمت **User's Programs Menu** یک آیکون به روش بالا قرار می دهیم. چنانچه قصد داشتید آیکونهای برنامه در یک پوشه قرار گیرند ابتدا با راست-کلیک کردن بر روی فولدر **User's Programs Menu** گزینه **Add** و سپس **Folder** را انتخاب کنید و یک پوشه در این قسمت بسازید سپس مراحل بالا را برای ساخت آیکون در داخل آن پوشه تکرار کنید.

۷. برای تایین آیکون برنامه در قسمت Add or Remove Program ویندوز نیز از سمت چپ محیط ویژوال در پنجره Solution Explorer نام پروژه را انتخاب کرده و از پنجره Properties بر روی گزینه AddRemoveProgramsIcon کلیک کرده و Browse را انتخاب کنید. بقیه مراحل کار شبیه بالا است.

۸. در همین قسمت(پنجره Properties) فیلد ProductName مشخص کننده نامی است که پس از نصب برنامه در قسمت Add or Remove Program ویندوز نمایش داده می شود. پس آنرا به گونه ای تغییر دهید که نشان دهنده پروژه ای باشد که بر روی سیستم هدف نصب کرده است و کابر برای حذف آن دچار سردرگمی نشود.

۹. برای ساختن برنامه Setup ابتدا حالت کامپایل برنامه را به Release تغییر دهید سپس از منوی Build گزینه Build Solution را انتخاب کنید. فایل Setup ساخته شده در شاخه Release پروژه قرار دارد.

نکته: برای اینکه برنامه شما پس از نصب بصورت خودکار با هر بار شروع ویندوز اجرا شود یک پوشه با نام Startup در قسمت User's Programs Menu بسازید و یک میانبر از برنامه تان را طبق گفته های بالا در آن قرار دهید.

به عکس زیر دقت کنید

نکته: برنامه های setup ساز از جمله پروژه setup خود Visual studio بخشی دارند تحت عنوان dependencies که وقتی فایل اجرایی را به پروژه اضافه می کنید خودش بررسی می کند و می گوید که چه dll هایی نیاز دارد و هنگام نصب اگر در سیستم نباشند خودکار نصب می شوند. ولی از آنجایی که پروژه setup خود Visual studio به قدرتمندی نرم افزارهایی مانند installshield نیست و امکانات کمی دارد شما می توان از این گونه نرم افزار ها نیز برای ساخت Setup پروژه خود استفاده کنید.

درباره Microsoft Visual Studio و MSDN

برای نوشتن و کامپایل و اشکال زدایی برنامه های خود، باید از یک ابزار برنامه نویسی کمک بگیرید. به این ابزار، IDE (Integrated Development Environment) یا محیط یکپارچه(مجتمع) تولید میگویند، چراکه ابزارهای مختلف موارد ذکر شده را در خود مجتمع کرده است. ویژوال استودیو که توسط مایکروسافت تولید شده است، یکی از مشهورترین IDE ها است که قابلیت نوشتن برنامه ها در C++، C# (سی شارپ)، ویژوال C++ ، ویژوال بیسیک و J# را میدهد. ما دراین آموزش از Visual Studio 2008 (یا به اختصار VS) استفاده خواهیم کرد.
اما MSDN (Microsoft Developer Network) در اصل راهنمای موجود در ویژوال استودیو است. MSDN هر راهنمایی که برای برنامه نویسی در ویژوال استودیو لازم داشته باشید را در خود دارد : راهنمای دستورات زبان، توابع، نحوه برنامه نویسی، نکات فنی، کدهای نمونه و... .

این موارد برای تمام زبانهایی که میتوانید با آنها در VS برنامه نویسی کنید وجود دارند. در حقیقت MSDN تا اینجا تنها راهنمایی است که چنین کامل و جامع است. شما در هر محیطی که بخواهید برنامه نویسی کنید، بازهم میتوانید از راهنمایی MSDN استفاده کنید. وقتی که با آن مواجه شوید، حتماً از اینکه همه چیز را دارد تعجب خواهید کرد! MSDN بدلیل حجم زیاد در اصل جدای از VS ارائه میشود و بعدا میتوانید آن را بعنوان Help به VS اضافه نمایید (شما میتوانید درست بعد از نصب VS هم اینکار را بکنید). اگر بسادگی بخواهیم بگوییم MSDN چیست، همان Help ای است که شما با زدن F1 در هر برنامه دیگری بدست می آورید. شما اصلاً نمیخواهد نگران چیزی باشید، فقط آن را بعد از VS نصب کنید تا بعداً از راهنمایی آن در جای مناسب استفاده کنید. البته MSDN از سایت شرکت مایکروسافت به آدرس www.msdn.microsoft.com هم قابل دسترسی است، اما وقتی شما میخواهید از آن استفاده کنید، اگر مجبور باشید هربار ده ها صفحه وب را مرور کنید تا به اطلاعات خود برسید، با زحمتی که وصل شدن به اینترنت و آوردن صفحه های وب به ما تحمیل میکند، اصلاً کار عاقلانه ای نیست! فقط وقتی که به جدیدترین اطلاعات نیاز دارید یا چیزی را در نسخه فعلی خود از MSDN نیافتید، بهتر است به سایت MSDN مراجعه کنید. برای کار با MSDN در محیط VS فقط کافی است روی موضوعی که میخواهید درباره اش راهنمایی بگیرید(مثلاً یک دستور که تایپ کرده اید) دکمه F1 را فشار دهید.

VS و MSDN بسته های نرم افزاری جدا از هم هستند، پس آنها را روی DVD یا چندین CD تهیه و براحتی نصب کنید.

گرفتن راهنمایی فوری از MSDN

برای گرفتن راهنمایی MSDN در مورد دستوراتی که در فایل وارد کرده اید، کافی است مکان نمای ورود از طریق صفحه کلید(که بشکل I هست) را بر روی دستور مورد نظر قرار دهید(یعنی فقط در بین یکی از کاراکترهای آن باشد) آنگاه F1 را بزنید (ممکن است MSDN با کمی تأخیر نمایش داده شود). اگر خطایی در هنگام کامپایل رخ دهد، برای گرفتن راهنمایی از MSDN در مورد آن خطا، ابتدا مکان نما را در تب Output روی خطی که عبارت 'error' در آن وجود دارد ببرید، سپس F1 را بزنید. البته در هر قسمتی از VS که باشید، میتوانید با زدن کلید F1 در مورد آن قسمت، از MSDN راهنمایی بگیرید.

چگونه در برنامه نویسی استاد شویم

بعد از دنبال کردن هر قسمت از این آموزش، خوب است که هر دستور را آزمایش کنید و تمام امکاناتی که ارائه میدهد را بررسی کنید(با استفاده از MSDN). اما اگر میدانید که یک قطعه کد را بخوبی فهمیده اید، حتماً لازم نیست آنرا امتحان کنید. آنچه که مهم است اینست که خودتان برنامه بنویسید. ابتدا هدف کاری که میخواهید بکنید را مشخص کنید، بعد ببینید چگونه باید این کار را با امکانات C++ انجام دهید و یک طرحی روی کاغذ یا ذهنتان داشته باشید، سپس شروع به کد نویسی و اشکال زدایی کنید. در حقیقت هرگاه میخواهید کاری را با کامپیوتر انجام دهید به دنبال پیدا کردن راهی باشید که آن را با C++ انجام دهید و سپس مراحل ذکر شده را دنبال کنید.

باز کردن فوری پروژه

هروقت که VS را اجرا میکنید، تب Start Page نمایش داده میشود، شما میتوانید با کلیک بر روی نام پروژه خود از درون این تب، پروژه را باز نمایید. (توجه کنید که VS بیشتر جاها به 'راه حل' هم 'پروژه' میگوید). اما اگر بخواهید میتوانید در منو File به قسمت Recent Projects بروید، سپس نام پروژه خود را انتخاب کنید. توجه کنید که در هردو روش، فقط چندتا از پروژه هایی که اخیراً باز یا ایجاد کرده اید نمایش داده میشوند، اگر نام پروژه خود را نیافتید باید از منوی File > Open استفاده کنید. پسوند فایل های راه حل، sln میباشد.

آموزش مقدماتی MFC

بهزاد جناب

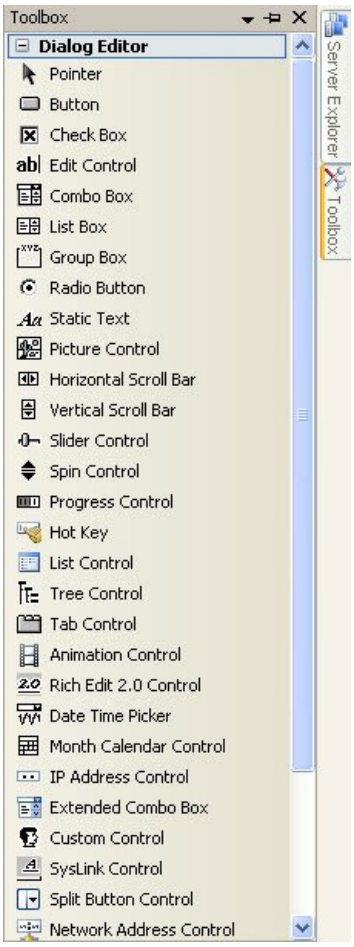
فصل سوم

کنترل های اصلی ویندوز

ویندوز دارای چندین کنترل استاندارد، از جمله کنترل های لیست(**List**) ، لغزنده (**Slider**) ها ، میله های پیشرفت(**Progres Bar**) و از این قبیل است. در این فصل با تعدادی از اساسی ترین کنترل های ویندوز کار خواهیم کرد:

- متن ثابت (**static text**)
- جعبه ادیت (**edit box**)
- دکمه فرمان (**command button**)
- جعبه چک (**check box**)
- دکمه رادیویی (**radio button**)
- جعبه لیست باز شو (**drop-down list box**) یا جعبه ترکیبی (**combo box**)

استفاده از این کنترلها در **Visual C++** بسیار ساده است. این کنترل ها را می توانید. در پنجره کشویی سمت راست با عنوان **Toolbox** مشاهده کنید.(شکل زیر)



کنترل متن ثابت

کنترل متن ثابت معمولا برای ارایه اطلاعات به کابر بکار می رود. کاربر قادر به دستکاری متن این کنترل نیست و در واقع این یک کنترل فقط خواندنی است. اما متن این کنترل را از طریق کد برنامه می توان تغییر داد.

کنترل جعبه ادیت

کنترل جعبه ادیت به کاربر امکان میدهد تا متن را وارد کرده و یا آن را دستکاری کند. این کنترل یکی از اصلیترین ابزارهای دریافت اطلاعات از کاربر و برقراری ارتباط با وی است. این کنترل فقط متن را برمی گرداند هیچگونه اطلاعاتی درباره فرمت این متن منتقل نخواهد کرد.

کنترل دکمه فرمان

دکمه فرمان کنترلیست که کاربر می تواند بکمک آن عملی را انجام دهد. دکمه فرمان معمولا د ارای عنوانی است که عملکرد آنرا توضیح می دهد. عنوان دکمه فرمان می تواند تصویر یا ترکیبی از متن و تصویر باشد.

کنترل جعبه چک

جعبه چک کنترلیست که کاربر می تواند آن را فعال یا غیر فعال کند. با فعال یا غیر فعال شدن جعبه چک مقدار این کنترل تغییر خواهد کرد. از جعبه چک معمولا برای کنترل متغییر هایی که فقط دو مقدار می گیرند استفاده می شود.

کنترل دکمه رادیویی

دکمه رادیویی هم کنترلیست که می تواند دو وضعیت فعال و غیر فعال داشته باشد و از این نظر شبیه جعبه چک است. تفاوت این دو درآن است که از یک گروه دکمه رادیوئی در هر لحظه فقط یکی می تواند فعال باشد. دکمه های رادیویی معمولا گروهی مورد استفاده قرار می گیرند که هر گروه رادیویی دارای عملکرد مستقلی است.

کنترل جعبه لیست باز شو

کنترل جعبه لیست باز شو یا کنترل جعبه ترکیبی یک جعبه ادیت است که به یک لیست متصل شده و کاربر می تواند یک گزینه را از میان چندین گزینه آن انتخاب کند. اگر گزینه مورد نظر کاربر در میان گزینه های لیست نباشد، وی می تواند آنرا در جعبه ادیت وارد کند.

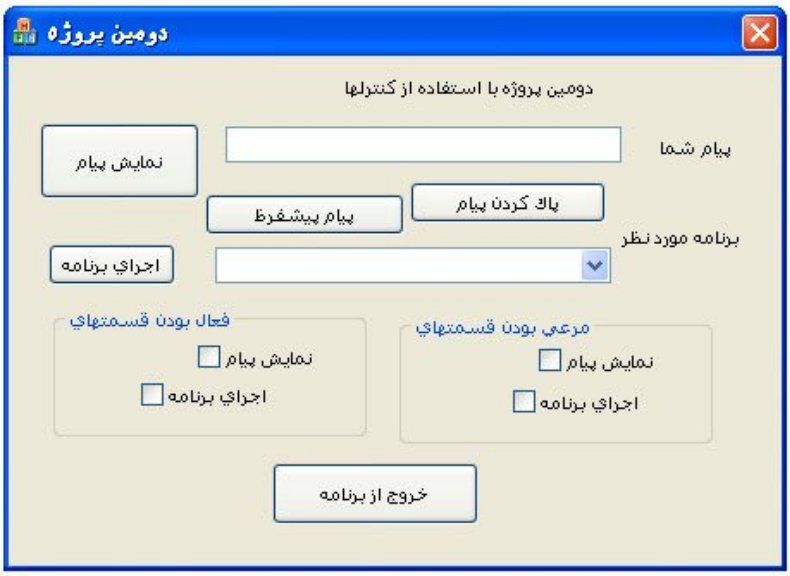
استفاده از کنترل ها در برنامه

برنامه ای که امروز قصد داریم آنرا بنویسیم برنامه ایست که از تعدادی کنترل های ویندوز در یک دیالوگ استفاده می کند.این کنترل ها عملکردهای متفاوتی دارند. در قسمت بالای پنجره جعبه ادیتی وجود دارد که کاربر می تواند پیام مورد نظرش را درآن بنویسد، این پیام هنگام کلیک شدن دکمه کنار جعبه ادیت نمایش داده خواهد شد. در زیر این جعبه ادیت دو دکمه دیگر هستند که یکی جعبه ادیت را پاک می کند و دیگری یک پیام از پیش تعیین شده را در آن می نویسد. در زیر این دکمه ها یک جعبه لیست باز شو قرار دارد که در آن تعدادی از برنامه های ویندوز را قرار داده ایم، کابر می تواند با انتخاب یکی از این برنامه ها و کلیک کردن دکمه کنار لیست آن برنامه را اجرا کند.در زیر این جعبه لیست دو گروه جعبه چک قرار دارند که هرکدام عملکرد کنترلهایی دیکر موجود بر روی پنجره برنامه را تاحت تاثیر قرار می دهند.گروه سمت چپ ، کنترلهای دیگر را فعال یا غیر فعال می کنند و گروه سمت راست ، این کنترلها را مرئی یا نا مرئی می کنند. در منتهی الیه پایین دیالوگ هم یک دکمه قرار دارد که با آن می توانید برنامه را ببندید.

طراحی و ایجاد برنامه

با استفاده از آنچه که در فصل قبل آموخته اید، برنامه امروز را با دنبال کردن مراحل زیر ایجاد کنید:

۱. یک پروژه جدید MFC Application با نام P2 ایجاد کرده و تنظیمات آن را بر روی Dialog based و Use MFC in a static library قرار دهید.
۲. تنظیمات زبان پنجره دیالوگ برنامه را به فارسی تغییر دهید.
۳. تمامی کنترلهای موجود روی پنجره برنامه را حذف کرده و کنترلهای جدیدی به شکل زیر بر روی آن قرار دهید.



طراحی پنجره برنامه

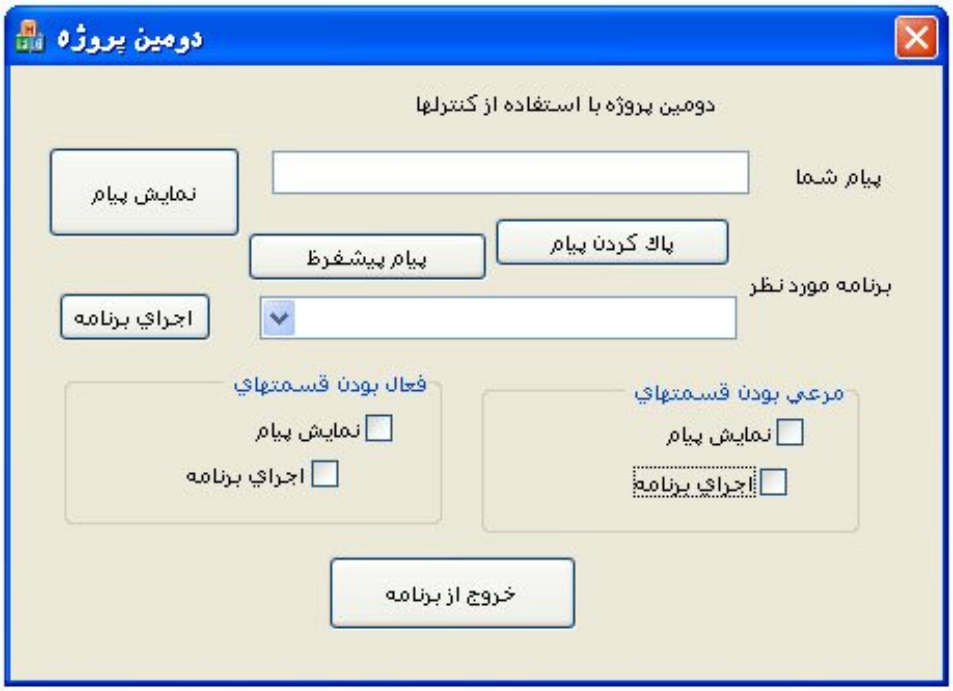
مثلا برای قرار دادن یک دکمه فرمان به برنامه ابتدا بر روی سربرگ Toolbox واقع در پنجره Properties سمت راست محیط ویژوال استدیو بروید تا پنجره کشویی آن باز شود ، حالا بر روی عبارت Button که برای ایجاد دکمه فرمان است کلیک کنید.

حالا یک دکمه فرمان بر روی پنجره برنامه رسم کنید و با استفاده از فیلد Caption واقع در پنجره Properties نام آن را عوض نمایید.
برای قرار دادن دیگر کنترلها بر روی پنجره برنامه نیز بوسیله همین روش ابتدا از پنجره کشویی Toolbox کنترلهای مورد نظر را انتخاب کنید، سپس با کلیک کردن روی پنجره دیالوگ برنامه آنرا ایجاد نمایید و با استفاده از فیلد Caption واقع در پنجره Properties نام هر یک از کنترلها را شبیه عکس بالا تغییر دهید. اما پیش فرض این کنترلها برای زبان انگلیسی که از چپ به راست است تعریف شده و چون ما از زبان فارسی استفاده کرده ایم که از راست به چپ است باید تغییراتی در نحوه نمایش این کنترلها برای زیباتر شدن آن اعمال کنیم و تنظیمات بعضی از کنترلهای برنامه را از راست به چپ تغییر دهیم.مراحل زیر را انجام دهید

در حالی که کلید کنترل از روی صفحه کلید پایین نگهداشته اید بوسیله کلید چپ ماوس

۱. کنترل متن (Edit Control) را انتخاب نمایید،
۲. کنترل جعبه لیست باز شو(Combo Box Control) را انتخاب نمایید،
۳. دو عدد کنترل گروه (Group Box Control) را انتخاب نمایید،
۴. چهار عدد کنترل جعبه چک (Check-box Control) را نتخاب نمایید.

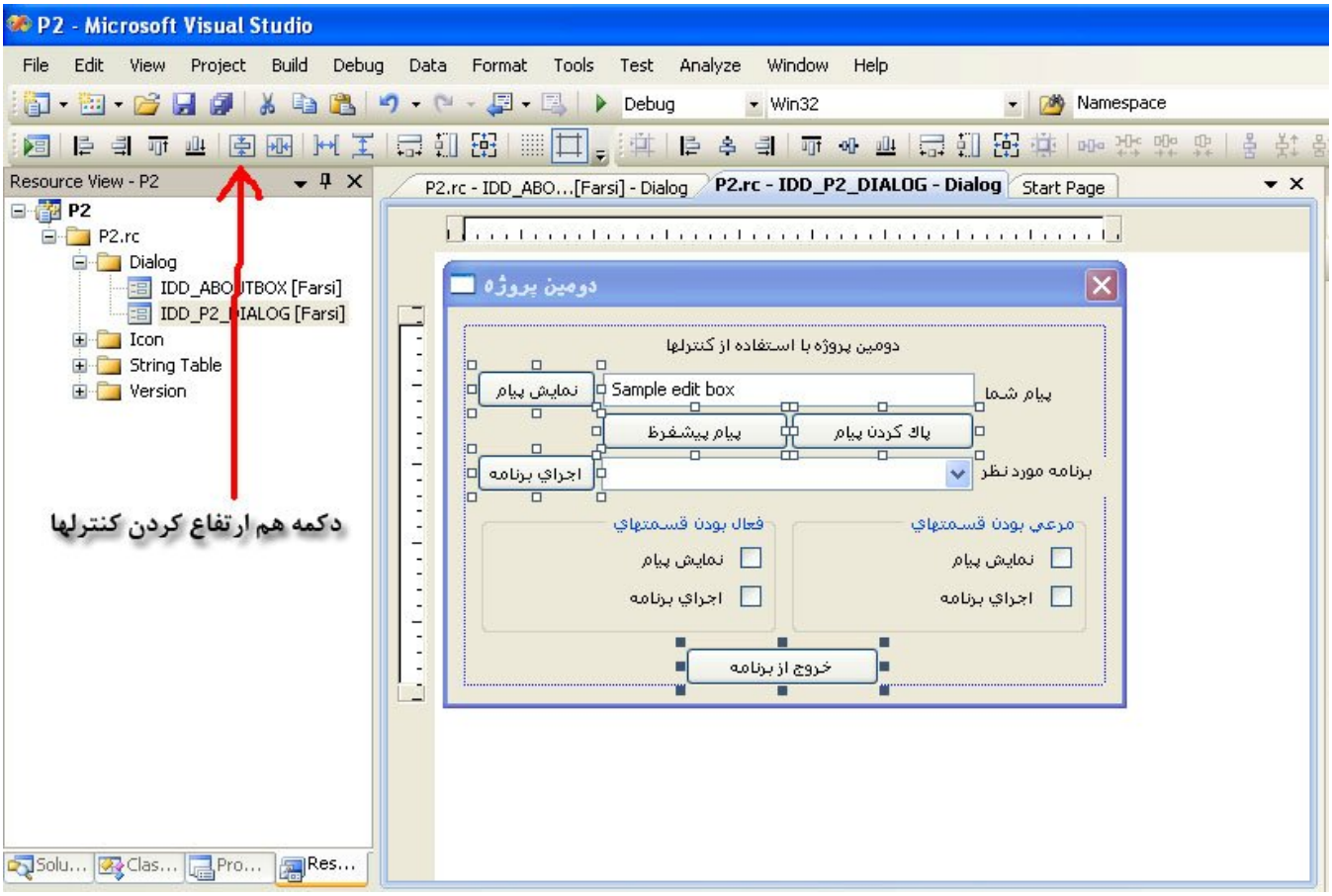
سپس از پنجره Properties دو فیلد Right Align Text وRight To Left Reading Order را به True تغییر دهید. چون این دو فیلد در تمامی این کنترلها وجود دارد لذا با اینکار به یکباره تمامی آنها را با هم تنظیم میکنیم و صرفه جویی در وقت می نماییم.در این مرحله پنجره برنامه به شکل زیر خواهد شد.



پس از قرار دادن ، تغییر نام و تنظیم کردن خواص کنترلها نوبت به چیدن آنها می رسد

در این مرحله مرتب کردن اینهمه کنترل بوسیله استفاده از چشم و تغییر اندازه دستی کار مشگلی است که احتمال خطا نیز وجود دارد. ویژوال استدیو برای انجام این کار ابزارهای مفیدی دارد که بوسیله آنها می شود چند کنترل را یک اندازه کرد و یا بر روی یک خط در پنجره مرتب نمود و از این قبیل کارای هایی که برای زیبا شدن پنجره برنامه مفید هستند. این ابزارها به صورت آیکون های کوچکی در بالای محیط ویژوال استدیو قرار دارند. برای مثال شما می خواهید ارتفاع تمامی دکمه های فرمان موجود در پنجره برنامه تان را یک اندازه کنید ، برای این کار کلید کنترل (Ctrl) از روی کیبورد را نگه داشته و سپس بر روی تک تک دکمه های فرمان کلیک می کنیم تا تمامی آنها انتخاب شوند ، حالا بر روی دکمه هم ارتفاع کردن کلیک می کنیم تا ارتفاع تمامی دکمه فرمانها به یک اندازه درآیند. دکمه های مفید دیگری نیز در این قسمت جهت مرتب کردن از چهار جهت ، وسط چین کردن و ... وجود دارند که شما خودتان با آزمایش و خطا از عملکرد آنها مطلع شوید.

به عکس زیر دقت کنید.



پنجره برنامه پس از مرتب شدن به شکل زیر در خواهد آمد

ست کردن نام شناسایی (ID) کنترلهای برنامه

در این مرحله می خواهیم نام کنترلهای برنامه را عوض کنیم،این نامی است که در کد های برنامه برای دسترسی به آن کنترل از آن استفاده می کنیم. هر کنترل ایجاد شده در ابتدا یک نام به صورت پیشفرض دارد. مثلا نام(ID) اولین دکمه فرمان ایجاد شده توسط شما BUTTON1 دومین دکمه فرمان BUTTON2 و ... می باشد و در کل به همین شکل برای بقیه کنترلها نیز صدق میکند.اما برای ساده تر کردن و آشنایی با روش تغییر نام کنترلها قصد داریم آنها را تغییر دهیم.

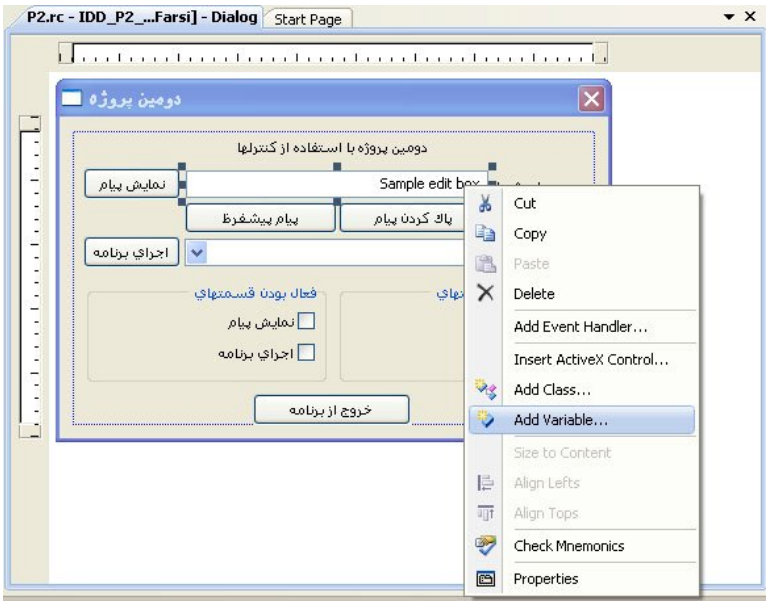
تمامی کنترلها را تک به تک انتخاب کرده و از پنجره Properties مقدار فیلد ID آنها را با جدول زیر برابر نمایید.

نوع کنترل و عنوان آن	نام کنترل ID	نوع کنترل و عنوان آن	نام کنترل ID
کنترل متن ثابت بالای پنجره	IDC_STATIC	کنترل دکمه فرمان پاک کردن پیام	IDC_CLRMSG
کنترل متن ثابت پیام شما	IDC_STATICMSG	کنترل دکمه فرمان اجرای برنامه	IDC_RUNPGM
کنترل متن ثابت برنامه مورد نظر	IDC_STATICPGM	کنترل دکمه فرمان خروج از برنامه	IDC_EXIT
کنترل ویرایش متن جهت نمایش پیام	IDC_MSG	کنترل جعبه چک مرئی شدن نمایش پیام	IDC_CKSHWMSG
کنترل جعبه لیست باز شو جهت اجرای برنامه	IDC_PROGTORUN	کنترل جعبه چک مرئی شدن اجرای برنامه	IDC_CKSHWPGM
کنترل دکمه فرمان نمایش پیام	IDC_SHWMSG	کنترل جعبه چک فعال شدن نمایش پیام	IDC_CKENBLMSG
کنترل دکمه فرمان پیام پیشفرظ	IDC_DFLTMSG	کنترل جعبه چک فعال شدن اجرای برنامه	IDC_CKENBLPGM

نسبت دادن متغیر به کنترل ها

اگر قبلا با Visual Basic برنامه نویسی کرده باشید شاید تصور کنید که در این مرحله آماده اید تا کد بنویسید.خوب ، در مورد Visual C++ چنین چیزی نیست. قبل از شروع کد نویسی باید به تمام کنترلها به جز متن ثابت و دکمه فرمان متغییر نسبت دهیم. کد نویسی در VC++ یعنی کار با این متغییرها. مقداری که کاربر در هر کنترل وارد می کند به این متغییر ها داده می شود. تا بعدا در برنامه مورد استفاده قرار گیرد. برنامه هم برای تغییر دادن محتوی کنترلها باید مقادیر مورد نظر را در این متغییرها قرار دهد.

اما این متغیرها را چگونه باید تعریف کرد؟ بر روی کنترل جعبه ویرایش متن که قصد تعریف متغییر برای آن دارید راست کلیک کنید و از منوی باز شده گزینه ADD Variable... را انتخاب نمایید.



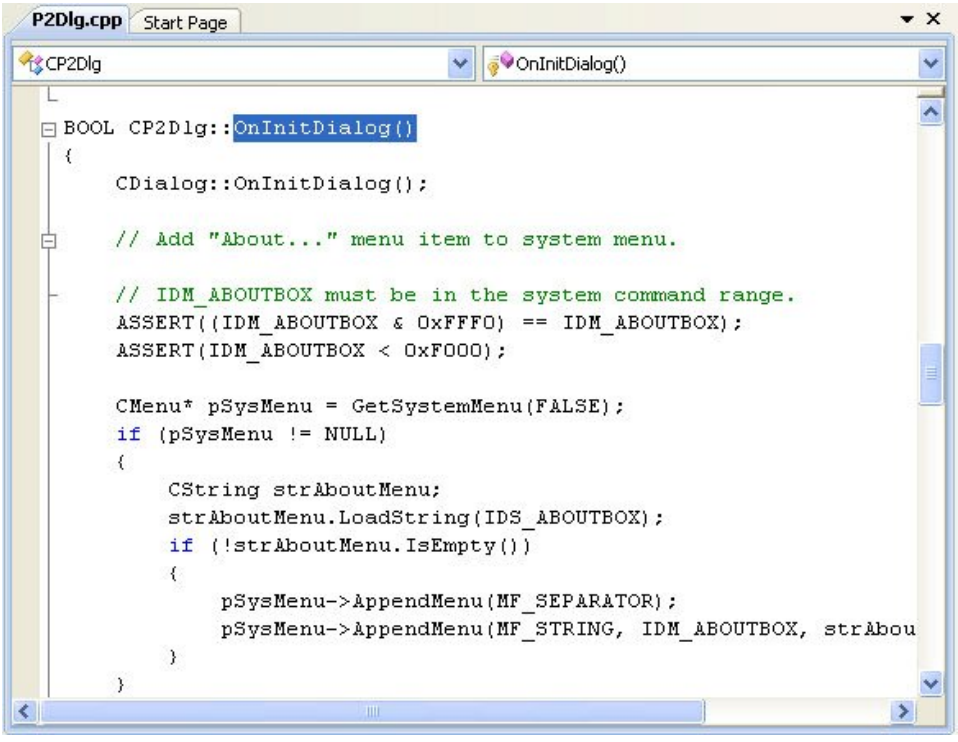
سپس در پنجره به نمایش در آمده عنوان Category را به Value تغییر دهید و در فیلد Variable name نام متغییر را m_strMessage تعیین نمایید و بر روی Finish کلیک کنید تا متغییر برای این کنترل تعریف شود.
کنترلهایی که به تعریف متغییر نیاز دارند را با همین روش طبق جدول زیر تعریف و نامگذاری نمایید.

نام کنترل (ID)	نام متغییر(Variable name)	مقوله (Category)	نوع متغییر (Variable Type)
IDC_MSG	m_strMessage	Value	CString
IDC_PROGTORUN	m_strProgToRun	Value	CString
IDC_PROGTORUN	m_strCB1	Control	CComboBox
IDC_CKENBLMSG	m_bEnableMsg	Value	BOOL
IDC_CKENBLPGM	m_bEnablePgm	Value	BOOL
IDC_CKSHWMSG	m_bShowMsg	Value	BOOL
IDC_CKSHWPGM	m_bShowPgm	Value	BOOL

توجه: در دنیای MFC نام متغیرهای عضو کلاس با m_ شروع می شود. بعد از آن با چند حرف نوع متغییر مشخص مشود،(مثلا b یعنی متغیر منطقی و str یعنی متغیر رشته ای و غیره ...) و به دنبال آن نام متغییر آورده می شود. این استاندارد در تمام کتابهای برنامه نویسی VC++ و MFC رعایت می شود.

عملیاتی کردن کنترلها

قبل از شروع کد نویسی برای کنترلها ابتدا باید به تعدادی از متغییر های عضو مقدار داده و آنها را آماده سازی کنید. برای انجام این کار ابتدا از پنجره سمت چپ محیط ویژوال استدیو سربرگ Solution Explorer را انتخاب نمایید، سپس بر روی فایل P2Dlg.cpp دابل-کلیک کنید تا کدهای درون فایل را مشاهده نمایید.حالا به دنبال تابع `OnInitDialog()` بگردید(عکس زیر).این تابع آماده سازی نام دارد و جزء اولین توابعی است که با اجرایی برنامه شما اجرا خواهد شد و معمولا از آن برای مقدار دهی اولیه در برنامه به متغییر ها و آماده سازی اولیه برنامه استفاده میکنند.



به آخر تابع رفته و کدهای زیر را بعد از توضیحات شروع شده با عبارت TODO به تابع اضافه نمایید.

```
m_strCB1.InsertString( 0, _T("ماشین حساب") );
m_strCB1.InsertString( 1, _T("دفترچه یادداشت") );
m_strCB1.InsertString( 2, _T("پاستور") );

m_strMessage=L"یک پیام بگذارید";

m_bShowMsg=TRUE;
m_bShowPgm=TRUE;
m_bEnableMsg=TRUE;
m_bEnablePgm=TRUE;

UpdateData(FALSE);
```

اگر قبلا با C یا C++ برنامه نوشته باشید، توجه کرده اید که روش مقدار دادن به متغیر m_strMessage بیشتر شبیه Visual Basic است تا C. کلاس CString کلاسیست که اجازه می دهد تا با رشته ها بسیار راحتتر و مانند زبان Visual Basic کار کنید.

- در سه خط اول به جعبه لیست باز شو سه عنوان اضافه می نماییم، چون از حروف فارسی که به صورت یونی کد است استفاده کرده ایم رشته ها را مستقیما در گیومه قرار ندادیم و به صورت _T("Text") نوشته ایم.
- در خط چهارم درون جعبه ویرایش متن یک پیام اولیه قرار داده ایم که به علت فارسی بودن از عبارت L"Text" به جای دو گیومه استفاده کرده ایم.
- خط پنجم تا هشتم هم کنترلهای جعبه چک را علامت دار می کند.
- مهمترین قسمت این کد خط آخر آن است، تابع UpdateData در واقع کلید کار با متغیر کنترل هاست. این تابع از یک طرف مقدار کنترل را گرفته و به متغیرهای وابسته می دهد و از طرف دیگر با استفاده از مقدار متغیرها کنترل ها را به روز در می آورد. جهت کار را آرگومان این تابع مشخص می کند. با آرگومان FALSE حالت کنترل ها با مقدار متغیرها به روز می شود و آرگومان TRUE باعث می شود تا حالت کنترل ها در متغیرهای وابسته نوشته شود.بعد از هر تغییر که در کنترل ها یا متغیرهای وابسته به آنها می دهیم باید این تابع را با آرگومان مناسب اجرا کنید تا تغییرات خواسته شده اعمال شوند.

بستن برنامه

اولین گام در هر برنامه ای آنست که مطمئن شویم می توان برنامه را بست. برای فعال کردن دکمه خروج از برنامه ابتدا بر روی آن راست کلیک کرده و از منوی به نمایش درآمده گزینه Add Event Handler... را انتخاب نمایید. در پنجره بعدی مقدار Message Type باید بر روی عبارت BN_CLICKED (که به صورت پیشفرض نیز روی همین گزینه قرار دارد) باشد تا تابعی که در حال ساخت آن هستیم با کلیک بر روی این دکمه فعال شود. با کلیک بر روی دکمه Add and Edit تابعی با نام OnBnClickedExit() تولید شده و به پنجره ای برای ویرایش کد آن راهنمایی می شوید. کنترل برنامه در صورت فشرده شدن دکمه خروج از برنامه به این تابع منتقل می شود، در نتیجه هر دستوری که در آن بنویسیم به محظ کلیک کردن آن اجرا میشود.دستور زیر که باعث بستن برنامه می شود را در این تابع بنویسید.

```
OnOK();
```

نمایش پیام کاربر

برای نمایش پیامی که کاربر در جعبه ادیت نوشته است باید تابعی برای دکمه نمایش پیام بسازیم که با کلیک بر روی آن تابع اجرا شده و پیام کاربر را نمایش دهد. طبق روشی که در بالا توضیح داده شده یک تابع برای کلیک کردن بر روی دکمه نمایش پیام ساخته و کد زیر را درآن بنویسید.

```
UpdateData(TRUE);
MessageBox(m_strMessage);
```

خط اول باعث می شود که مقدار درون کنترل جعبه ادیت به متغییر وابسته به آن یعنی m_strMessage منتقل شود، سپس دستور بعدی آن را در یک پنجره ای جداگانه نمایش می دهد.

پاک کردن پیام کاربر

تابعی برای کلیک کردن بر روی دکمه پاک کردن پیام ایجاد نمایید و کد زیر را درآن بنویسید.

```
m_strMessage="";
UpdateData(FALSE);
```

پیام پیشفرض

پس از ایجاد یک تابع برای دکمه پیام پیشفرض کد زیر را درآن بنویسید.

```
m_strMessage=L"سلام";
UpdateData(FALSE);
```

اجرای برنامه های دیگر

اگر به یاد داشته باشید نام سه تا از برنامه های ویندوز را در لیست جعبه ترکیبی نوشته ایم و وقتی برنامه در حال اجرا باشد می توان با باز کردن این لیست یکی از آیتم ها را انتخاب کرد. تابع دکمه فرمان اجرای برنامه باید این نام را گرفته و برنامه انتخاب شده را اجرا کند. کد زیر را در تابع این دکمه بنویسید.

```
UpdateData(TRUE);

if(m_strProgToRun == (L"ماشین حساب")) WinExec("calc.exe",SW_SHOW);
if(m_strProgToRun == (L"دفترچه یادداشت")) WinExec("notepad.exe",SW_SHOW);
if(m_strProgToRun == (L"پاستور")) WinExec("sol.exe",SW_SHOW);
```

شرطهای موجود در صورت برابری متغیر m_strProgToRun با هر کدام از نوشته های درون لیست ، برنامه متناضر با آن را بوسیله دستور WinExec اجرا می نمایند.

غیر فعال یا فعال نمودن کنترلها

برای انجام این کار دو تابع برای کلیک کردن بر روی جعبه چکهای درون کنترل گروه با عنوان فعال بودن قسمتهای می سازیم و کدهای زیر را در دو تابع ساخته شده قرار می دهیم.

کد درون جعبه چک نمایش پیام:

```
UpdateData(TRUE);
if(m_bShowMsg == TRUE)
{
	GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
	GetDlgItem(IDC_SHWMSG)->ShowWindow(TRUE);
	GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
	GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
	GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
}
else
{
	GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
	GetDlgItem(IDC_SHWMSG)->ShowWindow(FALSE);
	GetDlgItem(IDC_DFLTMSG)->ShowWindow(FALSE);
	GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
	GetDlgItem(IDC_STATICMSG)->ShowWindow(FALSE);
}
```

کد درون جعبه چک اجرای برنامه:

```
UpdateData(TRUE);
if(m_bShowPgm == TRUE)
{
	GetDlgItem(IDC_RUNPGM)->ShowWindow(TRUE);
	GetDlgItem(IDC_PROGTORUN)->ShowWindow(TRUE);
	GetDlgItem(IDC_STATICPGM)->ShowWindow(TRUE);
}
else
{
	GetDlgItem(IDC_RUNPGM)->ShowWindow(FALSE);
	GetDlgItem(IDC_PROGTORUN)->ShowWindow(FALSE);
	GetDlgItem(IDC_STATICPGM)->ShowWindow(FALSE);
}
```

مرئی یا نا مرئی کردن کنترلها

شبیه بالا عمل کرده و دو تابع برای کلیک کردن بر روی جعبه چکهای درون کنترل گروه با عنوان مرئی بودن قسمتهای می سازیم و کدهای زیر را در آن دو تابع قرار می دهیم.

کد درون جعبه چک نمایش پیام:

```
UpdateData(TRUE);
if(m_bEnableMsg == TRUE)
{
     GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
     GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
     GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
     GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
     GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
}
else
{
     GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
     GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
     GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
     GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
     GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
}
```

کد درون جعبه چک اجرای برنامه:

```
UpdateData(TRUE);
if(m_bEnablePgm == TRUE)
{
     GetDlgItem(IDC_RUNPGM)->EnableWindow(TRUE);
     GetDlgItem(IDC_PROGTORUN)->EnableWindow(TRUE);
     GetDlgItem(IDC_STATICPGM)->EnableWindow(TRUE);
}
else
{
     GetDlgItem(IDC_RUNPGM)->EnableWindow(FALSE);
     GetDlgItem(IDC_PROGTORUN)->EnableWindow(FALSE);
     GetDlgItem(IDC_STATICPGM)->EnableWindow(FALSE);
}
```

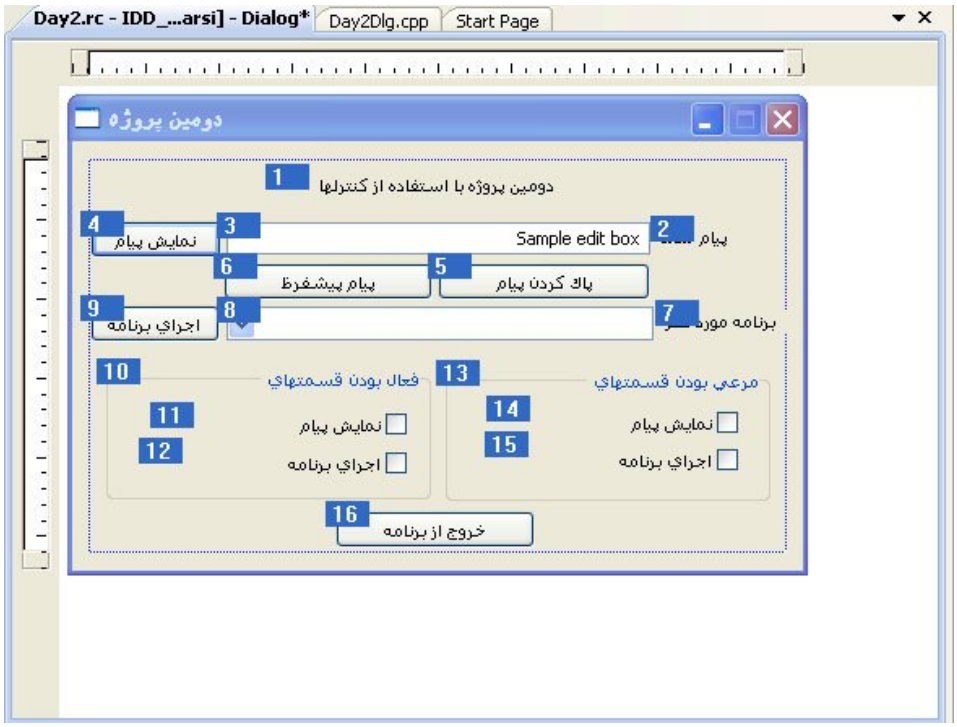
توضیح نهایی برنامه

در قسمت ابتدایی این توابع مقدار کنترل های روی پنجره در متغیرهای برنامه نوشته می شود. سپس مقدار متغییر های متناظر با جعبه چک ها مورد بررسی قرار می گیرد. اگر مقدار این متغییر ها TRUE باشد، کنترل ها باید فعال و یا مرئی شوند و اکر مقدار آنها FALSE باشد، کنترلها باید غیر فعال و یا نامرئی شوند. اگر برنامه را به درستی و بدون خطا نوشته باشید برنامه شما پس از کامپایل و اجرا به درستی کار خواهد کرد.

تعیین ترتیب حرکت بین کنترلها

بعد از قرار دادن کنترلها روی پنجره برنامه، باید مطمعن شویم که کاربر با زدن کلید Tab آنطور که شما می خواهید بین کنترلها حرکت خواهد کرد. برای تعیین این ترتیب حرکت که tab order نامیده میشود باید پس از انتخاب پنجره دیالوگ برنامه از منوی Format گزینه Tab

Order را انتخاب نمایید تا اعدادی در اطراف کنترلها نمایش داده شوند، که در واقع این اعداد ترتیب حرکت بین کنترلها را نشان می دهند. حالا شما با کلیک کردن بر روی کنترلها دوباره ترتیب حرکت بین انها را به صورت دلخاه مشخص کنید.پس از انجام کار دوباره از منوی Format گزینه Tab Order را انتخاب کنید تا از این حال خارج شوید.



آموزش مقدماتی MFC

بهزاد جناب

فصل چهارم

استفاده از ماوس و کی بورد

اغلب پیش می آید که یک برنامه باید کارهایی را با ماوس انجام دهد، مثلا بسته به محل کلیک شدن و نحوه حرکت آن اقداماتی را انجام دهد. یا گاهی لازم است برنامه بداند که کاربر کدام دکمه را کلیک کرده یا در حین نگه داشتن دکمه ماوس چه کارهایی انجام داده است.
اتفاقاتی که روی کی برد هم می افتد می تواند مورد توجه خاص برنامه باشد. مثلا ، کابر چه کلیدی را زده ، چقدر آنرا نگه داشته، یا چه زمانی آنرا رها کرده است. در این فصل خواهیم آموخت

- رویدادهای ماوس و به کارگیری آنها در برنامه
- چگونگی کشف رویدادهای ماوس
- رویدادهای کی بورد و نحوه تحریک آنها
- روش استفاده از رویدادهای کی بورد

آشنایی با رویدادهای ماوس

در فصل قبل دیدید که هر کنترل تعداد مشخص و محدودی رویداد دارد.تعداد رویدادهای ماوس هم بسیار محدود است و به کلیک و دو-کلیک محدود می شود. اما یک نگاه به ماوس نشان می دهد که ماوس بیش از اینها امکانات دارد.مثلا ، چگونه می توان با دکمه راست ماوس کار کرد و فهمید که آن چه زمانی فشرده شده است؟ یا مثلا چیزهایی را روی صفحه کامپیوتر جابجا کرد؟
مهمترین رویدادهای ماوس در جدول زیر نمایش داده شده است. به کمک این رویدادها می توانید هر کاری که با ماوس لازم باشد در برنامه تان انجام دهید.

پیام Messages	مفهوم
WM_LBUTTONDOWN	دکمه چپ ماوس فشرده شده است
WM_LBUTTONUP	دکمه چپ ماوس رها شده است
WM_LBUTTONDBLCLK	دکمه چپ ماوس دو-کلیک شده است
WM_RBUTTONDOWN	دکمه راست ماوس فشرده شده است
WM_RBUTTONUP	دکمه راست ماوس رها شده است
WM_RBUTTONDBLCLK	دکمه راست ماوس دو-کلیک شده است
WM_MOUSEMOVE	ماوس در فضای پنجره برنامه حرکت کرده است
WM_MOUSEWHEEL	چرخک ماوس چرخانده شده است

نقاشی با ماوس

امروز برای نمایش قابلیت های ماوس و طرز استفاده از آنها یک برنامه ساده نقاشی با ماوس خواهیم نوشت. این برنامه عمدتا از رویداد WM_MOUSEMOVE ، که حرکت ماوس را آشکار می کند، استفاده خواهد کرد.در این برنامه خواهید دید که چگونه می توان فهمید ماوس در کجای پنجره برنامه قرار دارد. نسبتا ساده بنظر می رسد، پس بیایید شروع کنیم.

۱. یک پروژه MFC Application به صورت Dialog Based ساخته و نام آنرا Mouse بگذارید.
۲. بعد از ایجاد پروژه، تمام کنترلهای روی آنرا حذف کنید تا صفحه دیالوگ برنامه ما بوم نقاشی شود. برای بدام انداختن رویدادهای ماوس و کی برد باید هیچ کنترلی روی پنجره برنامه نباشد ، در غیر اینصورت رویدادهای ماوس و کیبورد به کنترل دارای فوکوس(کنترلی که فعال است) خواهد رفت.
۳. نام کلاس دیالوگ مورد نظر که در اینجا CMouseDlg است را از Class View انتخاب کرده (با موس highlight کنید)

و در پنجره Properties آیکن Messages را برای مشاهده پیام ها انتخاب نمایید

سپس پیام WM_MOUSEMOVE را از لیست انتخاب و در پایان برای ساختن تابع مرد نظر برای این رویداد از Combo box کنار آن گزینه add را انتخاب نمایید تا به تابع مورد نظر در پنجره ادیتور جهت نوشتن کد آن هدایت شوید.

۴. در پنجره ادیتور تابع مرد نظر با عنوان OnMouseMove ایجاد می شود که باید کد زیر را در این تابع بنویسید.

```
if((nFlags & MK_LBUTTON) == MK_LBUTTON)
{
     CClientDC dc(this);
     dc.SetPixel(point.x , point.y ,RGB(0,0,0)));
}
```

اگر به خط اول تابع نگاه کنید خواهید دید که این تابع دو آرگومان ورودی دارد ، ورودی اول مجموعه ایست از چند پرچم که نشان می دهد کدام دکمه ماوس در حین حرکت آن فشرده شده است. بررسی این حالت در اولین خط کدی که نوشته ایم صورت می گیرد. یعنی کد زیر:

```
if((nFlags & MK_LBUTTON) == MK_LBUTTON)
```

نیمه دوم این دستور بررسی می کند که آیا دکمه چپ ماوس فشرده شده است یا خیر. نیمه اول دستور if پرچمیست که بررسی این موضوع را به عهده دارد. اگر این دو نیمه منطبق باشند، دکمه چپ ماوس فشرده شده تلقی خواهد شد.
ورودی دوم تابع OnMouseMove ، مکان ماوس را بر می گرداند. این آرگومان مختصات مکان فعلی ماوس را در خود دارد.از این اطلاعات برای رسم نقطه روی پنجره دیالوگ استفاده می شود.
قبل از آنکه بتوانیم چیزی روی دیالوگ رسم کنیم، باید محتوای ابزار(device context) آنرا بدست آوریم.این کار با تعریف یک وهله از کلاس CClientDC انجام می شود. این کلاس علاوه بر محتوای ابزار ، تمام توابع لازم برای ترسیم را هم در خود دارد. محتوای ابزار در واقع مانند یک بوم است که برنامه می تواند روی آن نقاشی کند. تا زمانی که محتوای ابزار یک پنجره را نداشته باشید نمی توانید هیچ چیزی روی آن رسم کنید. تابعی که برای نقاشی از آن استفاده کرده ایم تابع SetPixel است. این تابع سه ورودی می گیرد که عبارتند از مختصات X و Y نقطه ای که باید رنگ آن تغییر یابد و رنگ مورد نظر برای آن نقطه. اگر برنامه را کامپایل و اجرا کنید خواهید دید که می توان با گرفتن دکمه چپ ماوس و حرکت دادن آن روی پنجره دیالوگ نقاشی کرد.

نکته: در ویندوز هر رنگ یک عدد است، این عدد مقدار رنگهای قرمز، سبز و آبی را تعیین میکند. تابع RGB که مخفف اول نامهای این سه رنگ است سه عدد جداگانه را گرفته و آنها را با هم ترکیب می کند تا عدد رنگ مناسب با مقادیر داده شده ساخته شود. هر کدام از این سه رنگ می تواند مقداری بین ۰ تا ۲۵۵ بگیرد.

استفاده از AND و OR باینری

اگر تازه وارد دنیای C++ شده اید، باید با تفاوت انواع AND و OR آشنا شوید. دو نوع AND و OR وجود دارد، منطقی و باینری. AND و OR منطقی در دستورات شرطی مانند دستورات if به کار می روند چنانچه در فصل اول آن را توضیح دادیم. درحالیکه AND و OR باینری برای ترکیب اعداد باینری مورد استفاده قرار می گیرند.
برای AND باینری از علامت & استفاده می کنیم. یک علامت & برای AND باینری و دو علامت && برای AND منطقی بکار می رود. رفتار AND منطقی شبیه عملکرد AND در زبان Visual Basic است. یک عبارت AND منطقی زمانی درست است که هر دو قسمت آن ارزش درست داشته باشند.AND باینری فقط باعث تغییر حالت بیت ها خواهد شد و ارزش درست و نادرست در آن بی معنی است. وقطی دو بیت با یکدیگر AND می شوند نتیجه فقط زمانی ۱ خواهد شد که هر دو بیت ۱ باشند و در غیر اینصورت نتیجه صفر خواهد شد. برای درک بهتر مطلب به یک مثال توجه کنید. دو عدد باینری ذیل را که با هم AND باینری کرده ایم مشاهده کنید.

عدد اول	01011001
عدد دوم	00101001
AND باینری	00001001

توجه کنید که در عدد حاصله فقط بیت هایی 1 هستند که هر دو بیت متناظر با آنها در اعداد اولیه 1 باشند و تمام بیت های دیگر حتی آهایی که یکی از آنها 1 است 0 شده اند.

OR باینری نیز با علامت | نمایش داده می شود و مانند AND، یک علامت | نشان دهنده OR باینری و دو علامت || نشاندهنده OR منطقی است. OR منطقی هم در عبارات شرطی کاربرد دارد و عملکرد آن بسیار شبیه OR در Visual Basic است. از OR باینری برای ترکیب بیت ها استفاده می شود. حاصل OR شدن دو بیت فقط زمانی 0 است که هر دو قسمت آن مقدار 0 داشته باشند و در غیر این صورت مقدار 1 خواهد داشت. باز هم به یک مثال توجه کنید.ملاحضه می کنید که فقط بیت هایی 0 شده اند که هر دو جزء آنها 0 بوده است.

عدد اول	01011001
عدد دوم	00101001
OR باینری	01111001

پرچمهای باینری

AND و OR باینری در VC++ عمداتا برای خواندن و یا ست کردن پرچمها به کار می روند. پرچم(Flag) مقداریست که هر بیت آن وضعیت یا حالتی را نشان می دهند و برنامه نویس می تواند از این پرچمه برای تشخیص اوضاع استفاده کند یا آنرا تحت کنترل خود در آورد. از آنجایی که هر بیت می تواند یک حالت و موقییت را نشان دهد، یک پرچم می تواند چندین حالت را در خود حمل کند. برای ترکیب حاتها در یک پرچم معمولا از OR باینری استفاده می شود. مثلا اگر دو پرچم داشته باشیم که هر کدام حالاتی را نشان می دهد، می توانیم آنها را با OR کردن با هم ترکیب کنیم.

پرچم اول	00001000
پرچم دوم	00100000
ترکیب پرچم ها	00101000

بدین ترتیب می توان دو پرچم را در یکدیگر ادغام و در جا صرفه جویی کرد. در حقیقت اکثر کنترلهای ویندوز چنین می کنند و هر پرچم خواص متعددی را در خود نگه میدارد. خواصی که حالت روشن یا خاموش (بله یا خیر) دارند بسیار مستعد این وضعیت هستند. از طرفی دیگر، اکر بخواهید مقدار یک پرچم خواص را در ترکیب از چند پرچم بخوانید، باید از AND استفاده کنید.

ترکیب چند پرچم	00101000
پرچم مورد نظر	00001000
حاصل	00001000

با این روش می توان یک پرچم خواص را از میان ترکیب چندین پرچم بیرون کشید(و به اصطلاح فیلتر کرد). اگر عدد حاصل معادل آنچه ما انتظار داریم باشد پرچم مورد نظر در آن ترکیب وجود دارد و اگر حاصل صفر شود، پرچم مورد نظر در آن ترکیب وجود ندارد، در نتیجه می توان این عدد را در یک دستور if تست کرد. بدین ترتیب دستور if کد زیر را می توان ساده تر کرد.

```
if (nFlags & MK_LBUTTON)
```

اگر می خواهید عدم وجود یک پرچم را تست کنید می توانید از دستوری مانند زیر استفاده کنید.

```
if ( !(nFlags & MK_LBUTTON) )
```

استفاده از هر یک از این روشها به شرایط بستگی دارد و این برنامه نویس است که باید نوع مناسب را انتخاب کند.

اصلاح برنامه نقاشی

اگر برنامه را اجرا کرده باشید احتمالا متوجه یک اشکال کوچک در آن شده اید، برای رسم یک خط ممتد باید ماوس را بسیار آهسته حرکت دهید اما چرا این اشکال در دیگر برنامه های نقاشی وجود ندارد؟ چون آنها بین دو نقطه خط می کشند و نقطه ها (پیکسل ها) را ست نمی کنند، در واقع اکثر این قبیل برنامه ها چنین عمل می کنند. این قبیل برنامه ها در حین حرکت ماوس مکان آنرا چک می کنند و چون نمی توانند تمام مسیر آنرا بررسی کنند مجبورند بین خود فرضیاتی بکنند و سپس بین نقاط بدست آمده خط رسم کنند.
برای اینکه برنامه ما هم مانند این قبیل برنامه ها عمل کند، چه باید کرد؟ ابتدا باید مکان قبلی ماوس را بطریقی ذخیره کنیم. برای این منظور به دو متغییر جدید برای ذخیره کردن مختصات X و Y مکان قبلی ماوس نیاز داریم. برای تعریف این متغیر ها باید بر روی پنجره دیالوگ برنامه راست کلیک کنید و از منوی باز شده گزینه **Add Variable** را انتخاب نمایید، سپس در پنجره باز شده دو متغییر از نوع **int** و با دسترسی **private** با نامهای **m_iPrevX** و **m_iPrevY** ایجاد نمایید. پس از افزودن این دو متغییر تابع **OnMouseMove** را به شکل زیر تغییر دهید.

```
if((nFlags & MK_LBUTTON) == MK_LBUTTON)
{
	CClientDC dc(this);

	dc.MoveTo(m_iPrevX , m_iPrevY);
	dc.LineTo(point.x , point.y);

	m_iPrevX = point.x;
	m_iPrevY = point.y;
}
```

به کد رسم خط بین دو نقطه توجه کنید، ملاحضه می کنید که ابتدا باید به مکان قبلی ماوس رفته و سپس خطی به مکان فعلی آن رسم کنیم. گام امل مهم است چون بدون آن ویندوز نمی تواند بداند که خط را از کجا باید شروع کند. اگر برنامه را اجرا کنید، خواهید دید که عملکرد آن کمی بهترشده است. ولی در ضمن رفتار عجیبی هم از خود نشان می دهد، هر بار که دکمه چپ ماوس را میگیرید تا چیزی رسم کنید، برنامه خطی از انتهای ترسیم قبلی به شروع خط جدید رسم می کند!

آخرین اصلاحات

برنامه ما زمانی شروع به رسم می کند که دکمه چپ ماوس را فشار دهیم. با ست کردن متغییر های مکان قبلی ماوس در لحضه کلیک ماوس، می توانید رفتار برنامه را اصلاح کنیم. برای انجام اصلاحات شبیه روش ساخت تابع **OnMouseMove** عمل می کنیم، نام کلاس دیالوگ برنامه یعنی **CMouseDlg** است را از **Class View** انتخاب کرده (با موس **highlight** کنید) و در پنجره **Properties** آیکن **Messages** را برای مشاهده پیام ها انتخاب نمایید. سپس پیام **WM_LBUTTONDOWN** را از لیست انتخاب و در پایان برای ساختن تابع از **Combo box** کنار آن گزینه **OnLButtonDown <add>** را انتخاب نمایید.
سپس در پنجره ادیتور تابع مرد نظر با عنوان **OnLButtonDown** ایجاد شده است که باید کد زیر را در این تابع بنویسید.

```
m_iPrevX = point.x;
m_iPrevY = point.y;
```

حال اگر برنامه را کامپایل و اجرا کنید خواهید دید که برنامه بسیار بهتر عمل می کند.

بدام انداختن رویدادهای کی بورد

خواندن رویدادهای کی بورد بسیار شبیه رویدادهای ماوس است. در مورد کی بورد هم رویدادهای برای فشردن و رها کردن کلیدهای کی بورد وجود دارد. رویدادهای کی بورد را در جدول زیر مشاهده نمایید.

پیام	مفهوم
WM_KEYDOWN	کلیدی زده شده است
WM_KEYUP	کلیدی رها شده است

کی بورد دارای پیام های اندکیست، اما باید بدانید که با کی بورد هم می توان کارهای زیادی انجام داد. علاوه بر کنترل ها حتی خود دیالوگ هم می تواند پیام کلیدهای کی بورد را بگیرد، البته مشروط به اینکه هیچ کنترلی فوکوس را در اختیار نداشته باشد، در غیر این صورت تمام پیام های کی بورد به کنترل دارای فوکوس خواهند رسید(به همین دلیل قبلا تمامی کنترلهای برنامه را حذف کرده ایم).

تغییر دادن کرسر

برای آشنایی بیشتر با پیام های کی بورد، در این قسمت سعی می کنیم با زدن کلیدی خواص کرسر برنامه نقاشی را تغییر دهیم. مثلا کاری میکنیم که با زدن کلید A کرسر پیشفرض انتخاب شود، با زدن کلید B کرسر به I تبدیل شود و با زدن کلید C کرسر به ساعت شنی تغییر حالت دهد. برای این منظور طبق روشهای بالا تابعی برای پیام WM_KEYDOWN سخته و کد زیر را در آن تابع یعنی OnKeyDown بنویسید.

```
char IsChar;
HCURSOR IhCursor;
IsChar = char(nChar);

if(IsChar == 'a' || IsChar == 'A')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
     SetCursor(IhCursor);
}

if(IsChar == 'b' || IsChar == 'B')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
     SetCursor(IhCursor);
}

if(IsChar == 'c' || IsChar == 'C')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
     SetCursor(IhCursor);
}

if(IsChar == 'x' || IsChar == 'X')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
     SetCursor(IhCursor);
     OnOK();
}
```

در خط اول تعریف تابع مشاهده می کنید که تابع OnKeyDown دارای سه ورودی است. ورودی اول کلید زده شده است. این ورودی در واقع کد کلید زده شده را برمی گرداند و قبل از هر کاری باید آنرا به یک کارکتر تبدیل کرد. بعد از این تبدیل است که می توان روی آن مقایسه انجام داد. دومین ورودی تابع OnKeyDown تعداد دفعات زده شدن کلید است. معمولا وقتی کلیدی زده و بلافاصله رها می شود این ورودی یک خواهد بود، اما اگر کلیدی را نگه دارید این ورودی به سرعت افزایش خواهد یافت و با رها شدن کلید تعداد تکرارها در این ورودی به ویندوز گزارش خواهد شد. ورودی سوم تابه OnKeyDown پرچمیست که نشان می دهد که آیا کلید Alt همزمان با کلید دیگر زده شده یا خیر، این پرچم نمی تواند حالت کلیدهای Ctrl یا Shift را گزارش کند. در دستورهای if مشخص می شود که کدام کلید زده شده است و چون روشن بودن کلید Caps Lock مشخص نیست هردو کارکتر کوچک و بزرگ را به وصیله OR منطقی تست می نماییم.
پس از مشخص شدن اینکه کدام کلید زده شده است نوبت به تغییر دادن کرسر می رسد که فرایندی دو مرحله ای است. اولین مرحله عبارت است از بار کردن کرسر در حافضه، این کار با تابع LoadStandardCursor انجام می شود. این تابع بعد از بار کردن کرسر استاندارد ویندوز شماره شناسائی آنرا برمی گرداند. بعد از بار شدن کرسر در حافظه، شماره شناسایی آن به تابع SetCursor داده می شود تا شکل کرسر عوض شود. اگر برنامه را کامپایل و اجرا کنید، خواهید دید که با زدن کلیدهای مزبور می توانید شکل کرسر را تغییر دهید. اما به محض شروع رسم، کرسر دوباره به حالت اولیه خود باز می گردد. در قسمت بعدی خواهید دید که چگونه می توان این مشکل را برطرف کرد.

ثابت کردن شکل کرسر

مشکل برنامه نقاشی آنست که با هر حرکت ماوس کرسر آن از نو روی صفحه رسم می شود. باید راهی برای متوقف کردن این وضعیت وجود داشته باشد. هر بار که کرسر به هر علتی مانند حرکت کردن، جابجا شدم پنجره ها و ... به ترسیم مجدد نیاز داشته باشد، یک پیام WM_SETCURSOR به برنامه شما فرستاده می شود. اگر رفتار پیش فرض این رویداد را تغییر دهیم، شکل کرسر ثابت باقی خواهد ماند البته تا زمانی که مجدد آنرا تغییر دهیم.
برای انجام این کار به روش قبلی متغییر جدیدی در کلاس CmouseDlg از نوع BOOL و با نام m_bCursor و به صورت Private تعریف کنید. سپس باید مقدار آنرا در تابع OnInitDialog برابر با FALSE قرار دهید، یعنی کد زیر را به تابع اضافه نمایید.

```
m_bCursor = FALSE;
```

و سپس، هنگام تغییر دادن شکل کرسر، در تابع OnKeyDown پرچم m_bCursor را به TRUE ست کنید. یعنی این تابع را به شکل زیر اصلاح نمایید.

```
 char IsChar;
HCURSOR IhCursor;
IsChar = char(nChar);
if(IsChar == 'a' || IsChar == 'A')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
     SetCursor(IhCursor);
}

if(IsChar == 'b' || IsChar == 'B')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
     SetCursor(IhCursor);
}
```

```
if(IsChar == 'c' || IsChar == 'C')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
     SetCursor(IhCursor);
}

if(IsChar == 'x' || IsChar == 'X')
{
     IhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
     m_bCursor = TRUE;
     SetCursor(IhCursor);
     OnOK();
}
else
{
     m_bCursor = TRUE;
     SetCursor(IhCursor);
}
```

در آخر برای پیام WM_SETCURSOR پنجره دیالوگ برنامه یک تابع بسازید و تابع را به شکل زیر اصلاح نمایید.

```
BOOL CMouseDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
     // TODO: Add your message handler code here and/or call default
     if(m_bCursor)
          return TRUE;
     else
          return CDialog::OnSetCursor(pWnd, nHitTest, message);
}
```

تابع OnSetCursor یا همیشه TRUE بر میگرداند یا تابعی را که خود از آن مشتق شده اجرا می کند. تابعی که OnSetCursor از آن مشتق شده کرسر ماوس را به حالت اول برمیگرداند. به همین دلیل است که مقدار متغییر m_bCursor قبل از آنکه کابر کلیدی را بزند باید FALSE باشد تا تابع OnSetCursor رفتار طبیعی خود را داشته باشد. اما وقتی کابر شکل کرسر را با زدن کلید مربوطه تغییر داد، ست کردن پرچم مزبور باعث می شود تا کرسر ثابت بماند و در واقع جلوی رفتار طبیعی OnSetCursor گرفته می شود. بدین ترتیب کابر می تواند شکل کرسر را تغییر داده و با آن نقاشی کند.

نکته: برای استفاده از کرسر های غیر استاندارد ویندوز یا آنهایی که خودتان ساخته اید، باید از تابع LoadCursor استفاده کنید. مثلا اگر کرسری را با استفاده از ادیتور منبع Visual C++ ساخته و نام آنرا IDC_MYCURSOR گذاشته اید، می توانید با دستور ذیل از آن استفاده کنید.

```
IhCursor = AfxGetApp()->LoadCursorW(IDC_MYCURSOR);
```

آموزش مقدماتی MFC

بهزاد جناب

فصل پنجم

ساختن آیکون در سیستم ترای ویندوز (آیکون های بغل ساعت ویندوز)

برای اینکه برنامه ما یک آیکون در سیتم ترای ویندوز داشته باشد باید یک ساختار از نوع NOTIFYICONDATA تعریف کنیم. برای استفاده از این ساختار به سر فایل shlwapi.h نیاز داریم بنابر این به وسیله دستور include زیرآنرا به برنامه اضاف کنید. برای اینکه ساختار ما از درون تمامی توابع برنامه قابل دسترسی باشد باید آن را به صورت عمومی(Global) تعریف کنیم لذا تعریف آنرا در بالای برنامه در کنار فراخانی سرفایل آن قرار می دهیم چون ممکن است در هر جای برنامه بخواهیم تغییراتی در آن ایجاد کنیم یا پیامی را نمایش دهیم.

```
#include "shlwapi.h"
NOTIFYICONDATA m_SysTryIcon;
```

در مرحله بعد در تابع آماده سازی یعنی OnInitDialog کدهای زیر را اضافه نمایید.

```
m_SysTryIcon.hWnd=this->GetSafeHwnd();
m_SysTryIcon.uID = 0x1;
m_SysTryIcon.cbSize=NOTIFYICONDATA_V2_SIZE;
m_SysTryIcon.hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
m_SysTryIcon.uFlags = NIF_ICON | NIF_TIP | NIF_MESSAGE;
StrCpy(m_SysTryIcon.szTip, L"نام برنامه");
m_SysTryIcon.uCallbackMessage=WM_USER+75;
Shell_NotifyIcon(NIM_ADD,&m_SysTryIcon);
```

توضیحات

hWnd یک گیره به پنجره ای که پیامهای این آیکون را دریافت می کند

uID یک شماره شناسایی است. در مواردی که برنامه از چند آیکون همزمان در سیستم ترای سیستم استفاده میکند کاربرد دارد

cbSize اندازه ساختار به بایت

hIcon یک گیره به آیکونی که در سیستم ترای نمایش داده می شود

uFlags این عضو یک مجموعه پرچم است که در ذیل توضیحات سه پرچم استفاده شده در کد بالا قرار دارد

۱. پرچم NIF_ICON که در آن ست شده است باعث می شود تا آیکون برنامه که توسط عضو hIcon آنرا مشخص کرده ایم در پایین نمایش داده شود.
۲. پرچم NIF_TIP باعث نمایش متن درون عضو szTip به صورت ToolTip (توضحیاتی که با نگه داشتن ماوس بر روی اشیا مختلف ظاهر می شوند) می شود.
۳. پرچم NIF_MESSAGE که باعث می شود در صورت فشردن کلیدهای ماوس بر روی آیکون سیستم ترای پیامی که در عضو uCallbackMessage قرار دارد به پنجره دیالوگ برنامه ارسال شود.

szTip به وصیله تابع **StrCpy** متنی را در این عضو کپی کرده ایم که در صورت قرار گرفتن کرسر بر روی آیکون برنامه در سیستم ترای یک متن راهنمای هاوی این متن نمایش داده می شود.
uCallbackMessage این عضو دارای شماره پیامی است که در صورت فعالیت رویدادهای موس(راست کلیک و ...) بر روی آیکون ما واقع در سیستم ترای سیستم به پنجره متصل به آن ارسال می شود.
در آخر پس از ست کردن خواص مورد نظر برای ایجاد آیکون سیستم ترای تابع **Shell_NotifyIcon** را با پارامتر **NIM_ADD** فراخانی می کنیم.

حذف آیکون سیستم ترای در هنگام خروج یا اجرای برنامه

قبل از خروج از برنامه باید ساختار ساخته شده برای نمایش آیکون در سیستم ترای را حذف نماییم، گاهی هم ممکن است در حال اجرای برنامه بخواهیم این کار را انجام دهیم. به هر دلیل برای حذف آیکون به هنگام خروج باید برای پیام **WM_DESTROY** دیالوگ مورد نظر یک تابع بسازید (این پیامی است که هنگام خروج از برنامه اجرا خواهد شد) و کد زیر را در آن قرار دهید تا آیکون را از سیتم ترای ویندوز حذف نماید.
برای حذف آیکون سیستم ترای به هنگام اجرای برنامه نیز از همین دستور می توانید استفاده کنید

```
Shell_NotifyIcon(NIM_DELETE,&m_SysTryIcon);
```

تشخیص کلیک شدن ماوس بر روی آیکون سیستم ترای برنامه

چنانچه پرچم **NIF_MESSAGE** را در عضو **uFlags** ست کرده باشیم با فشردن کلید های ماوس بر روی آیکون برنامه یک پیام به پنجره دیالوگ برنامه ارسال می شود، پیامی که قبلا در عضو **uCallbackMessage** قرار داده ایم یعنی پیام **WM_USER+75**. بنابرین یک تابع می سازیم و این پیام را به آن متصل می کنیم.
با فرض اینکه نا پروژه **SysTry** است برای اتصال این پیام به تابع در قسمت بالای فایل دیالوگ برنامه(**SysTryDlg.CPP**) به دنبال عبارت **BEGIN_MESSAGE_MAP(CSysTryDlg, Cdialog)** بگردید و کد زیر را در میان پیامها اضافه کنید.

```
ON_MESSAGE (WM_USER+75, OnSystemTryMessage)
```

این کد مشخص میکند که تابع **OnSystemTryMessage** (می توان هر نامی برای این تابع انتخاب نمود) در صورت رسیدن این پیام(که همان کلیک ماوس بر روی آیکون سیستم ترای برنامه است) به پنجره برنامه فراخوانی خواهد شد.
حالا از پنجره **Class View** بر روی کلاس دیالوگ برنامه (**CSysTryDlg**) راست-کلیک کرده و گزینه **Add->Add Function** را انتخاب نمایید، سپس تابعی با مشخصات زیر برای این کلاس بسازید.

```
LRESULT OnSystemTryMessage (WPARAM wParam, LPARAM lParam)
```

راهنمای ساختن تابع برای کلاس

و کد زیر را در آن بنویسید

```
switch(lParam)
{
case WM_LBUTTONDOWN:
	this->ShowWindow(SW_NORMAL);   //   باعث نمایش پنجره دیالوگ جاری می شود
	SetForegroundWindow();   //   باعث نمایش پنجره در جلوی دیگر پنجره ها می شود
	SetFocus();   //   باعث ست شدن فوکوس بر روی پنجره می شود
break;
case WM_RBUTTONDOWN:
	this->ShowWindow(SW_HIDE);   //   باعث مخفی شدن پنجره دیالوگ جاری می شود
break;
}
```

این تابع در صورت کلیک ماوس بر روی آیکون سیستم ترای برنامه فراخانی می شود. پارامتر **lParam** در این تابع مشخص می کند که کدام کلید ماوس فشرده شده است. پس از بررسی آن با دستور **switch** بوسیله دستور **ShowWindow** در صورت فشرده شدن کلید راست پنجره مخفی و با کلیک چپ ماوس دوباره نشان داده می شود.

نمایش داده نشدن پنجره برنامه هنگام شروع

برای اینکه در هنگام شروع پروژه پنجره برنامه ما نمایش داده نشود تا اینکه کابر با کلیک بر روی آیکون سیستم ترای پنجره را نمایش دهد باید ابتدا برای کلاس دیالوگ برنامه یک متغییر از نوع **BOOL** با نام **m_visible** و با دسترسی **private** تعریف کنید(در پنجره **class view** بر روی کلاس دیالوگ برنامه راست-کلیک کرده و گزینه ...**Add->Add Variable** را انتخاب کنید سپس در پنجره باز شده فیلد **Access** را به **private** تغییر دهید و در فیلد **Variable Type** متن **BOOL** را تایپ کنید و در قسمت **Variable Name** نام متغیر یعنی **m_visible** را تایپ کنید).

حال باید پیام WM_WINDOWPOSCHANGING را override کنید و کد درون آنرا به شکل زیر تغییر دهید(کلاس دیالوگ برنامه را انتخاب کنید و در پنجره Peroperties بر روی Messages کلیک کنید و از لیست پیامها، پیام WM_WINDOWPOSCHANGING را پیدا کرده و با کلیک بر روی فیلد جلوی آن یک تابع که با نام OnWindowPosChanging است برای آن پیام بسازید).

```
if(!m_visible)
     lpwndpos->flags &= ~SWP_SHOWWINDOW;
CDialog::OnWindowPosChanging(lpwndpos);
```

و در نهایت کد درون تابع OnSystemTryMessage (تابعی که در بالا نوشته ایم و با کلیک بر روی آیکون سیستم ترای برنامه را نمایش می دهد) را به شکل زیر اصلا کنید.

```
switch(lParam)
{
case WM_LBUTTONDOWN:
     m_visible = TRUE;
     this->ShowWindow(SW_NORMAL);
     SetForegroundWindow();
     SetFocus();
break;
case WM_RBUTTONDOWN:
     this->ShowWindow(SW_HIDE);
break;
}
```

مخفی شدن پنجره پروژه هنگام انتخاب کلید Minimize

برای مخفی شدن پنجره هنگام کلیک بر روی دکمه Minimize باید تابعی برای پیام WM_SIZE پنجره دیالوگ بسازیم و کد آنرا به صورت زیر تغییر دهیم.

```
if(nType == SIZE_MINIMIZED)
     ShowWindow(SW_HIDE);
else
     CDialog::OnSize(nType, cx, cy);
```

نمایش منو در صورت راست کلیک کردن بر روی آیکون سیستم ترای

در واقع کاربر با کلید بر روی سیستم ترای باعث نمایش یک منوی باز شونده (Popup) می شود. برای نمایش یک منوی باز شونده ابتدا یک منو با آیدی IDR_MENU1 می سازیم و با دستور TrackPopupMenu آنرا فراخانی می کنیم. (برای ساخت منو در فضای خای پنجره Resource View راست-کلیک کرده و گزینه ...Add Resource را انتخاب نمایید سپس از قسمت Resource Type عبارت Menu را انتخاب و بر روی دکمه New کلیک کنید)

تابع OnSystemTryMessage را به شکل زیر اصلاح کنید.

```
if(lParam == WM_RBUTTONDOWN) // شرط در صورت فشرده شدن کلید راست اجرا می شود
{
      CMenu mnu;     // تعریف یک ساختار از نوع منو و بار کردن منو در حافظه
      mnu.LoadMenu(IDR_MENU1);
      CMenu *PopUpMenu;
      PopUpMenu=mnu.GetSubMenu(0);
      SetForegroundWindow();
      CPoint pt;   // تعریف ساختار برای ذخیره نقطه ماوس در آن
      GetCursorPos(&pt);     // ذخیره مکان فعلی ماوس
      PopUpMenu->TrackPopupMenu(TPM_RIGHTALIGN,pt.x,pt.y,this); // نمایش منو
}
```

نکته: برای ساختن منو بر روی پنجره منابع پروژه(Resource View) راست کلیک کرده و از منوی که باز می شود عبارت Add Resource ... را انتخاب می نماییم. سپس در پنجره به نمایش درآمده عبارت Menu را انتخاب کرده و بر روی دکمه New کلیک کنید. سپس در پنجره منابع پروژه قسمتی به نام منو اضافه خواهد شد. با دابل کلیک بر روی نام آن که به صورت پیش فرض IDR_MENU1 است آن را ویرایش نمایید. برای تغییر ID خود منو و فارسی کردن آن باید نام آن را در پنجره منابع پروژه انتخاب نمایید، سپس در پنجره Properties فیلد زبان آنرا فارسی قرار دهید و ID انرا نیز در صورت نیاز تغییر دهید.

تغییر خواص و مشخصات سیستم ترای برنامه مانند آیکون و متن راهنما به هنگام اجرا

برای تغییر خواص و از جمله آیکون سیستم ترای برنامه در هنگام اجرای آن باید پس از تغییر مشخصات ساختار سیستم ترای مورد نظرمان تابع Shell_NotifyIcon را با پارامتر NIM_MODIFY فراخانی نمایید یعنی برای تغییر آیکون و توضیحات در هنگام اجرای برنامه کدی به صورت زیر می نویسیم. (برای برنامه قبلا آیکونی با ID به نام IDI_ICON1 ساخته ایم.)

```
m_SysTryIcon.hIcon = AfxGetApp()->LoadIcon(IDI_ICON1);
StrCpy(m_SysTryIcon.szTip, L"تغییر آیکون برنامه");
Shell_NotifyIcon(NIM_MODIFY,&m_SysTryIcon);
```

نمایش بالون در سیستم ترای

برای نمایش بالون در سیستم ترای باید پرچم NIF_INFO را در عضو uFlags ساختار سیستم ترای مورد نظر ست کنیم. سه عضو مربوط به نمایش بالون وجود داد که یکی پرچم dwInfoFlags است که بوسیله آن می توانید نوع آیکون بالون مورد نظر را تعیین کنید و دو عضو دیگر به نامهای szInfo و szInfoTitle که جهت نگهداری اطلاعتی هستند که باید در متن اصلی و تیتر بالون نمایش داده شوند.پس از ست کردن این اعضا با فراخانی تابع Shell_NotifyIcon با پارامتر NIM_MODIFY تغییرات را اعمال و بالون را به نمایش در می آوریم و در نهایت دوباره پرچم نمایش بالون را برمی داریم تا ناخواسته بالون نمایش داده نشود(مثلا وقتی می خواهیم آیکون سیستم ترای را عوض کنیم با اجرای دستور Shell_NotifyIcon دوبار بالون نمایش داده می شود). کد را به صورت زیر می نویسیم:

```
m_SysTryIcon.uFlags = m_SysTryIcon.uFlags | NIF_INFO; // ست کردن پرچم نمایش بالون
m_SysTryIcon.dwInfoFlags = NIIF_INFO;    // مشخص کردن نوع آیکون بالون
StrCpy(m_SysTryIcon.szInfo,_T("متن اصلی بالون")); //کپی کردن متن اصلی بالون
StrCpy(m_SysTryIcon.szInfoTitle,_T("تیتر بالون"));    //  کپی کردن متن تیتر بالون
Shell_NotifyIcon(NIM_MODIFY,&m_SysTryIcon);    //  اعمال تغییرات و نمایش بالون
m_SysTryIcon.uFlags = m_SysTryIcon.uFlags ^ NIF_INFO; // حذف کردن پرچم نمایش بالون
```

توضیحات

dwInfoFlags این عضو یک پرچم است که مشخص کننده نوع آیکون نمایش داده شده در بالون است و یکی از سه پرچم زیر را می توان در آن ست نمود

NIIF_INFO این پرچم تعیین کننده آیکون اطلاعات است یعنی آیکونی با یک حرف i در وسط آن

NIIF_ERROR این پرچم تعیین کننده آیکون خطا است که به شکل یک ضربدر در وسط یک دایره قرمز رنگ است

NIIF_NONE این پرچم به این معنی است که بالون ما هیچ آیکونی ندارد

szInfo این عضو حاوی متنی است که در بالون نمایش داده می شود

szInfoTitle این عضو تیتر بالای بالون است که به صورت بزرگتری نشان داده می شود که در صورت خالی بودن نمایش داده نمیشود و فقط متن اصلی که درون عضو **szInfo** است به نمایش در خواهد آمد.

محو کردن بالون به نمایش در آمده در سیستم ترای

بالون های برنامه پس از مدتی خود به خود محو میشوند ولی برای محو کردن بالون فقط کافیست که عضو **szInfo** را برابر با یک رشته خالی قرار دهید و تغییرات را با تابع **Shell_NotifyIcon** اعمال نمایید.کد زیر را می نویسیم:

```
StrCpy(m_SysTryIcon.szInfo,_T(""));
Shell_NotifyIcon(NIM_MODIFY,&m_SysTryIcon);
```

آموزش مقدماتی MFC

بهزاد جناب

فصل ششم

اطلاعات اولیه درباره رجیستری ویندوز

اهمیتی که رجیستری در یک سیستم دارد ایجاب می نماید تا هر برنامه نویسی برای کارایی بهتر با آن آشنایی داشته باشد. قبل از شروع آموزش در مورد ساختار رجیستری ویندوز توضیحات کوتاهی ارائه می شود. رجیستری وبندوز یک پایگاه داده با ساختار درخت واره است، که از آن برای ذخیره و بازیابی تنظیمات پیکربندی های موجود در ویندوز ۳۲ بیتی استفاده می شود. سلسله مراتب **Registry** از عناصر زیر تشکیل شده است:

HKEY_CLASSES_ROOT

HKCR اطلاعات مربوط به فایل را در خود نگه می دارد. به این ترتیب که چه نوع فایل هایی با چه نوع برنامه ای کار می کند. به عنوان مثال در این قسمت است که مشخص می شود فایل با پسوند **txt** با نرم افزار **notepad** باز می شود. **HKCR** تعاریف هر یک از اشیاء موجود در محیط ویندوز را نیز در خود نگه می دارد.

HKEY_CURRENT_USER

HKCU حاوی پروفایل کاربری است که در حال حاضر از ویندوز استفاده می کند. یک پروفایل سیستم، سخت افزار و سیستم های برنامه سفارشی سازی شده برای یک کاربر خاص را شامل می شود. تمامی این اطلاعات در یک **Hive** با نام **USER.DAT** ذخیره می شوند.

HKEY_LOCAL_MACHINE

HKLM ورودی های **CPU**، گذرگاه سیستم و سایر اطلاعات پیکربندی سخت افزاری به وسیله ویندوز در هنگام شروع ویندوز را شامل می شود.

HKEY_USERS

HKU اطلاعات پروفایل برای کاربران محلی کامپیوتر را شامل می باشد. حداقل دو ورودی در **HKU** ظاهر می شود. اولین ورودی **Default** که حاوی گروه پیش فرض تنظیم ها که برای کاربرانی است که بدون پروفایل به سیستم وارد می شوند. دومین ورودی که آنرا همیشه در **HKU** مشاهده می کنید **Administrator** توکار است.

HKEY_CURRENT_CONFIG

HKCC اطلاعات مربوط به پروفایل سخت افزار در حال استفاده و اطلاعات وسیله ای که در خلال شروع ویندوز جمع آوری شده است را نگه می دارد.

هر کدام از این قسمتها شامل تعدادی زیر کلیدها و داده ها به صورت تودرتو و سلسله مراتبی هستند. اگر ساختار رجیستری ویندوز را با سیستم فایل آن مقایسه کنیم، کلیدها با فولدرها و داده ها با فایلها متناظر می شوند. همانطور که فولدرها برای دسته بندی فایلها به کار می روند، کلیدها هم برای دسته بندی داده های رجیستری استفاده می شوند. داده های رجیستری هم همانند فایلها حاوی اطلاعات مورد نیاز سیستم و یا کاربر هستند. هر کلید خود می تواند شامل چند زیر کلید باشد و . . .

ساختار درختی رجیستری ویندوز می تواند تا ۵۱۲ سلسه مراتب تودرتو داشته باشد. شما می توانید در یک مرحله اجرای توابع ساخت کلید در رجیستری ویندوز تا ۳۲ مرحله تو در تو کلید و زیرکلید ایجاد کنید. اسامی کلیدها می توانند حداکثر از ۲۵۵ کارکتر تشکیل شوند و اسامی داده های درون کلیدها هم می توانند تا ۱۶۳۸۳ کاراکتر داشته باشند. حداکثر حجم مقادیر ذخیره شده در داده ها نیز ۱ مگابایت است.

نکته: تمامی توابع کار با رجیستری در صورت اینکه با موفقیت اجرا گردند مقدار برابر با ERROR_SUCCESS برمیگردانند که جهت مطلع شدن از درستی اجرای این توابع می توان در یک شرط if خروجی آنها را بررسی کرد.

تهیه پشتیبان از رجیستری و بازیابی دوباره آن

قبل از هر تغییری در رجیستری ویندوز حتما از آن نسخه پشتیبان تهیه کنید. برای اینکار می توانید از نرم افزار Registry Editor که به صورت رایگان همراه ویندوز نصب می شود استفاده کنید. برای اجرای آن از منوی Start گزینه Run را اجرا نمایید و در پنجره جدید متن regedit را بنویسید و بر روی OK کلیک کنید. پس از اجرای نرم افزار برای پشتیبان گیری از رجیستری از منوی File گزینه Export را انتخاب نمایید و نام و مسیر فایل را مشخص کنید و بر روی گزینه Save کلیک کنید. برای بازیابی نیز از گزینه Import و فایلی که قبلا ذخیره کرده ایم استفاده می کنیم.

نوشتن یک رشته از نوع CString در رجیستری ویندوز

قبل از هر کاری باید یک زیر کلید جدید بوسیله دستور RegCreateKeyExW ایجاد نماییم، سپس بوسیله دستور RegSetValueExW یک داده جدید با نام MySTR و از نوع REG_SZ تعریف کنیم و مقدار رشته را در آن قرار داده و در نهایت با دستور RegCloseKey رجیستری مورد نظر را می بندیم. تابع RegCreateKeyExW در صورت موجود بودن کلید مورد نظر آنرا باز می کند اما چنانچه کلید مورد نظر ما وجود نداشت آن را می سازد.

قصد داریم برنامه ای بنویسیم که یک متن را توسط یک Edit Control از کاربر بگیرد و آن را در رجیستری ذخیره کند. برای نوشتن متن تایپ شده در رجیستری اول یک متغییر از نوع CString برای Edit Control که متن کابر را می گیرد می سازیم با فرض اینک نام متغییر وابسته به آن m_strRegInUser است کدی به صورت زیر برای نوشتن رشته در رجیستری می نویسیم.

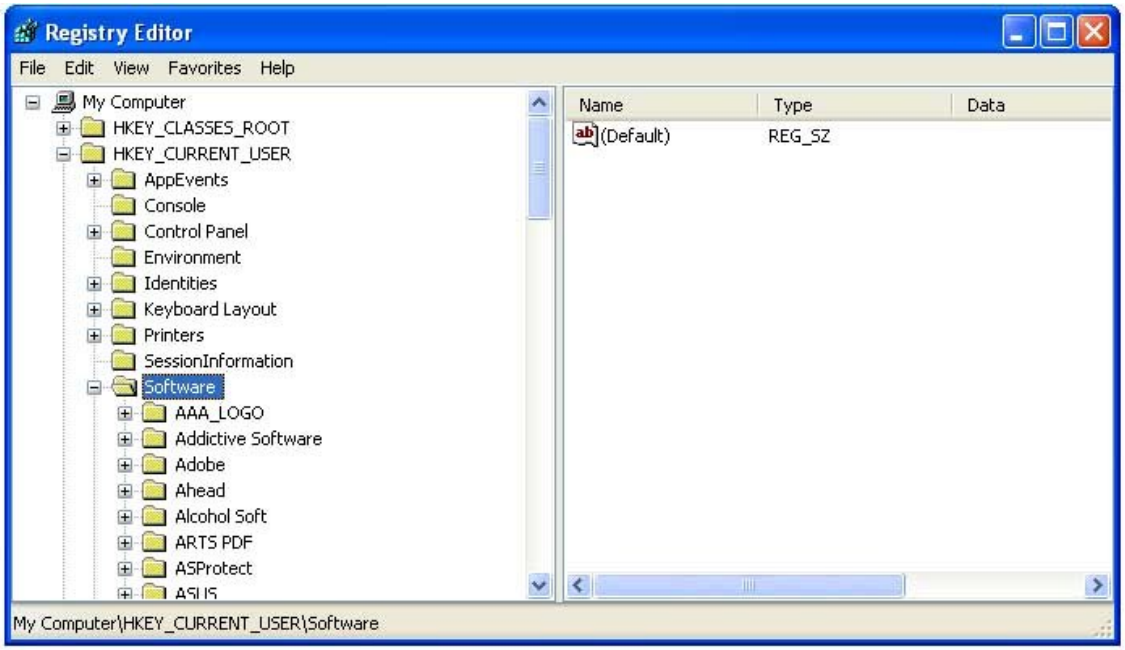
```
HKEY hkey;
DWORD dwDisp;
UpdateData(TRUE); // قرار گرفتن متن تایپ شده توسط کاربر در متغیر وابسته به آن
wchar_t buf[255]={0};
wcscpy_s(buf,m_strRegInUser);   //تبدیل رشته از نوع "سی استرینگ" به "ویچار" توسط این تابع
wchar_t *strMyRegValue=buf;
long errorcheck;
errorcheck=RegCreateKeyExW( // تابعی که جهت باز کردن کلید از آن استفاده می شود که در صورت نبود کلید آنرا می ساز
      HKEY_CURRENT_USER,     // یکی از مسیرهای اصلی رجیستری ویندوز
      L"Software\\Mahtab_V1",          // نام زیر کلیدی که تابع باید ساخته و یا باز کند
      0,          //   این پارامتر همیشه باید صفر باشد
      NULL,
      REG_OPTION_NON_VOLATILE, // مشخص کننده موقتی یا دائمی بودن کلید ساخته شده در رجیستری ویندوز است
      KEY_ALL_ACCESS,              // تعیین محدوده دسترسی به کلید
      NULL,
      &hkey,      //   یک اشاره گر به متغیری از نوع رجیستری که اطلاعات کلید باز شده در آن قرار می گیرد
      &dwDisp); // اطلاعات اینکه آیا کلید جدیدی بازشده یا از قبل در رجیستری موجود بوده در این متغییر قرار می گیرد
if(errorcheck == ERROR_SUCCESS) //   در صورت اجرای موفقیت آمیز تابع بالا خروجی آن برابر با شرط خواهد بود
{
      errorcheck=RegSetValueExW( //تابعی که داده کاربر را پس از ساختن مقدارجدید در آن  می نویسد
```

```
            hkey,   //    متغییر هاوی اطلاعات زیر کلیدی که قبلا آنرا باز نموده اید
            L"MySTR",   //نام داده ای که پس از ایجاد آن در زیر کلید مقدار کاربر باید در آن قرار گیرد
            0,   // این پارامتر همیشه باید صفر باشد
            REG_SZ,         //مشخص می کند که نوع داده ما رشته ای از نوع یونی کد است
            (LPBYTE) strMyRegValue,   // اطلاعات درون رشته مورد نظر
            (DWORD)(lstrlen(strMyRegValue)+1)*sizeof(TCHAR));//حافظه مورد نیاز جهت ذخیره
        RegCloseKey(hkey);   // تابعی که باعث بستن کلیدی که قبلا باز شده می شود
        if((errorcheck == ERROR_SUCCESS)
        {
          // در صورت اجرای موفقیت آمیز تابع بالا این شرط اجرا خواهد شد
        }
}
```

تابع **wcscpy_s** رشته درون متغیر **m_strRegInUser** که از نوع **CString** است را به نوع **wchar_t** تبدیل می کند تا بتوان آنرا بوسیله تابع **RegSetValueExW** در رجیستری ذخیره کرد.

برای کار با رجیستری قبل از هر کاری باید متغیری از نوع رجیستری بوسیله دستور **HKEY** تعریف کنیم سپس متغیری از نوع **DWORD** تعریف میکنیم که با اجرای تابع اطلاعاتی درون این متغییر قرار می گیرد که مشخص کننده چگونگی باز شدن یا ایجاد کلید است. سپس بوسیله دستور **RegCreateKeyEx** کلید یا زیرکلیدهای مورد نظر را برای تغییرات در آن باز می کنیم. این تابع در صورت موجود نبودن کلید یا زیر کلیدهای مشخص شده آنها را در رجیستری ایجاد می کند،

به طور استاندارد اطلاعات مربوط به هر برنامه ای که در سیستم نصب است می بایست در قسمت **HKEY_CURRENT_USER** رجیستری و در کلید **Software** ذخیره شود. اگر توسط نرم افزار **Regedit** به این آدرس مراجعه کنید اطلاعات برنامه های که بر روی سیتم شما نصب است را مشاهده خواهید کرد.



با وجود اینکه اجباری در استفاده از این مسیر نیست ولی ما هم از این استاندارد پیروی کرده و اطلاعات برنامه خود را در این مسیر ذخیره می کنیم.

پارامتر دوم این تابع نام و ترتیب زیرکلیدهایی است که قصد ساختن آنها را داریم. با فرض اینکه هنوز هیچ کلیدی نساخته ایم اگر این مسیر را برابر "Software\\Mahtab_V1" قرار دهیم تابع در کلید Software زیرکلید Mahtab_V1 را می سازد یا اگر مسیر را برابر با "Software\\Mahtab_V1\\Behzad" قرار دهیم تابع در کلید Software زیرکلید Mahtab_V1 را ایجاد کرده و سپس زیرکلیدی با نام Behzad در آن می سازد و به همین صورت ادامه خواهد داشت.

پارامتر پنجم این تابع مشخص کننده دائمی و یا موقتی بودن کلید ساخته شده توسط آن است چنانچه آنرا برابر با REG_OPTION_VOLATILE قرار دهید کلید های که با آن می سازید به صورت موقتی ساخته می شود و با شروع مجدد ویندوز از بین خواهند رفت و در صورتی که مقدار آنرا به REG_OPTION_NON_VOLATILE تغییر دهیم کلید ساخته شده به صورت دائمی و ثابت در رجیستری باقی خواهد ماند.

پارامتر آخر این تابع یک خروجی است که در متغیر dwDisp قرار می گیرد و فقط دو مقدار را بر می گرداند که معنی هرکدام به شکل زیر است.

REG_CREATED_NEW_KEY کلید موجود نبوده و توسط تابع ساخته شده است

REG_OPENED_EXISTING_KEY کلید موجود بوده و بدون ایجاد تغییری در آن فقط باز شده است

تابع RegSetValueExW رشته ورودی کاربر را پس از ایجاد داده ای با نام MySTR در آن می نویسد. پارامتر دوم تابع نام داده ای است که باید ایجاد و مقدار ورودی کاربر در آن قرار گیرد. پارامتر چهارم تابع مشخص می کند که نوع داده ما رشته ای از نوع یونی کد است.

خروجی دو تابعی که در بالا توضیح داده شد در صورت اینکه با موفقیت اجرا گردد برابر با مقدار ERROR_SUCCESS خواهد شد که جهت مطلع شدن از درستی اجرای این دو تابع می توان در شرط if آنها را بررسی کرد.

خواندن یک رشته از نوع CString از یک کلید در رجیستری

برای خواندن رشته ای که قبلا آنرا در رجیستری نوشته ایم ابتدا توسط تابع RegOpenKeyEx کلیدی که هاوی داده ما است را باز می کنیم سپس با دستور RegQueryValueEx مقدار ذخیره شده درون داده ای به نام MySTR را می خوانیم.

با فرض اینکه متغیری از نوع CString به یک Static Text نصبت داده و نام آنرا m_strRegOutData گذاشته ایم کد زیر را می نویسیم.

```
HKEY hkey;
long errorcheck;
errorcheck=RegOpenKeyEx(   //  تابع کلید را در صورت موجود بودن باز می کند
      HKEY_CURRENT_USER,   //  یکی از مسیرهای اصلی رجیستری ویندوز
      L"Software\\Mahtab_V1",          //  مسیر و نام کلید و  زیرکلید مورد نظر
      0,          //   این پارامتر همیشه باید صفر باشد
      KEY_ALL_ACCESS,   //   مشخص کننده محدوده دسترسی ما به کلید است
      &hkey);   //  یک اشاره گر به متغیری از نوع رجیستری که اطلاعات کلید باز شده در آن قرار می گیرد
if(errorcheck == ERROR_SUCCESS) //    در صورت اجرای موفقیت آمیز تابع بالا این شرط اجرا خواهد شد
{
      wchar_t buf[255] = {0};
      DWORD dwType,
            dwBufSize=500L; //    مقدار حافظه ای که برای بافر اختصاص داده ایم به بایت را در این متغیر مشخص میکنیم
      errorcheck=RegQueryValueEx( //تابعی که مقدار درون داده مورد نظر ما را می خواند
```

```
            hkey, // یک اشاره گر از نوع رجیستری که اطلاعات کلید باز شده در آن قراردارد
            L"MySTR", // نام داده ای که قصد خواندن مقدار درون آنرا داریم
            0, // این پارامتر همیشه باید صفر باشد
            &dwType, // نوع اطلاعات ذخیره شده است در این داده را بر می گرداند
            (BYTE*)buf, // اطلاعات درون داده را برمی گرداند
            &dwBufSize); // میزان فضای استفاده شده توسط داده را بر می گرداند
      RegCloseKey(hkey); // تابعی که باعث بستن کلیدی که قبلا باز شده می شود
      if(errorcheck == ERROR_SUCCESS) // در صورت اجرای موفق تابع بالا شرط اجرا می شود
      {
            m_strRegOutData=buf; // داده خوانده شده از رجیستری در متغیر برای نمایش قرار می گیرد
            UpdateData(FALSE);
      }
}
```

نوشتن یک عدد از نوع long در رجیستری

بوسیله دستور **RegCreateKeyExW** زیر کلیدی که قصد ایجاد داده جدید در آن داریم را باز می نماییم، سپس بوسیله دستور **RegSetValueExW** یک داده جدید با نام **MyNUM** و از نوع **REG_DWORD** تعریف کنیم و عدد را در آن قرار داده و در نهایت با دستور **RegCloseKey** رجیستری مورد نظر را می بندیم. تابع **RegCreateKeyExW** در صورت موجود بودن کلید مورد نظر آنرا باز می کند اما چنانچه کلید مورد نظر ما وجود نداشت آن را می سازد.

قصد داریم برنامه ای بنویسیم که یک عدد حداکثر ده رقمی را توسط یک **Edit Control** از کاربر بگیرد و آن را در رجیستری ذخیره کند. برای نوشتن عدد تایپ شده توسط کاربر در رجیستری اول یک متغییر از نوع **long** برای **Edit Control** که عدد کابر را می گیرد می سازیم با فرض اینک نام متغییر وابسته به آن **m_longNumberInUser** است کدی به صورت زیر برای نوشتن عدد در رجیستری می نویسیم.

```
HKEY hkey;
DWORD dwDisp,cpData=4;
UpdateData(TRUE);
DWORD longMyRegValue=m_longNumberInUser;
long errorcheck;
errorcheck=RegCreateKeyExW( // تابع کلید را ساخته و یا در صورت موجود بودن آن را باز می کند
      HKEY_CURRENT_USER, // یکی از مسیرهای اصلی رجیستری ویندوز
      L"Software\\Mahtab_V1", // مسیر و نام زیرکلیدی که باید ساخته یا باز شود
      0, // این پارامتر همیشه باید صفر باشد
      NULL,
      REG_OPTION_NON_VOLATILE, // مشخص کننده دائمی یا موقتی بودن کلید ساخته شده است
      KEY_ALL_ACCESS, // کلید به دسترسی محدوده تعیین
      NULL,
      &hkey, // یک اشاره گر به متغیری از نوع رجیستری که اطلاعات کلید باز شده در آن قرار می گیرد
      &dwDisp); // اطلاعات اینکه آیا کلید جدیدی بازشده یا از قبل در رجیستری موجود بوده در این متغییر قرار می گیرد
if(errorcheck == ERROR_SUCCESS) // در صورت اجرای موفق تابع بالا این شرط اجرا می شود
{
```

```
	errorcheck=RegSetValueExW(	//تابعی که داده کاربر را پس از ساختن مقدارجدید در آن  می نویسد
		hkey,	//	متغییر هاوی اطلاعات زیر کلیدی که قبلا آنرا باز نموده اید
		L"MyNUM",	//نام داده ای که پس از ایجاد آن در زیر کلید مقدار کاربر باید در آن قرار گیرد
		0,	// این پارامتر همیشه باید صفر باشد
		REG_DWORD,	//مشخص می کند که نوع داده ما یک عدد ۳۲ بیتی است
		(LPBYTE)	&longMyRegValue,	// اطلاعات عددی که کار بر وارد نموده جهت ذخیره در داده
		cpData);	//حافظه مورد نیاز جهت ذخیره داده
	RegCloseKey(hkey);	// تابعی که باعث بستن کلیدی که قبلا باز شده می شود
	if((errorcheck == ERROR_SUCCESS)
	{
		// در صورت اجرای موفقیت آمیز تابع بالا در نوشتن مقدار در داده این شرط اجرا خواهد شد
	}
}
```

خواندن یک عدد از نوع long از رجیستری

برای خواندن عددی از نوع **long** که قبلا آنرا در رجیستری نوشتیم ابتدا توسط تابع **RegOpenKeyEx** کلیدی که هاوی داده ما است را باز کرده و سپس با دستور **RegQueryValueEx** مقدار ذخیره شده درون داده مورد نظر که در اینجا نام آن **MyNUM** است را می خوانیم.
با فرض اینکه متغیری از نوع **long** به یک **Static Text** نصبت داده و نام آنرا **m_longRegValueOut** گذاشته ایم کد زیر را برای خواندن مقدار داده ذخیره شده با نام **MyNUM** و از نوع **long** می نویسیم.

```
HKEY hkey;
long errorcheck;
errorcheck=RegOpenKeyEx(	// تابع کلید را در صورت موجود بودن باز می کند
	HKEY_CURRENT_USER,	// یکی از مسیرهای اصلی رجیستری ویندوز
	L"Software\\Mahtab_V1",		// مسیر و نام کلید و  زیرکلید مورد نظر
	0,		// این پارامتر همیشه باید صفر باشد
	KEY_ALL_ACCESS,	//	مشخص کننده محدوده دسترسی ما به کلید است
	&hkey);	// یک اشاره گر به متغیری از نوع رجیستری که اطلاعات کلید باز شده در آن قرار می گیرد
if(errorcheck == ERROR_SUCCESS) //	در صورت اجرای موفقیت آمیز تابع بالا این شرط اجرا خواهد شد
{
	DWORD buf = 4;
	DWORD dwType = REG_DWORD;
	DWORD dwBufSize = sizeof(DWORD);
	errorcheck=RegQueryValueEx(	//تابعی که مقدار درون داده مورد نظر ما را می خواند
		hkey,	// یک اشاره گر از نوع رجیستری که اطلاعات کلید باز شده در آن قراردارد
		L"MyNUM",	//نام داده ای که قصد خواندن مقدار درون آنرا داریم
		0,	// این پارامتر همیشه باید صفر باشد
		&dwType,	// نوع اطلاعات ذخیره شده است در این داده را بر می گرداند
		(PBYTE)&buf,	// اطلاعات ذخیره شده در درون داده را برمی گرداند
```

```
            &dwBufSize); // میزان فضای استفاده شده توسط داده را بر می گرداند
      RegCloseKey(hkey); // تابعی که باعث بستن کلیدی که قبلا باز شده می شود
      if(errorcheck == ERROR_SUCCESS) // در صورت اجرای موفق تابع بالا شرط اجرا می شود
      {
            m_longRegValueOut=buf; //داده خوانده شده از رجیستری در متغیر برای نمایش قرار می گیرد
            UpdateData(FALSE);
      }
}
```

حذف داده از یک کلید در رجیستری

برای حذف داده ابتدا کلید نگهدارنده داده را توسط دستور RegOpenKeyEx باز کرده سپس با دستور RegDeleteValue داده مورد نظر را پاک می کنیم.

```
HKEY hkey;
long errorcheck;
errorcheck=RegOpenKeyEx( // تابع کلید را در صورت موجود بودن باز می کند
      HKEY_CURRENT_USER, // یکی از مسیرهای اصلی رجیستری ویندوز
      L"Software\\Mahtab_V1", // مسیر و نام کلید و زیرکلید مورد نظر
      0, // این پارامتر همیشه باید صفر باشد
      KEY_ALL_ACCESS, // مشخص کننده محدوده دسترسی ما به کلید است
      &hkey); // یک اشاره گر به متغیری از نوع رجیستری که اطلاعات کلید باز شده در آن قرار می گیرد
if(errorcheck == ERROR_SUCCESS) // در صورت اجرای موفقیت آمیز تابع بالا این شرط اجرا خواهد شد
{
      errorcheck= RegDeleteValue( // تابعی که باعث پاک شدن داده از کلید می شود
            hkey, // مشخصات کلیدی که داده ما در آن قرار دارد
            L"MySTR"); // نام داده ای که قصد پاک کردن آنرا داریم
      RegCloseKey(hkey); // تابعی که باعث بستن کلیدی که قبلا باز شده می شود
      if(errorcheck == ERROR_SUCCESS)
      {
            // در صورت موفقیت تابع بالا در پاک کردن داده این شرط اجرا می شود
      }
      else
      {
            //در صورت ناموفق بودن تابع بالا در پاک کردن داده این شرط اجرا می شود
            //ممکن است داده مورد نظر در کلید ما وجود نداشته باشد
      }
}
else
{
      //در صورت ناموفق بودن تابع در باز کردن کلید این شرط اجرا می شود
      //ممکن است کلید یا زیرکلید مورد نظر وجود نداشته باشد
}
```

حذف کلید از رجیستری

بوسیله تابع **RegDeleteKey** می توانید یک کلید را حذف کنیم. باید توجه داشته باشید که فقط در صورتی کلید مورد نظر پاک خواهد شد که هیچ زیرکلیدی در آن وجود نداشته باشد در غیر اینصورت قبل از پاک کردن آن باید تمامی زیر کلیدهای داخل آنرا پاک کنید. کدی که در زیر نوشته ایم باعث پاک شدن کلید **Mahtab_V1** در مسیر **HKEY_CURRENT_USER\Software** رجیستری می شود.

```
long errorcheck;
errorcheck=RegDeleteKey(   //  تابعی که یک کلید را در صورت نداشتن زیرکلید پاک می کند
      HKEY_CURRENT_USER,L"Software\\Mahtab_V1");   //مسیر و نام کلیدی که قصد داریم پاک کنیم
if(errorcheck==ERROR_SUCCESS)
{
  //  در صورت پاک شدن کلید این شرط اجرا می شود
}
else
{
  //  در صورت بروز خطا در پاک کردن کلید این شرط  اجرا خواهد شد
}
```

شمارش کلیدهای یک مسیر از رجیستری

برای شمارش کلیدهای یک مسیر از تابع **RegEnumKeyEx** استفاده می کنیم.

کد زیر نام تمامی کلیدهای موجود در مسیر **HKEY_CURRENT_USER\Software** را در یک آرایه از نوع **CString** می نویسد.

```
HKEY hkey;
DWORD dwIndex=0,dKeyNameBufSize=256;
wchar_t cahrKeyName[256]={0};
CString strKeyNames[100];
if(ERROR_SUCCESS==RegOpenKeyExW(HKEY_CURRENT_USER, L"Software", 0, KEY_ENUMERATE_SUB_KEYS, &hkey))
{
	while(ERROR_NO_MORE_ITEMS!=RegEnumKeyExW(hkey ,dwIndex ,cahrKeyName ,&dKeyNameBufSize ,NULL ,NULL, NULL, NULL))
	{
		strKeyNames[dwIndex]=cahrKeyName;
		dwIndex++;
		dKeyNameBufSize=256;
	}
	RegCloseKey(hkey);
}
```

dwIndex پارامتر دوم این تابع یک شماره است که برای بار اول صفر و برای دفعات بعد یک واحد به آن می افزاییم و مشخص کنند چندمین نام کلیدی است که می خواهید اطلاعات آن را دریافت کنیم.

cahrKeyName پارامتر سوم یک متغییر رشته ای و یک بافر برای ذخیره نام کلید است.

dKeyNameBufSize پارامتر چهارم یک ورودی-خروجی است، یعنی قبل از صدا زدن تابع باید مقدار بافری را که برای نام کلید در نظر گرفته ایم در آن قرار دهیم(چنانچه اندازه نام کلید از مقدار بافری که مشخص کرده ایم بیشتر باشد خروجی تابع **ERROR_MORE_DATA** خواهد بود) و بعد از صدا زدن تابع مقدار درون آن برابر با تعداد کارکترهای تشکیل دهنده نام کلید به صورت یک رشته **TCHARs** خواهد بود ، به همین دلیل پس از هر بار صدا زدن تابع مجددا این متغیر را برابر با ۲۵۶ قرار می دهیم.

وقتی که دیگر هیچ کلیدی در مسیر نباشد تابع **ERROR_NO_MORE_ITEMS** را بر می گرداند .

شمارش داده های یک مسیر از رجیستری

برای شمارش داده های یک مسیر از تابع **RegEnumValue** استفاده می کنیم.

کد زیر نام داده های موجود در مسیر **HKEY_CURRENT_USER\Software\Mahtab** را در یک متغییر از نوع **CString** می نویسد.

```
HKEY hkey;
DWORD dwIndex=0 ,dRegNameBufSize=256;
wchar_t cahrValueName[256]={0};
CString strValueNames[100];
if(ERROR_SUCCESS==RegOpenKeyExW(HKEY_CURRENT_USER, L"Software\\Mahtab", 0, KEY_QUERY_VALUE, &hkey))
{
	while(ERROR_NO_MORE_ITEMS!=RegEnumValueW(hkey ,dwIndex ,cahrValueName ,&dRegNameBufSize ,0 ,0 ,0 ,0))
	{
		strValueNames[dwIndex]=cahrValueName;
		dwIndex++;
		dRegNameBufSize=256;
	}
	RegCloseKey(hkey);
}
```

dwIndex پارامتر دوم این تابع یک شماره است که برای بار اول صفر و برای دفعات بعد یک واحد به آن می افزاییم و مشخص کنند چندمین نام داده ای است که می خواهید اطلاعات آن را دریافت کنیم.

cahrKeyName پارامتر سوم یک متغییر رشته ای و یک بافر برای ذخیره نام داده است.

dKeyNameBufSize پارامتر چهارم یک ورودی-خروجی است، یعنی قبل از صدا زدن تابع باید مقدار بافری را که برای نام داده در نظر گرفته ایم در آن قرار دهیم(چنانچه اندازه نام داده از مقدار بافری که مشخص کرده ایم بیشتر باشد خروجی تابع **ERROR_MORE_DATA** خواهد بود) و بعد از صدا زدن تابع مقدار درون آن برابر با تعداد کارکترهای تشکیل دهنده نام داده به صورت یک رشته **TCHARs** خواهد بود، به همین دلیل پس از هر بار صدا زدن تابع مجددا این متغیر را برابر با ۲۵۶ قرار می دهیم.

وقتی که دیگر هیچ داده ای در مسیر نباشد تابع **ERROR_NO_MORE_ITEMS** را بر می گرداند .

نکته: تابع **RegEnumValue** همزمان با نام داده ها مقدار و نوع آنها را نیز برمیگردان که در پارامترهای ششم تا هشتم قرار دارند که توضیح آن به صورت زیر است (اما چون در بالا ما نیازی به آنها نداشتیم برابر با **NULL** یا همان صفر قرار داده ایم).

پارامتر ششم یک اشاره گر به متغیری است که نوع مقدار ذخیره شده در این داده را بر می گردان (مانند **REG_DWORD** ، **REG_SZ** و ...).

پارامتر هفتم یک اشاره گر به بافری است که مقدار ذخیره شده درون داده را دریافت می کند.

پارامتر هشتم اشاره گر به متغیری از نوع خروجی-ورودی است، به این ترتیب که قبل از صدا زدن تابع میزان بافر (اندازه متغیر برابر با پارامتر هفتم) در نظر گرفته شده برای ذخیره مقدار داده را در آن قرار می دهیم و پس از اجرای تابع متغیر برابر با تعداد کارکتر های خوانده شده از داده است.

آموزش مقدماتی MFC

بهزاد جناب

فصل هفتم

افزودن تایمر به برنامه

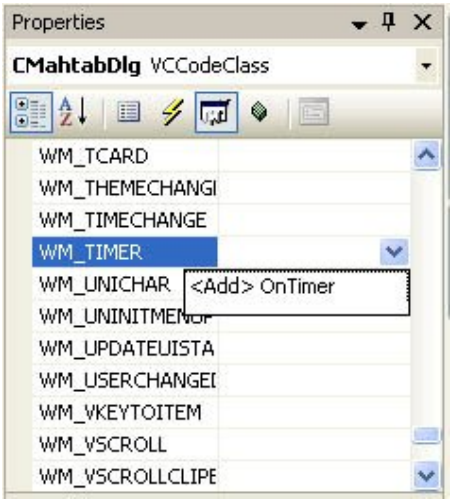
چنانچه قصد داشته باشید کاری را به فواصل معین زمانی در برنامه انجام دهید باید یک تایمر تعریف کنید. برای تعریف تایمر بر روی فضای خالی پنجره **Resource View** پروژه راست-کلیک کرده و از منوی باز شده گزینه **Resource Symbols...** را انتخاب نمایید. بر روی دکمه **New** کلیک کنید و در پنجره جدید باز شده در فیلد **Name** یک نام برای تایمرتان انتخاب نمایید (در اینجا ما **ID_MMTIMER** را به عنوان نام قرار می دهیم).

تا اینجا تایمر را تعریف کرده ایم ولی برای شروع به کار آن باید آنرا فعال کنیم. برای فعال کردن تایمر از دستور زیر استفاده می کنیم

```
SetTimer(ID_MMTIMER, 1000, NULL);
```

با اجرای دستور فوق تایمر ما با گامهای معیین شده شروع به کار می کند و در هر گامی یک پیام **WM_TIMER** به برنامه ارسال می کند. برای تغییر گامهای تایمر پس از آغاز به کار آن نیز از همین دستور استفاده می شود. پارامتر اول نام تایمر است و پارامتر دوم این تابع میزان گامهای تایمر را بر حسب میلی ثانیه تعیین می کند، که در اینجا ۱۰۰۰ میلی ثانیه(برابر با یک ثانیه) است. پس این تایمر هر یک ثانیه یکبار تحریک می شود (پیامهای تایمر فقط در هنگام بیکاری برنامه ارسال می شوند، یعنی مثلا چنانچه برنامه در یک تابع دیگر مشغول به انجام کاری باشد تمامی پایمها تایمر ارسال شده از دست خواهند رفت).

با هر بار تحریک این تایمر یک پیام **WM_TIMER** به پنجره دیالوگ برنامه ارسال می شود، بنابراین چنانچه یک تابع برای این پیام بسازید با هر بار تحریک زمان این تابع اجرا خواهد شد. برای ساختن تابع برای این پیام نام کلاس دیالوگ مورد نظر را از **Class View** انتخاب کرده (با موس **highlight** کنید) و در پنجره **Properties** آیکن **Messages** را برای مشاهده پیام ها انتخاب نمایید، سپس پیام **WM_TIMER** را از لیست انتخاب و در پایان برای ساختن تابع از **Combo box** کنار آن گزینه **add** را انتخاب نمایید تا تابع ساخته شود.



نکته: شما می توانید در یک برنامه از چند تایمر هم زمان استفاده کنید

غیر فعال کردن تایمر

برای غیر فعال کردن تایمر از دستور `KillTimer` به همراه نام تایمر مورد نظر به شکل زیر استفاده می کنیم

```
KillTimer(ID_MMTIMER);
```

آموزش مقدماتی MFC
بهزاد جناب
فصل هشتم

مباحث متفرقه

در این فصل مطالبی متنوع با موضوعات مختلف را می خوانید که جهت کمک به شما جهت تکمیل پروژه تان مفید است.

الگوریتم تبدیل تاریخ میلادی به تاریخ شمسی

تبدیل تاریخ میلادی به شمسی بسیار راحتتر از تبدیل تاریخ شمسی به میلادی است. برای نوشتن این الگوریتم به اختلاف روزهای میان اولین روز سال میلادی و اولین روز سال شمسی نیاز داریم که این اختلاف روز(در صورتی که سال کبیسه باشد یا نباشد ۷۹ روز است.

برای تشخیص کبیسه بودن یا نبودن سال از روش زیر استفاده می کنیم:
اگر سال داده شده بر۱۰۰و۴۰۰ بخشپذیر باشد یا بر ۱۰۰ بخشپذیر نباشد بر ۴ بخشپذیر باشد آنگاه سال کبیسه است، در غیر این صورت سال کبیسه نیست.
با توجه به کبیسه بودن یا کبیسه نبودن سال مشخص می کنیم که در کدامین روز سال میلادی قرار داریم.

دو حالت پیش می آید:

- روزی که در آن قرار داریم از ۷۹ بیشتر است
  به این معنی است که در ماههای بعد از فروردین قرار داریم.

حال باید مشخص کنیم که در ۶ ماه اول سال شمسی قرار داریم یا در ۶ ماه دوم سال قرار داریم،
برای اینکار ابتدا ۷۹ روز از تعداد روزها کم می کنیم تا در اول فروردین قرار بگیریم حال اگر تعداد روزها از "۱۸۶"(۶*۳۱) کمتر باشد یعنی در ۶ ماه اول سال شمسی قرار داریم در غیر اینصورت در ۶ ماه دوم قرار داریم .

۱. اگر در ۶ ماه اول سال قرار گرفته باشیم :تعداد روزها را بر ۳۱ تقسیم می کنیم(۶ ماه اول در سال شمسی ۳۱ روزه است).
   اگر باقیمانده این تقسیم صفر شد خارج قسمت تقسیم برابر با ماه شمسی می شود و روز شمسی برابر با ۳۱ می شود.
   اگر باقیمانده صفر نشود ماه شمسی برابر با خارج قسمت باضافه یک می شود و روز شمسی همان باقیمانده است.
۲. اگر در ۶ ماه دوم سال قرار گرفته باشیم : ۱۸۶ روز از تعداد روزها کم می کنیم و آن را بر ۳۰ تقسیم می کنیم.
   اگرباقیمانده این تقسیم صفر شد خارج قسمت تقسیم باضافه ۶ برابر با ماه شمسی می شود و روز شمسی برابر با ۳۰ می شود.
   اگر باقیمانده صفر نشود ماه شمسی برابر با خارج قسمت باضافه ۷ می شود و روز شمسی همان باقیمانده است.

سال شمسی از تفاضل سال میلادی با"۶۲۱" بدست می آید.

- روزی که در آن قرار داریم کمتر از ۷۹ است که این به این معنی است که درروزهایی بین اولین روزسال میلادی تا اولین روز شمسی (ماههای دی ،بهمن و اسفند)قرار داریم.

- اختلاف روز بین اولین روز سال میلادی داده شده و اولین روز دی ماه در سال شمسی را در نظر می گیریم که این اختلاف برای سال کبیسه ۱۱ و برای غیر کبیسه ۱۰ است. دقت کنید که در این الگوریتم برای مشخص کردن این اختلاف باید سال قبل از سال داده شده را در نظر بگیریم زیرا سال قبل بر روی اولین روز سال میلادی تاثیر می گزارد.

اختلاف روز را با تعداد روزهای محاسبه شده جمع می کنیم . آن را بر۳۰ تقسیم می کنیم (۳ ماه آخر سال شمسی ۳۰ روزه است).
اگر باقیمانده این تقسیم صفر شود خارج قسمت تقسیم باضافه ۹ برابر با ماه شمسی می شود و روز شمسی برابر با ۳۰ می شود.
اگر باقیمانده صفر نشود ماه شمسی برابر با خارج قسمت باضافه ۱۰ می شود و روز شمسی همان باقیمانده است.

در این حالت سال شمسی از تفاضل سال میلادی با ۶۲۲ بدست می آید. (زیرا در سال قبل قرار داریم)

پیاده سازی الگوریتم به زبان **VC++**:

```
typedef struct _SHAMSIDATE
{
    int iYear;
    int iMonth;
    int iDay;
}SHAMSIDATE;

SHAMSIDATE MiladiToShamsi(int iMiladiMonth,int iMiladiDay,int iMiladiYear)
{

    int  shamsiDay, shamsiMonth, shamsiYear;
    int  dayCount,farvardinDayDiff,deyDayDiff ;
    int  sumDayMiladiMonth[] = {0,31,59,90,120,151,181,212,243,273,304,334};
    int  sumDayMiladiMonthLeap[]= {0,31,60,91,121,152,182,213,244,274,305,335};
    SHAMSIDATE  shamsidate;

 farvardinDayDiff=79;

if (MiladiIsLeap(iMiladiYear))
 {
    dayCount = sumDayMiladiMonthLeap[iMiladiMonth-1] + iMiladiDay;
 }
else
 {
    dayCount = sumDayMiladiMonth[iMiladiMonth-1] + iMiladiDay;
 }
if((MiladiIsLeap(iMiladiYear - 1)))
 {
    deyDayDiff = 11;
 }
else
 {
    deyDayDiff = 10;
 }
if (dayCount > farvardinDayDiff)
{
    dayCount = dayCount - farvardinDayDiff;
    if (dayCount <= 186)
```

```
    {
     switch (dayCount%31)
      {
       case 0:
        shamsiMonth = dayCount / 31;
        shamsiDay = 31;
       break;
       default:
        shamsiMonth = (dayCount / 31) + 1;
        shamsiDay = (dayCount%31);
       break;
      }
     shamsiYear = iMiladiYear - 621;
    }
   else
     {
    dayCount = dayCount - 186;
    switch (dayCount%30)
      {
       case 0:
        shamsiMonth = (dayCount / 30) + 6;
        shamsiDay = 30;
       break;
     default:
       shamsiMonth = (dayCount / 30) + 7;
       shamsiDay = (dayCount%30);
       break;
     }
      shamsiYear = iMiladiYear - 621;
    }
  }
else
  {
    dayCount = dayCount + deyDayDiff;

    switch (dayCount%30)
    {
    case 0 :
      shamsiMonth = (dayCount / 30) + 9;
      shamsiDay = 30;
     break;
    default:
      shamsiMonth = (dayCount / 30) + 10;
      shamsiDay = (dayCount%30);
     break;
    }
    shamsiYear = iMiladiYear - 622;

  }
    shamsidate.iYear = shamsiYear;
    shamsidate.iMonth = shamsiMonth;
    shamsidate.iDay = shamsiDay;
```

```
  return shamsidate ;
}

// the function check a miladiyear is leap or not.
BOOL MiladiIsLeap(int miladiYear)
{
if(((miladiYear % 100)!= 0 && (miladiYear % 4) == 0) || ((miladiYear % 100)== 0
&& (miladiYear % 400) == 0))
  return TRUE;
else
  return FALSE;
}
```

اجرای یک پنجره دیالوگ دیگر در هنگام اجرای برنامه

برای اجرای پنجره دیالوگی دیگر در هنگام اجرای برنامه مثلا دیالوگ مخصوص توضیحات برنامه به شکل زیر عمل می کنیم

```
CDialog aboutDlg(IDD_ABOUTBOX);
aboutDlg.DoModal();
```

شیشه ای کردن پنجره دیالوگ

کد زیر باعث شیشه ای شدن پنجره می شود. متغییر **a** در اینجا عددی بین ۰ تا ۲۵۵ است و مقدار شفافیت پنجره را مشخص می کند به صورتی که با عدد صفر پنجره دیده نمی شود و هرچه این عدد از یک تا ۲۵۵ بیشتر شود پنجره نمایانتر می شود. اگر متغییر **a** را برابر با ۲۵۵ قرار دهیم پنجره به صورت کامل نمایش داده می شود و دیگر پشت پنجره دیده نخواهد شد.

```
SetWindowLong(this->GetSafeHwnd(),GWL_EXSTYLE,
     GetWindowLongPtr(this->GetSafeHwnd(), GWL_EXSTYLE) | WS_EX_LAYERED);
SetLayeredWindowAttributes(0, a, LWA_ALPHA);
```

تغییر عکس میز کار (Desktop)

برای اینکار ابتدا شما نیاز به اضافه کردن دو سر فایل **wininet.h** و **shlobj.h** دارید، **wininet.h** باید بعد از **stdafx.h** و قبل از **shlobj.h** قرار بگیرد (مثلا به صورت زیر).

```
#include "stdafx.h"
#include "wininet.h"
#include "TestMFC.h"
#include "TestMFCDlg.h"
#include <shlobj.h>
#include <comdef.h>
```

بعد از این مرحله یک اشاره گر به واسط **IActiveDesktop** ایجاد میکنیم. این تعریف رو در فایل **h** برنامه انجام می دیم و به اصطلاح یک عضو **private** از کلاس **Dialog** برنامه ایجاد میکنیم.

```
private:
     IActiveDesktop *pActive;
```

به تابع **OnInitDialog** رفته و کدهای زیر رو در اون قرار میدیم :

```
HRESULT hRes;
CoInitialize(NULL);
hRes = ::CoCreateInstance(
CLSID_ActiveDesktop,
NULL,
CLSCTX_INPROC_SERVER,
IID_IActiveDesktop,
(void**)&pActive);
if ( hRes != S_OK )
return FALSE;
```

یک **Button** روی فرم اضافه کنین و کد زیر رو در اون وارد می کنیم :

```
if ( pActive->SetWallpaper(_T("C:\\A.jpg"),NULL) == S_OK )
{
pActive->ApplyChanges(AD_APPLY_ALL);
MessageBox(_T("Set Wallpapaer"));
}
::CoUninitialize();
```

الان اگه برنامه رو اجرا کنین با اجرا **Button** ، تصویر **A** که در درایو **C** قرار داره به عنوان **Wallpaper** انتخاب میشه و شما میتونین تغییرات رو مشاهده کنین.

نکته: اما اگر باز در هنگام اجرا برنامه پیامی مبنی بر ناشناس بودن **IActiveDesktop** دریافت کردید، فایل **stdafx.h** رو باز کرده و **wininet.h** رو در محل زیر قرار بدید :

```
// turns off MFC's hiding of some common and often safely ignored warning
messages
#define _AFX_ALL_WARNINGS
#include <afxwin.h> // MFC core and standard components
#include <afxext.h> // MFC extensions
//**********************************
//**********************************
#include <wininet.h>
//**********************************
//**********************************

#include <afxdisp.h> // MFC Automation classes
```

توضیحات در مورد کدهایی که در OnInitDialog قرار گرفتن

CoInitialize

قبل از اینکه ما بتونیم از مولفه های **COM** استفاده کنیم باید کتابخانه های **COM** رو حافظه بار کنیم. با این کار میتونیم استفاده از یک اشاره گر به تخصیص دهنده حافظه ، به متود های مورد نظر دسترسی داشته باشیم.

CoCreateInstance

اول بهتره چند تا تعریف داشته باشیم. هر **COM** شامل تعدادی کلاس که **CoClass** نا میده میشن و تعدادی **Interface** هست که در برگیرنده متود های مورد نظر هستن وجود داره. برای استفاده از کلاس های **COM** باید اونها **Register** شده باشن. هر کلاس دارای یک کد ۶۴ **bit**ی و یا ۱۲۸ **bit**ی ، یکتا هست. به عبارت دیکه شما دو **CoClass** با کد های یکسان پیدا نمیکنین. وقتی شما یک **COM** رو در سیستم **Register** میکنین ، برای هرکلاس یک کد در **Registery** سیستم عامل ثبت میشه. همینطور در مورد **Interface** ها.

تابع **CoCreateInstance** یک شئ از کلاس مورنظر ما روی سیستم محلی **(local)** ایجاد میکنه.
آرگومان اول **id** کلاس مورد نظر رو میگیره.
آرگومان دوم اشاره گری هست به واسط **IUnknown** که فعلا به اون کاری نداریم و به اون مقدار **NULL** میدیم.
آرگومان سوم وضعیت **object** جدیدی که ایجاد شده رو مشخص میکنه. که مقدار اون رو **CLSCTX_INPROC_SERVER** در نظر گرفته شده. به این معنی که **Object** مورد نظر در یک **Dll** قرار داره که در همون چرخه اجرا میشه.
آرگومان چهارم هم که **id** واسط مورد نظر هست.
آرگومان پنجم هم آدرس اشاره گر مورد نظر ما به واسط **IActiveDesktop** هست. که اون رو به صورت **Private** تعریف کردیم.
مقدار برگشتی این تابع از نوع **HRESULT** هست که یا **S_OK** در صورت موفقیت و یا **S_FALSE** در صورت پش اومدن مشکل هست.

در پایان هم باید با استفاده از **CoUninitialize** حافظه ای که برای بار شدن کتابخانه ی **COM** در نظر گرفته شده آزاد بشه.

مشخص کردن تعداد درایوهای متصل به سیستم

برای مشخص نمودن تعداد و نام درایوهای متصل به سیستم از تابع `GetLogicalDriveStrings` استفاده می کنیم. کد زیر نام تمامی درایوهای سیستم را در یک آرایه به نام `strDriveName` قرار می دهد و در پایان کد عدد باقی مانده در متغیر `intDriveNum` تعداد درایوهای متصل به سیستم را مشخص میکند.

```
DWORD nBufferLength=100;
wchar_t charDrive[100];
CString strDriveName[20];
int intDriveNum=0, c=0, i;
for(i=0;i<100;i++) charDrive[i]=_T(' ');
GetLogicalDriveStringsW(nBufferLength,charDrive);
wchar_t buf[5]={0};
i=0;
while(charDrive[i]!=_T(' '))
{
     buf[c]=charDrive[i];
     c++;
     if(charDrive[i]==0)
     {
          c=0;
          strDriveName[intDriveNum]=buf;
          ++intDriveNum;
     }
     i++;
}
intDriveNum--;
```

خواندن نام فایلها و پوشه های یک مسیر

برای این منظور از دو تابع FindFirstFile و FindNextFile استفاده می کنیم. کد زیر نام تمامی فایلها و پوشه های موجود در درایو D را در یک آرایه از نوع CString قرار میدهد. در پایان مقدار متغیر i برابر با تعداد فایلها و پوشه های پیدا شده است.

```
CString strFileName[256];
WIN32_FIND_DATA MyFileInfo;
HANDLE hFind;
wchar_t charPath[256]=L"D:\\*";
BOOL bTest;
int i;
hFind = FindFirstFile(charPath, &MyFileInfo);
if (hFind != INVALID_HANDLE_VALUE)
{
     i=0;
     do
     {
          strFileName[i]=MyFileInfo.cFileName;
          bTest=FindNextFile(hFind ,&MyFileInfo);
          i++;
     }while(bTest!=NULL);
     FindClose(hFind);
}
```

متغیر charPath باید مسیر و ماسک را مشخص کند، مثلا برای همه فایلها * و برای تمامی فایلهای اجرایی *.exe را در آخر مسیر قرار می دهیم. تابع FindFirstFile در صورت موفقیت یک شماره گیره را بر میگرداند که از آن در تابع FindNextFile استفاده می کنیم و چنانچه موفق نشود مقدار INVALID_HANDLE_VALUE را برمیگرداند. سپس از تابع FindNextFile برای فایلهای و پوشه های بعدی استفاده می کنیم، این تابع در صورت موفقیت یک مقدار غیر از صفر برمیگرداند و در صورت ناموفق بودن مقدار صفر را برمیگرداند.
متغیر MyFileInfo که از نوع ساختار WIN32_FIND_DATA تعریف کرده ایم داری مشخصات فایل پیدا شده توسط این دو تابع است که دو عضو مهم آنها در زیر آورده ام

cFileName نام و پسوند فایل
dwFileAttributes خاصیتهای فایل

در آخر هم با دستور FindClose کار را تمام می کنیم.

تغییر خواص یک فایل یا پوشه

برای تغییر خواص یک فایل یا پوشه از تابع SetFileAttributes استفاده می کنیم. کد زیر باعث مخفی شده فایل Test.TXT در مسیر ریشه درایو D می شود.

```
SetFileAttributes(L"D:\\Test.TXT",FILE_ATTRIBUTE_HIDDEN);
```

اما برای اینکه دیگر خواص فایل را تغییر ندهیم باید ابتدا خواصیت فایل را با دستور GetFileAttributes بخوانیم سپس با ست کردن پرچمها فقط خاصیت مورد نظر را برداریم، چون در دستور بالا مثلا اگر فایل فقط خواندنی باشد پس از اجرای دستور پاک می شود.

کد زیر بدون تغییر در دیگر خواص فایل فقط فایل را از حالت مخفی خارج می کند.

```
DWORD dAttributes;
wchar_t charPathAndFile[256]=L"D:\\Test.TXT";
dAttributes=GetFileAttributes(charPathAndFile);
if((dAttributes&FILE_ATTRIBUTE_HIDDEN)==FILE_ATTRIBUTE_HIDDEN)
{
     dAttributes=dAttributes ^ FILE_ATTRIBUTE_HIDDEN;
     SetFileAttributes(charPathAndFile ,dAttributes);
}
```

حذف فایل

برای حذف یک فایل از تابع DeleteFile استفاده می کنیم، این تابع قادر به حذف فایلهایی با خاصیت فقط-خواندنی نیست و در صورت اجرای دستور مقدار ERROR_ACCESS_DENIED را بر میگرداند و همچنین در صورت وجود نداشتن فایل ERROR_FILE_NOT_FOUND را برگردانده می شود. تابع در صورت موفقیت عدد غیر-صفر و در صورت عدم موفقیت عدد صفر را برمیگرداند. کد زیر فایل Test.TXT را از مسیر ریشه درایو D پاک می کند.

```
DeleteFile(L"D:\\Test.TXT");
```