سئوال

برنامه نویسی شی گرا یعنی چه و چه تفاوتی با برنامه نویسی ساخت یافته دارد؟

منظور از کلاس در برنامه نویسی شی گرا چیست

وقتی گفته می شود که یک زبان شی گرا هست منظور این است که این زبان سه ویژگی زیر را پشتیبانی می کند:

۱.کپسوله سازی:encapsulation

۲.وراثت:Inheritance

۳.چند ریختی:polymorphism

کپسوله سازی:

برنامه نویسی شی گرا (Oriented Programming Object یا OOP) داده ها (خصوصیات) و توابع(رفتار) را در بسته هائی به نام کلاس محصور می کند.و از این طریق جزئیات پیاده سازی در داخل خود کلاس ها پنهان هستند.(فقط اشیاء کلا سهای دیگر می دانند که فلان شی از فلان کلاس ،فلان رفتار را انجام میدهد ولی نمی دانند که این شی چگونه این رفتار را انجام می دهد)

وراثت:

یعنی یک کلاس از یک کلاس دیگر ارث می برد.ارث بری صورتی از قابلیت استفاده مجدد است.فرض کنید یک کلاس به نام دانشجو دارید که رفتار و خصوصیاتی را برایش تعریف کرده اید حالا می خواهید یک کلاس دانشجوی کارشناسی تعریف کنید . فکر می کنید کدام روش مناسبتر است : یک کلاس کاملا جدید تعریف کنید یا اینکه کلاس دانشجوی کارشناسی را نوعی از دانشجو قرار بدهید.

اگر روش دوم را انتخاب کنید یک برنامه نویس حرفه ای هستید. با این کار کلاس دانشجوی کارشناسی از کلاس دانشجو ارث می برد یعنی کلاس دانشجوی کارشناسی تمام خصوصیات یک دانشجو را به ارث می برد و علاوه بر آن می توانید خصوصیات و رفتار های دیگری را علاوه بر خصوصیات دانشجو ،که مختص یک دانشجوی کارشناسی است به آن اضافه کنید.

در این حالت به کلاس دانشجو <کلاس والدclass parent یا پایه >و به کلاس دانشجوی کارشناسی <کلاس مشتق شدهdrived class > گفته می شود .دو نوع ارث بری داریم:یگانه و چند گانه

چند ریختی:

فرض کنید مجموعه ای از کلاس های هندسی مثل دایره و مثلث و مستطیل دارید که همه از کلاس پایهshape مشتق شده ، هر کدام از این کلاسها فرمول ریاضی خاص خود را برای محاسبه مساحت دارند. حال فرض کنید در کلاس والد، رفتار(تابع) area تعریف شده باشد در نتیجه هر کدام از کلاسهای مشتق شده تابعarea مخصوص به خودشان را دارند ولی نام تمام آنها همان area مربوط به کلاس والد است.این امکان با استفاده از پشتیبانی یک زبان از polymorphism به وجود می آید.

تعریف شی:

یک شی شامل مجموعه ای از متدها و در برخی موارد شامل ویژگی ها و رخدادهاست

تعریف کنترل:

کنترل به اشیایی گفته می شود که بر روی فرم نمایان می شود و در زمان اجرا کاربر می تواند با آنها تعامل داشته باشد

تعریف کامپوننت:

کامپوننت ها اشیای خاصی هستند که در زمان طراحی می توان ویژگی های آنها را دستکاری و تغییر داد

ویژگی :

مجموعه ای از داده هاست که یک شی در طول دوره حیات خود نگهداری می کند

متد :

عبارت است از مجموعه ای از اعمال که یک شی قادر به انجام آن می باشد

رخداد:

عبارت است از مجموعه ای از وقایع که یک شی قادر به تشخیص آن می باشد

تعریف کلاس

کلاس یک ساختار داده ای پیچیده است که شامل مجموعه ای از داده ها و توابع مورد نیاز برای کار روی داده ها می باشد . کلاس ها به ما این امکان را می دهند تا بتوانیم مجموعه داده و توابع و تکنولوژی کار با آن داده را به صورت کپسوله درآوریم . به داده ها و توابع یک کلاس اعضای کلاس می گویند. این اعضا می توانند به صورت خصوصی ، عمومی و یا مجازی باشند.

یک عضو خصوصی مختص همان کلاسی است که متعلق به آن است و دسترسی به آن در خارج از محدوده کلاس امکان پذیر نیست (فقط توابع عضو کلاس می توانند به عضوهای خصوصی کلاس دسترسی داشته باشند) بخش تعاریف خصوصی یک کلاس با کلید واژه `private` شروع می شود . در صورتی که عضو عمومی می تواند به ارث داده شود و دسترسی به آن هم در کلیه مشتقات کلاس امکان پذیر است بخش تعاریف عمومی یک کلاس با کلید واژه `public` شروع می شود.

یک عضو مجازی می تواند توسط وارث رونویسی شود و عملکرد آن مطابق میل و خواست وارث تغییر داده شود.
تعاریف مجازی یک کلاس با کلید واژه `virtual` شروع می شود

عضو های سازنده و مخرب در یک کلاس :

عضو سازنده در یک کلاس وظیفه مقدار دهی اولیه مقادیر مربوط به داده های کلاس را بر عهده دارد نام سازنده (`constructor`) با نام کلاس یکسان است . یک کلاس می تواند بیش از یک سازنده داشته باشد
عضوب مخرب (`destructor`) در یک کلاس وظیفه آزاد سازی حافظه در اختیار گرفته شده توسط داده های کلاس را برعهده دارد. نام عضو مخرب همان نام کلاس است که علامت ~ قبل از آن قرار گرفته است
هر دوی این عضو ها در زیر مجموعه اعضای عمومی کلاس قرار دارند.

پرسش های پایان درس

۱ – کدام گزینه در مورد یک زبان شی گرا صحیح می باشد

الف) پشتیبانی از وراثت ب) پشتیبانی از چند ریختی ج) پشتیبانی از کپسوله سازی د) همه موارد

۲ – کدام گزینه در مورد تعریف شی صحیح نمی باشد

الف) دسترسی به اعضای خصوصی اشیا توسط سایر اشیاء دیگر امکان پذیر نمی باشد

ب) یک شی شامل مجموعه ای از متدها و در برخی موارد شامل ویژگی ها و رخدادهاست

ج) یک شی همانند یک متغیر عمومی است

د) دسترسی به اعضای عمومی اشیا توسط سایر اشیاء دیگر امکان پذیر می باشد

۳ – توابع عضو در کلاس .....

الف) به همه اعضای کلاس دسترسی دارند ب) فقط به اعضای عمومی دسترسی دارند

ج) فقط به اعضای عمومی و مجازی دسترسی دارند د) فقط به اعضای خصوصی دسترسی دارند

٤ – عضو عمومی در کلاس .....

الف) در تمام مشتقات کلاس در دسترس می باشد ب) در مشتقات کلاس می توان آن را رونویسی کرد

ج) فقط در داخل کلاس در دسترس می باشد د) فقط در داخل فضای نام فعلی در دسترس می باشد

۵– عضو مجازی در کلاس .....

الف) در کلاس های مشتق شده قابل استفاده است

ب) می تواند پارامترها و شکل کلی آن توسط وارث تغییر داده شود

ج) وارث نمی تواند ساختار داخلی عضو مجازی را مطابق سلیقه خودش تغییر دهد

د) الف و ج

٦– عضو عمومی در کلاس با کلید واژه ..... مشخص می شود

۷– عضو خصوصی در کلاس با کلید واژه ..... مشخص می شود

۸– عضو مجازی در کلاس با کلید واژه ..... مشخص می شود

۹– مجموعه ای از داده هاست که یک شی در طول دوره حیات خود نگهداری می کند.

۱۰ –.... عبارت است از مجموعه ای از اعمال که یک شی قادر به انجام آن می باشد.

۱۱ – .... مجموعه ای از رخدادها / وقایع است که یک شی قادر به تشخیص آن می باشد.

ساختار کلی تعریف کلاس در c++ به شکل زیر می باشد

```
<classkey> <classname> <baselist> { <member list> };
```

<classkey> : یکی از کلید واژه های class ، struct و یا union می باشد **<classname>** :
تعیین کننده نام کلاس است . این نام باید در محدوده (scope) خودش یکتا باشد

<baselist> : لیست کلاس های پایه ای است که کلاس مورد نظر از آنها مشتق شده است **<baselist>**
اختیاری است و می تواند استفاده نشود

<member list> : توابع و داده های مربوط به کلاس را تعریف می کند

نکته۱ : در داخل یک کلاس به داده ها ، (داده عضو) Data Members و به توابع مربوط به داده ها، (توابع عضو) Member Functions می گوییم. در واقع می توان گفت کار توابع یک کلاس بر روی داده های همان کلاس است .

نکته ۲: گذاشتن علامت سمی کالن (;) انتهایی در تعریف کلاس الزامی است.

مثال : تعریف یک کلاس به نام Test

```
class  Test
{
…
};
```

نمونه سازی از کلاس

برای استفاده از کلاس باید نمونه ای از کلاس را ایجاد کنید. ایجاد نمونه ای از کلاس را نمونه سازی و نمونه ایجاد شده از کلاس را شی می گویند.

شکل کلی دستور ایجاد نمونه از یک کلاس به صورت زیر می باشد

در صورتی که بخواهیم متغیر نمونه از نوع اشاره گر باشد از فرم زیر استفاده می کنیم

```
ClassName *objectName=new ClassName();
```

در صورتی که بخواهیم متغیر نمونه از نوع اشاره گر نباشد از فرم زیر استفاده می کنیم

```
ClassName objectName;
```

که در آن className نام کلاس و objectName نام نمونه / شی می باشد.

مثال : ایجاد شی ی به نام objTest از کلاس Test

```
Test prtTest=new Test();
```

مثال : ایجاد شی ی به نام objTest از کلاس Test

```
Test objTest();
```

دسترسی به اعضای اشیاء

پس از اینکه یک نمونه از یک کلاس ایجاد شود اعضای عمومی (public) آن کلاس از طریق شی قابل دسترسی می باشد. برای دسترسی به این اعضا از نام شی / نمونه به همراه یک نقطه و سپس نام عضو عمومی مورد نظر عمل می کنیم

مثال : فرض کنید کلاس Test دارای یک عضو عمومی به نام x باشد و شی objTest از نوع کلاس Test تعریف شده باشد. در این صورت برای دسترسی به عضو x به شکل زیر عمل می کنیم

```
objTest.x
```

مثال : فرض کنید کلاس Test دارای یک عضو عمومی به نام x باشد و شی ptrTest ی اشاره گری(Pointer) از نوع کلاس Test تعریف شده باشد. در این صورت برای دسترسی به عضو x به شکل زیر عمل می کنیم

```
ptrTest->x
```

اعضای کلاس

اعضای کلاس می تواند یکی از موارد ثابت ها ، فیلد ها ، خواص ، متد ها ، سازنده ها ، مخرب ها و ... باشد

فرم کلی تعریف اعضای کلاس به شکل زیر می باشد

```
[سطح دستیابی:]
    classmembers
```

***سطح دستیابی** می تواند public و یا private باشد.*

تعریف اعضای ثابت

اعضای ثابت با کلید واژه const تعریف می شوند

مثال :

```
class Test
{
  Private:
      static const int x=10;
  public:
      static  const double PI=3.14;
};
```

نکته : امکان تغییر مقادیر ثابت در ادامه برنامه وجود ندارد.

تعریف فیلد ها

فیلد ها همانند متغیر های معمولی تعریف می شوند و برای ذخیره داده ها به کار می روند. نام دیگر فیلد ها متغیر نمونه است.

فرم کلی تعریف فیلد ها به شکل زیر می باشد

```
[سطح دستیابی:]
 نوع داده [static] نام فیلد ;
```

مثال :

```
class Test
{
  private:
    int x;
    static int y;
  public:
     int z;
};
```

تعریف متد ها

متد ها مجموعه ای از دستورالعمل ها هستند که عملیاتی را بر روی داده های کلاس انجام می دهند. فرم کلی تعریف متد به صورت زیر می باشد

```
[سطح دستیابی:]
 نوع داده  نام متد ([پارامترها]);
```

مثال :

```
class Test
{
   public:
    int Sum(int x,int y)
    {
      int s;
      s=x+y;
      return s;
    }
};
```

متد های همنام

دو متد در صورتی می توانند همنام باشند که از نظر تعداد ،ترتیب و یا نوع پارامترها با هم متفاوت باشند

متد های بازگشتی

اگر یک متد مجددا خودش را فراخوانی کند به آن متد بازگشتی می گوییم.

تعریف ویژگی ها / خواص

از نظر کد خارج از کلاس ویژگی ها همانند فیلد ها هستند و لی از نظر کلاس متفاوت می باشند. خاصیت ها دو کار را انجام می دهند ۱- داده های خود را مقدار می دهند ۲- مقدار داده ها را بازیابی می کنند.

شکل کلی(ساده شده) تعریف ویژگی / خاصیت به فرم زیر می باشد

__property نوع داده خروجی نام ویژگی = {read=متد مدیریت بخش خواندن,write=متد مدیریت بخش نوشتن} ;

مقدار دادن به خاصیت در قسمت write و بازیابی مقدار در قسمت read انجام می شود.

مثال :

```
class Test
{
    private:
        int _y;
        int getData(){return _y;} /* متد مدیریت بخش خواندن */
        void setData(int x){ _y=x;} /* متد مدیریت بخش نوشتن */
    public:
       __property int y={read=getData,write=setData};
};
```

نکته قابل توجه در مورد ویژگی ها این است که در موقع مقدار دهی در بخش write می توان کنترل نمود که داده های غیر مجاز توسط کاربر وارد نشود. نکته دیگر اینکه وقتی کد خارج از کلاس مقداری را به یک خاصیت نسبت دهد این مقدار توسط پارامتر تابع مربوط به بخش read قابل دسترسی می باشد.

نکته : چنانچه در تعریف یک ویژگی / خاصیت بخش read حذف شود آن ویژگی فقط نوشتنی و در صورتی که بخش write حذف شود آن ویژگی فقط خواندنی می باشد.

عضو سازنده کلاس

عضو سازنده کلاس همنام خود کلاس می باشد و می تواند دارای پارامتر باشد. یک کلاس می تواند بیش از یک سازنده داشته باشد

مثال :

```
class Test
{
   private:
     int a,b;
   public:
     Test (){a=0;b=0;}
     Test (int x,int y){a=x;b=y;}
};
```

عضو مخرب کلاس

عضو مخرب کلاس همنام خود کلاس می باشد که قبل از آن علامت ~ قرار دارد. یک کلاس نمی تواند بیش از یک عضو مخرب داشته باشد.عضو مخرب در زمانی که استفاده از شی ایجاد شده از کلاس تمام می شود به صورت خودکار اجرا می شود . بنابراین تمام مواردی که قرار است در پایان عمر شی انجام شود در این قسمت قرار می گیرد. مثلا فرض کنید یک کلاس برای کار با فایل ها ایجاد کرده باشید. عضو سازنده در ابتدای کار می تواند فایل را برای عملیات مورد نظر باز کند و در انتهای کار عضو مخرب باید عمل بستن فایل را انجام دهد

مثال :

```
class Test
{
   Private:
      int a,b;
   public:
   ~Test ()
    {
       …
    }

};
```

مثال : تعریف کلاس برای کار ذخیره سازی یک آرایه رشته ای ۱۰ تایی

```
#include <vcl.h>
#include <stdio.h>

class Famille
{
private:
AnsiString FNames[10];
AnsiString GetName(int Index);
void SetName(int, AnsiString);
public:
__property AnsiString Names[int Index] = {read=GetName,
write=SetName};
Famille(){}
~Famille(){}
};

AnsiString Famille::GetName(int i)
{
return FNames[i];
}

void Famille::SetName(int i,const AnsiString s)
{
FNames[i]=s;

}

int main()
{
Famille C;
C.Names[0]="Steve";   //calls Famille::SetName()

C.Names[1]="Amy";
C.Names[2]="Sarah";
C.Names[3]="Andrew";
for (int i = 0; i <= 3; i++)
{
//calls Famille::GetName()
puts(C.Names[i].c_str());
}
}
```

پرسش های پایان درس

۲- به یک متغیر Public که در یک کلاس تعریف شده باشد..... می گویند

۳- از نظر ..... بین فیلد و ویژگی تفاوتی وجود ندارد

الف) طراح کلاس ب)استفاده کننده از کلاس ج) کلاس وارث د) ب و ج

٤- ایجاد نمونه ای از کلاس را ..... و نمونه ایجاد شده از کلاس را ..... می گویند.

الف) تعریف شی ، کلاس ب)امتغیر نمونه ، نمونه سازی ج) **نمونه سازی** ، شی د) نمونه سازی ، کنترل

۵-..... به یک متغیر Public اطلاق می شود که در یک کلاس تعریف شده باشد

الف) فیلد ب) ویژگی ج) متد د)ایستا

٦ - کدام گزینه صحیح نمی باشد

الف) عضو سازنده در یک کلاس وظیفه مقدار دهی اولیه مقادیر مربوط به داده های کلاس را بر عهده دارد

ب) اعضای ثابت با کلید واژه Const تعریف می شوند

ج) از نظر کد خارج از کلاس ویژگی ها همانند فیلد ها هستند

د) اگر یک متد مجددا خودش را فراخوانی کند به آن متد همنام می گوییم

تمرین ۱ :

یک کلاس بنویسید که بتواند عمل جمع ، ضرب ، تفریق ، تقسیم و باقیمانده صحیح را بر روی دو عدد انجام دهد. کلاس دارای دو خاصیت به نام های x , y برای مشخص نمودن دو عدد و ۵ متد به نام های add و sub و mul و div و mod برای اعمال جمع ، ضرب ، تفریق ، تقسیم و باقیمانده صحیح باشد.

تمرین ۲:

یک کلاس برای انجام عملیات جمع ، تفریق و ضرب دو عدد n رقمی و محاسبه فاکتوریل یک عدد n رقمی بر پایه کار روی رشته های عددی ایجاد کنید

مثال :

```
class stars {

   int magnitude;        // Data member
   int starfunc(void);   // Member function

};
```

مثال : تعریف کلاس برای ذخیره سازی یک آرایه رشته ای ۱۰ تایی

```
#include <vcl.h>
#include <stdio.h>

class Famille
{
private:
AnsiString FNames[10];
AnsiString GetName(int Index);
void SetName(int, AnsiString);
public:
__property AnsiString Names[int Index] = {read=GetName,
write=SetName};
Famille(){}
~Famille(){}
};

AnsiString Famille::GetName(int i)
{
return FNames[i];
}

void Famille::SetName(int i,const AnsiString s)
{
FNames[i]=s;

}

int main()
{
Famille C;
C.Names[0]="Steve";   //calls Famille::SetName()

C.Names[1]="Amy";
C.Names[2]="Sarah";
C.Names[3]="Andrew";
for (int i = 0; i <= 3; i++)
{
//calls Famille::GetName()
puts(C.Names[i].c_str());
}
}
```

An object of a class with only public members and no constructors or base classes (typically a structure) can be initialized with an initializer list. If a class has a constructor, its objects must be either initialized or have a default constructor. The latter is used for objects not explicitly initialized.

Objects of classes with constructors can be initialized with an expression list in parentheses. This list is used as an argument list to the constructor. An alternative is to use an equal sign followed by a single value. The single value can be the same type as the first argument accepted by a constructor of that class, in which case either there are no additional arguments, or the remaining arguments have default values. It could also be an object of that class type. In the former case, the matching constructor is called to create the object. In the latter case, the copy constructor is called to initialize the object.

در مثال زیر بدنه تابع **x** برای وضوح و خوانایی مثال حذف شده است

```
class X

{
   int i;
public:
   X();
   X(int x);
   X(const X&);
};
void main()
{
   X one;        // default constructor invoked
   X two(1);     // constructor X::X(int) is used
   X three = 1;  // calls X::X(int)
   X four = one; // invokes X::X(const X&) for copy
   X five(two);  // calls X::X(const X&)

}
```

سازنده کلاس می تواند مقادیر اعضای کلاس را به دو طریق به آنها نسبت دهد

حالت اول : مقادیر را به صورت پارامتر دریافت و عمل مقدار دهی را در بدنه تابع سازنده انجام دهد

```
class X

{
   int a, b;
public:
   X(int i, int j) { a = i; b = j }

};
```

حالت دوم : با استفاده از لیستی از مقادیر که قبل از بدنه تابع قرار می گیرد

```
class X

{
نکته : متغیر c یک متغیر مرجع می باشد//
int a, b, &c;
public:
   X(int i, int j) : a(i), b(j), c(a) {}

};
```

The initializer list is the only place to initialize a reference variable.

In both cases, an initialization of X x(1, 2) assigns a value of 1 to x::a and 2 to x::b. The second method, the initializer list, provides a mechanism for passing values along to base class constructors.

Base class constructors must be declared as either public or protected to be called from a derived class. Note:

```
class base1

{
   int x;
public:
   base1(int i) { x = i; }
};

class base2
{
   int x;
public:
   base2(int i) : x(i) {}
};
class top : public base1, public base2
{
   int a, b;
public:
   top(int i, int j) : base1(i*5), base2(j+i), a(i) { b = j;}

};
```

With this class hierarchy, a declaration of top one(1, 2) would result in the initialization of base1 with the value 5 and base2 with the value 3. The methods of initialization can be intermixed.

As described previously, the base classes are initialized in declaration order. Then the members are initialized, also in declaration order, independent of the initialization list.

```
class X

{
  int a, b;
public:
  X(int i, j) :  a(i), b(a+j) {}

};
```

With this class, a declaration of X x(1,1) results in an assignment of 1 to x::a and 2 to x::b.

Base class constructors are called prior to the construction of any of the derived classes members. If the values of the derived class are changed, they will have no effect on the creation of the base class.

```
class base

{
   int x;
public:
   base(int i) : x(i) {}
};
class derived : base
{
   int a;
public:
   derived(int i) : a(i*10), base(a) { } // Watch out! Base will be
```

```
                                        // passed an uninitialized 'a'

};
```

With this class setup, a call of derived d(1) will not result in a value of 10 for the base class member x. The value passed to the base class constructor will be undefined.

When you want an initializer list in a non-inline constructor, don't place the list in the class definition. Instead, put it at the point at which the function is defined.

```
derived::derived(int i) : a(i)

{
  .
  .
  .

}
```

```
/* Compile with  bcc32 famille.cpp vcl.lib ole2w32.lib */
#include <vcl.h>
#include <stdio.h>

class Famille
{
private:
AnsiString FNames[10];
AnsiString GetName(int Index);
void SetName(int, AnsiString);
public:
__property AnsiString Names[int Index] = {read=GetName,
write=SetName};
Famille(){}
~Famille(){}
};

AnsiString Famille::GetName(int i)
{
return FNames[i];
}

void Famille::SetName(int i,const AnsiString s)
{
FNames[i]=s;

}

int main()
{
Famille C;
C.Names[0]="Steve";   //calls Famille::SetName()

C.Names[1]="Amy";
C.Names[2]="Sarah";
C.Names[3]="Andrew";
for (int i = 0; i <= 3; i++)
{
//calls Famille::GetName()
puts(C.Names[i].c_str());
```

```
}
}
```

مشتق گرفتن از کلاس (تعریف یک کلاس بر روی کلاس دیگری)

Base and derived class access

When you declare a derived class D, you list the base classes B1, B2, ... in a comma-delimited base-list:

```
class-key D : base-list { <member-list> }
```

D inherits all the members of these base classes. (Redefined base class members are inherited and can be accessed using scope overrides, if needed.) D can use only the public and protected members of its base classes. But, what will be the access attributes of the inherited members as viewed by D? D might want to use a public member from a base class, but make it private as far as outside functions are concerned. The solution is to use access specifiers in the base-list.

Since a base class can itself be a derived class, the access Note: attribute question is recursive: you backtrack until you reach the basemost of the base classes, those that do not inherit.

When declaring D, you can use the access specifier public, protected, or private in front of the classes in the base-list:

```
class D : public B1, private B2, ... {

  .
  .
  .

}
```

These modifiers do not alter the access attributes of base members as viewed by the base class, though they can alter the access attributes of base members as viewed by the derived class.

The default is private if D is a class declaration, and public if D is a struct declaration.

Unions cannot have base classes, and unions cannot be used as base Note: classes.

The derived class inherits access attributes from a base class as follows:

public base class: public members of the base class are public members of the derived class. protected members of the base class are protected members of the derived class. private members of the base class remain private to the base class.
protected base class: Both public and protected members of the base class are protected members of the derived class. private members of the base class remain private to the base class.

private base class: Both public and protected members of the base class are private members of the derived class. private members of the base class remain private to the base class.

Note that private members of a base class are always inaccessible to member functions of the derived class unless friend declarations are explicitly declared in the base class granting access. For example,

```
/* class X is derived from class A */

class X : A {               // default for class is private A
  .
  .
  .
}
/* class Y is derived (multiple inheritance) from B and C
   B defaults to private B */
class Y : B, public C {     // override default for C
  .
  .
  .
}
/* struct S is derived from D */
struct S : D {              // default for struct is public D
  .
  .
  .
}
/* struct T is derived (multiple inheritance) from D and E
   E defaults to public E */
struct T : private D, E {   // override default for D

                            // E is public by default
  .
  .
  .

}
```

The effect of access specifiers in the base list can be adjusted by using a qualified-name in the public or protected declarations of the derived class. For example:

```
class B {

   int a;               // private by default
public:
   int b, c;
   int Bfunc(void);
};
class X : private B {   // a, b, c, Bfunc are now private in X
   int d;               // private by default, NOTE: a is not
                        // accessible in X
public:
   B::c;                // c was private, now is public
   int e;
   int Xfunc(void);
};

int Efunc(X& x);        // external to B and X
```

The function Efunc() can use only the public names c, e, and Xfunc().

The function Xfunc() is in X, which is derived from private B, so it has access to

The “adjusted-to-public” c
The “private-to-X” members from B: b and Bfunc()
X’s own private and public members: d, e, and Xfunc()

However, Xfunc() cannot access the “private-to-B” member, a.

Friends of classes

A friend F of a class X is a function or class, although not a member function of X, with full access rights to the private and protected members of X. In all other respects, F is a normal function with respect to scope, declarations, and definitions.

Since F is not a member of X, it is not in the scope of X, and it cannot be called with the x.F and xptr->F selector operators (where x is an X object and xptr is a pointer to an X object).

If the specifier friend is used with a function declaration or definition within the class X, it becomes a friend of X.

friend functions defined within a class obey the same inline rules as member functions (see Inline functions). friend functions are not affected by their position within the class or by any access specifiers. For example:

```
class X {

   int i;                           // private to X
   friend void friend_func(X*, int);
/* friend_func is not private, even though it's declared in the private sec
tion */
public:
     void member_func(int);
};
/* definitions; note both functions access private int i */
void friend_func(X* xptr, int a) { xptr->i = a; }
void X::member_func(int a) { i = a; }

X xobj;
/* note difference in function calls */
friend_func(&xobj, 6);

xobj.member_func(6);
```

You can make all the functions of class Y into friends of class X with a single declaration:

```
class Y;                      // incomplete declaration

class X {
   friend Y;
   int i;
   void member_funcX();
};
class Y; {                    // complete the declaration
   void friend_X1(X&);
   void friend_X2(X*);
.
.
.

};
```

The functions declared in Y are friends of X, although they have no friend specifiers. They can access the private members of X, such as i and member_funcX.

It is also possible for an individual member function of class X to be a friend of class Y:

```
class X {

.
.
.
   void member_funcX();
}
class Y {
   int i;
   friend void X::member_funcX();
.
.
.

};
```

Class friendship is not transitive: X friend of Y and Y friend of Z does not imply X friend of Z. Friendship is not inherited.

ایجاد کلاس ، استفاده از یک کلاس در کلاس دیگر ، وراثت از یک کلاس

ایجاد کلاس :

ایجاد یک کلاس ساده به نام myDate با چند سازنده و دو متد به نام های GetDate و SetDate برای برگرداندن و مقداردهی متغیرهای داخلی کلاس

```
class myDate
{
   private:
      int M, D, Y;
   public:
      myDate() { M = 0; D = 0; Y = 0; }
      myDate(int year) { M = 0; D = 0; Y = year; }
      myDate(int year, int month) { M = month; D = 0; Y = year; }
      myDate(int year, int month,int day)
             { M = month; D = day; Y = year; }
      String GetDate()
      {
             return String(Y) + "-" + String(M) + "-" + String(D);
      }
      String SetDate(int year, int month,int day)
      {
             M = month; D = day; Y = year;
             return GetDate();
      }
};
```

- استفاده از کلاس myDate در کلاس Person
- نوشتن متد () GetPersonInfo برای برگرداندن مقادیر فیلد های کلاس Person

```
//-----------------------------------------------------------------------
class Person
{
   private:
      String fname, lname;
      String getName(){return lname;}
      void   setName(String value){lname=value;}
      String getFamily(){return fname;}
      void   setFamily (String value){fname=value;}
   public:
      myDate birthDate;
      Person(){
         Name ="";
         Family = "";
         birthDate =  myDate();
      }
      Person(String name,String family,myDate BirthDate){
         Name=name;
         Family = family;
         birthDate = BirthDate;
      }
      __property String Name={read=getName,write=setName};
      __property String Family={read=getFamily,write=setFamily};

      String GetPersonInfo(){
          return Name+" ,"+Family+","+birthDate.GetDate();
      }
};
//-----------------------------------------------------------------------
```

- وراثت
- ایجاد کلاس Employee بر روی کلاس پایه Person
- کلاس Employee تمامی ویژگی های عمومی کلاس پایه را به ارث می برد
- ایجاد متد GetEmployeeInfo () برای برگرداندن مقادیر فیلد های کلاس Employee

```
//-------------------------------------------------------------------
class Employee : public Person
{
    public:
       Employee() {}
       myDate Start;
       String GetEmployeeInfo()
       {
        return Person::GetPersonInfo()+","+Start.GetDate();
       }

};
//-------------------------------------------------------------------
```

نحوه استفاده از کلاس Employee

```
   Employee emp;
   emp.Name = "ali";
   emp.Family = "ahmadi";
   emp.birthDate =myDate(1360, 01, 01);
   emp.Start = myDate(1388, 01, 01);
   ShowMessage(emp.GetEmployeeInfo());
```

نمونه خروجی مثال فوق

تعریف و رونویسی متد های مجازی

تعریف کلاس Shape با یک متد مجازی به نام Area() برای برگرداندن مساحت

```
//-------------------------------------------------------------------
class Shape
{
   public:
      virtual double Area() { return 0; } //virtual method
};
//-------------------------------------------------------------------
```

تعریف کلاس Circle با رونویسی متد مجازی محیط برای محاسبه محیط دایره

```
class Circle : Shape
{
  private:
     double r;
     double get_r(void){return r;}
     void set_r(double x){r=x;}
  public:
```

```
    __property double Radius={read=get_r,write=set_r};
     double Area() { return r * r * 3.14159; }//override virtual method
};

//-----------------------------------------------------------------
```

نحوه استفاده از کلاس Circle

```
Circle c;
c.Radius=1;
double A= c.Area();
ShowMessage(String(A));
```

مثال :

کلاس های تعریف شده قبلی را طوری باز نویسی کنید که هر کلاس دارای یک متد مجازی به نام ToString () باشد و اعمال خواسته شده در زیر را انجام دهد

- متد ToString () در کلاس myDate برای بازگرداندن تاریخ به فرمت yy-mm-dd
- متد ToString () در کلاس Person برای بازگرداندن نام ، نام خانوادگی و تاریخ تولد شخص با جداکننده کاما
- متد ToString () در کلاس Employee برای بازگرداندن نام ، نام خانوادگی و تاریخ تولد و تاریخ استخدام کارمند با جداکننده کاما

ایجاد یک کلاس ساده به نام Date با چند سازنده و دو متد به نام های GetDate و SetDate برای برگرداندن و مقداردهی متغیرهای داخلی کلاس

ایجاد یک کلاس ساده به نام myDate با چند سازنده و دو متد به نام های GetDate و SetDate برای برگرداندن و مقداردهی متغیرهای داخلی کلاس

```
class myDate
{
   private:
      int M, D, Y;
   public:
      myDate() { M = 0; D = 0; Y = 0; }
      myDate(int year) { M = 0; D = 0; Y = year; }
      myDate(int year, int month) { M = month; D = 0; Y = year; }
      myDate(int year, int month,int day)
             { M = month; D = day; Y = year; }
      String GetDate()
      {
             return String(Y) + "-" + String(M) + "-" + String(D);
      }
      String SetDate(int year, int month,int day)
      {
             M = month; D = day; Y = year;
             return GetDate();
      }
      virtual String ToString(){return GetDate();}
};
```

- استفاده از کلاس myDate در کلاس Person
- نوشتن متد () GetPersonInfo برای برگرداندن مقادیر فیلد های کلاس Person
- ایجاد متد مجازی () ToString برای برگرداندن مقادیر فیلد های کلاس Person

//----------

```
class Person
{
   private:
      String fname, lname;
      String getName(){return lname;}
      void   setName(String value){lname=value;}
      String getFamily(){return fname;}
      void   setFamily (String value){fname=value;}
   public:
      myDate birthDate;
      Person(){
         Name ="";
         Family = "";
         birthDate =  myDate();
      }
      Person(String name,String family,myDate BirthDate){
         Name=name;
         Family = family;
         birthDate = BirthDate;
      }
      __property String Name={read=getName,write=setName};
      __property String Family={read=getFamily,write=setFamily};

      String GetPersonInfo(){
          return Name+" ,"+Family+","+birthDate.GetDate();
      }
      virtual String ToString(){
           return Name+" ,"+Family+","+ birthDate.ToString();
      }
};
//-----------------------------------------------------------------------
```

- وراثت
- ایجاد کلاس Employee بر روی کلاس پایه Person
- کلاس Employee تمامی ویژگی های عمومی کلاس پایه را به ارث می برد
- ایجاد متد GetEmployeeInfo () برای برگرداندن مقادیر فیلد های کلاس Person
- رونویسی متد مجازی ToString () برای برگرداندن مقادیر فیلد های کلاس Employee

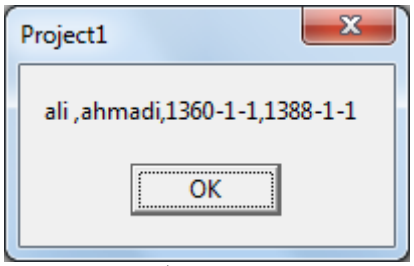
```
//-----------------------------------------------------------------------
class Employee : public Person
{
    public:
       Employee() {}
       myDate Start;
       String GetEmployeeInfo()
       {
        return Person::GetPersonInfo()+","+Start.GetDate();
       }
       virtual String ToString(){
         return Person::ToString()+","+Start. ToString();
       }

};
//-----------------------------------------------------------------------
```

نحوه استفاده از کلاس Employee

```
   Employee emp;
   emp.Name = "ali";
```

```
emp.Family = "ahmadi";
emp.birthDate =myDate(1360, 01, 01);
emp.Start = myDate(1388, 01, 01);
ShowMessage(emp.ToString());
```



نمونه خروجی مثال فوق

تمرین:

متدی به نام DiffDate به کلاس myDate اضافه کنید که تفاضل دو تاریخ شمسی را از هم محاسبه کند و حاصل را به صورت روز و ماه و سال برگرداند. (نکته : این متد باید در بخش public کلاس myDate اضافه شود تا قابل دسترسی باشد) جواب) با فرض اینکه تاریخ d2 بزرگتر از تاریخ d1 باشد می توان جواب را به شکل زیر نوشت

```
//-----------------------------------------------------------------------
    myDate DiffDate(myDate d2, myDate d1)
    {
        if (d1.D > d2.D)
        {
            d2.D += (d2.M <= 7) ? 31 : 30;
            d2.M--;
            if (d2.M == 0)
             {
                d2.Y--;
                d2.M = 12;
             }
        }
        if (d1.M > d2.M)
        {
            d2.Y--;
            d2.M += 12;
        }
        return myDate(d2.Y - d1.Y, d2.M - d1.M, d2.D - d1.D);
    }
//-----------------------------------------------------------------------
```

راه حل دوم استفاده از عملگر ؟ به جای دستور if

```
//-----------------------------------------------------------------------
    myDate DiffDate(myDate d2, myDate d1)
    {
        if (d1.D > d2.D)
        {
            d2.D += (d2.M <= 7) ? 31 : 30;
            d2.M--;
            d2.Y = d2.Y - ((d2.M == 0) ? 1 : 0);
            d2.M = (d2.M == 0) ? 12 : d2.M;
        }
        if (d1.M > d2.M)
        {
            d2.Y--;
            d2.M += 12;
        }
        return myDate(d2.Y - d1.Y, d2.M - d1.M, d2.D - d1.D);
```

```
        }
    //-----------------------------------------------------------------------
```

در پایان کار شکل کامل کلاس myDate به صورت زیر خواهد بود

```
//-----------------------------------------------------------------------
class myDate
{
   private:
       int M, D, Y;
   public:
       myDate() { M = 0; D = 0; Y = 0; }
      //-------------------------------------------------------
       myDate(int year) { M = 0; D = 0; Y = year; }
      //-------------------------------------------------------
       myDate(int year, int month) { M = month; D = 0; Y = year; }
      //-------------------------------------------------------
       myDate(int year, int month,int day)
          { M = month; D = day; Y = year; }
      //-------------------------------------------------------
       String GetDate()
       {
          return String(Y) + "-" + String(M) + "-" + String(D);
       }
      //-------------------------------------------------------
       String SetDate(int year, int month,int day)
       {
          M = month; D = day; Y = year;
          return GetDate();
       }
      //-------------------------------------------------------
       myDate DiffDate(myDate d2, myDate d1)
       {
          if (d1.D > d2.D)
          {
             d2.D += (d2.M <= 7) ? 31 : 30;
             d2.M--;
           if (d2.M == 0)
             {
                d2.Y--;
                d2.M = 12;
             }
          }
          if (d1.M > d2.M)
          {
             d2.Y--;
             d2.M += 12;
          }
          return myDate(d2.Y - d1.Y, d2.M - d1.M, d2.D - d1.D);
       }
      //-------------------------------------------------------
      virtual String ToString(){return GetDate();}
};
//-------------------------------------------------------
```

نحوه استفاده از کلاس myDate و تفاضل دو تاریخ به کمک متد DiffDate

```
    myDate d,d1=myDate(88,01,01),d2=myDate(88,04,01);
    d=d.DiffDate(d2,d1);
    ShowMessage(d.ToString());
```

تمرین :

:یک متد دیگر به نام DiffDate به کلاس myDate اضافه کنید که یک تاریخ و یک مقدار عددی (به عنوان روز) را دریافت و مقدار عددی را از تاریخ مورد نظر کسر و نتیجه را به عنوان یک داده از نوع کلاس myDate برگرداند

برای مثال اگر از تاریخ برابر ۲۲ آبان ۸۹ و روز برابر ۱۰ باشد خروجی باید برابر تاریخ ۱۲ آبان ۸۹ باشد

```
myDate DiffDate(myDate d, int day)
{
    …
}
```

برنامه ای بنویسید که با استفاده از کلاس Employee نام، نام خانوادگی،تاریخ تولد و تاریخ شروع به کار ۱۰ نفر از پرسنل را گرفته و سپس نمایش دهد.(راهنمایی : آرایه ای از نوع Employee تعریف کنید)

پرسش های پایان درس

۱- بعد از اجرای کامل دستورات زیر مقدار متغیر k چیست

```
int k=5;
for(int i=0;i<10;i++);
        for(int j=5;j<20;j++) j++;
```

الف) ۱۵۵ ب) ۲۰ ج) ٦ د) ٥

۲- بعد از اجرای کامل دستورات زیر مقدار متغیر k چیست

```
int k=5;
for(int i=0;i<10;i++);
        for(int j=5;j<20;j++) k++;
```

الف) ۱۵۵ ب) ۲۰ ج) ٦ د) ٥

۳- خروجی متد abc چیست

```
int abc(int x, int y)
{
  int t;
  while(x%y) {   t=y;  y =x%y;   x=t;   }
  return y;
}
```

الف) محاسبه ک.م .م بین دو عدد x , y ب) محاسبه ب.م .م بین دو عدد x , y

ج) محاسبه باقیمانده تقسیم x بر y د) محاسبه خارج قسمت x بر y

٤- نتیجه اجرای قطعه کد زیر چیست

```
Class MyClass{
 Public:
 int a(int x){return x%2==0;}
 int b(int x){if(a(x)) return x; else return 0; }
};
MyClass  x;
bool y=x.a(10);
```

الف) متغیر y دارای مقدار true می شود

ب) تعریف کلاس ایراد دارد

ج) دسترسی به متد a با توجه به تعریف کلاس امکان پذیر نیست و باید از متد b به جای آن استفاده شود

د) برنامه با خطا متوقف می شود

٥- یک کلاس طراحی کنید که :

- **دارای دو ویژگی برای تعیین دو عدد صحیح باشد**
- **متدی به نام BMM برای تعیین بزرگترین مقسوم علیه مشترک بین دو عدد موجود در ویژگی های فوق باشد**
- **متدی به نام KMM برای تعیین کوچکترین مضرب مشترک بین دو عدد موجود در ویژگی های فوق باشد**

٦ - متدی بنویسید که بتواند یک رشته عددی را دریافت و متتم ۹ آن را برگرداند.

۷- کد ملی یک شماره ۱۰ رقمی است که رقم سمت راست آن از روی سایر ارقام بدست می آید.بدین ترتیب که از سمت راست ارقام موجود در موقعیت۲ تا ۱۰ آن در موقعیت خودشان ضرب می شوند و سپس نتیجه ها با هم جمع شده و در نهایت عدد حاصل بر ۱۱ تقسیم می شود. اگر باقیمانده عددی کمتر از ۲ باشد در این صورت رقم سمت راست باید برابر باقیمانده باشد در غیر اینصورت رقم سمت راست باید برابر یازده منهای باقیمانده باشد. یک کلاس طراحی کنید که دارای متدی عمومی برای چک کردن اعتبار کد ملی باشد. (کد ملی را بگیرد و در صورت اعتبار مقدار true و در غیر اینصورت مقدار false را به عنوان نتیجه برگرداند)

مثال : فرض کنید می خواهیم اعتبار کد ملی ۰۷۴۸۸۲۰۲۱۳ را بررسی کنیم . در این صورت طبق جدول زیر ارقام ۲ تا ۱۰ کد را در موقعیت شان ضرب کرده و حاصل را با هم جمع می کنیم (۲+۶+۰+۱۰+۴۸+۵۶+۶۳+۰ برابر است با ۲۱۷، سپس ۲۱۷ را بر ۱۱ تقسیم می کنیم . باقیمانده این تقسیم برابر ۸ است که کمتر از ۲ نیست پس رقم سمت راست کد ملی باید ۸-۱۱ باشد که در مورد این کد رقم سمت راست ۳ می باشد. با این حساب این کد ملی به عنوان یک کد ملی معتبر مورد قبول است)

ارقام کد ملی	۰	۷	٤	۸	۸	۲	۰	۲	۱	۳
موقعیت رقم	۱۰	۹	۸	۷	٦	۵	٤	۳	۲	۱
حاصل ضرب	۰	٦۳	۳۲	۵٦	٤۸	۱۰	۰	٦	۲	

رونویسی عملگرها

شکل کلی رونویسی عملگر ها به صورت زیر می باشد

```
operator <operator symbol>( <parameters> )
{
   <statements>;
}
```

مثال : تعریف عملگر منها برای تفاضل دو تاریخ در کلاس mydate

```
myDate operator- (myDate d1)
{
      return DiffDate(myDate(Y,M,D),d1);
}
```

نحوه استفاده از عملگر فوق

```
myDate d,d1=myDate(88,01,01),d2=myDate(88,04,01);
d=d2-d1;
ShowMessage(d.ToString());
```

مثال : تعریف عملگر منها برای کم کردن روز از یک تاریخ و بدست آوردن تاریخ معادل در کلاس mydate

```
//-----------------------------------------------------------------------
 myDate operator-(int count)
 {
  while(count>0){
    D--;
    count--;
    if(D==0){
       M--;
       if(M==0){
          Y--;
         M=12;
       }
```

```
      if(M>6) D=30; else D=31;
     }//if
   }//while
  return myDate(Y,M,D);
 }
 //-----------------------------------------------------------------------
```

نحوه استفاده از عملگر فوق

```
myDate d,d1=myDate(88,04,01);
d=d1-61;
ShowMessage(d.ToString());
```

تمرین ۱ : عمگرهای ++ و -- را برای کلاس myDate تعریف کنید

تمرین ۲ : یک کلاس برای کار با اعداد کسری تعریف کنید که دارای عملگرهای + ، - ، * و / برای جمع ، تفریق ، ضرب و تقسیم اعداد کسری باشد.

رشته ها :

به دنباله ای از کاراکترها اطلاق می شود

در C++ Builder سه نوع رشته قابل تعریف است که عبارتند از :ShortString ، AnsiString و WideString

نوع	حداکثر طول	حافظه مود نیاز	کاربرد
ShortString	255 characters	2 to 256 bytes	سازگاری با نگارش های قدیمی backward compatibility
AnsiString	~2^31 characters	4 bytes to 2GB	کاراکترهای ۸ بیتی 8-bit (ANSI) characters
WideString	~2^30 characters	4 bytes to 2GB	کاراکترهای یونی کد Unicode characters; multi-user servers and multi-language applications

به نوع AnsiString در اغلب مواقع رشته بلند (Long String) نیز اطلاق می شود

بررسی کلاس AnsiString

بررسی متد های مهم

متد AnsiString

این متد یک نمونه/ شی از نوع کلاس AnsiString ایجاد می کند. پارامتر ورودی می تواند بر حسب مورد متفاوت باشد

فرم کلی این متد به شکل های زیر است

```
__fastcall AnsiString();
__fastcall AnsiString(const char* src);
__fastcall AnsiString(const AnsiString& src);
__fastcall AnsiString(const char* src, unsigned int len);
__fastcall AnsiString(const wchar_t* src);
__fastcall AnsiString(int src);
__fastcall AnsiString(double src);

__fastcall AnsiString(char src);
__fastcall AnsiString(short);
__fastcall AnsiString(unsigned short);
__fastcall AnsiString(unsigned int);
__fastcall AnsiString(long);
__fastcall AnsiString(unsigned long);
__fastcall AnsiString(__int64);
__fastcall AnsiString(unsigned __int64);
__fastcall AnsiString(const WideString &src);
```

مثال : تبدیل عدد به رشته :

```
int x=1234567;
AnsiString s=AnsiString(x);
```

متد AnsiCompare

این متد رشته فعلی را با یک رشته دیگر با هم مقایسه می کند و در صورتی که رشته ها با هم برابر باشند مقدار صفر و در صورتی که رشته رشته فعلی کوچکتر از رشته ورودی باشد مقدار کمتر ار صفر و در غیر این صورت (رشته فعلی بزرگتر از رشته ورودی) یک مقدار مثبت به عنوان نتیجه برمی کرداند.. (در این متد عمل حساس به متن است)

فرم کلی این متد به شکل زیر است

```
int __fastcall AnsiCompare(const AnsiString& rhs) const;
```

مثال :

```
AnsiString s1="ali arash";
int p=s1.AnsiCompare("arash");
```

در اینجا چون رشته ali arash از نظر موقعیت در دیکشنری بالاتر از arash قرار می گیرد نتیجه ۱- به عنوان خروجی برگشت داده می شود

متد AnsiCompareIC

این متد همانند متد AnsiCompare عمل مقایسه بین رشته فعلی و یک رشته دیگر را انجام می دهد با این تفاوت که در اینجا عمل مقایسه حساس به متن نیست

فرم کلی این متد به شکل زیر است

int __fastcall AnsiCompareIC (**const** AnsiString& rhs) **const**;

متد AnsiPos

این متد موقعیت شروع یک رشته را داخل رشته فعلی برمی گرداند. (شماره موقعیت ها از ۱ شروع می شود یعنی ۱ برای موقعیت اول و۲ برای موقعیت دومو ...). در صورتی که رشته مورد نظر در داخل رشته اصلی پیدا نشود مقدار صفر برکشت داده می شود.

فرم کلی این متد به شکل زیر است

int __fastcall AnsiPos(**const** AnsiString& subStr) **const**;

مثال :

AnsiString s1="ali arash";
int p=s1. AnsiPos("arash");

در اینجا چون زیر رشته arash از موقعیت شماره ۵ در رشته ali arash شروع می شود مقدار ۵ به عنوان موقعیت شروع برگشت داده می شود

متد Pos

این متد موقعیت شروع یک رشته را داخل رشته فعلی برمی گرداند. (شماره موقعیت ها از ۱ شروع می شود یعنی ۱ برای موقعیت اول و۲ برای موقعیت دومو ...). در صورتی که رشته مورد نظر در داخل رشته اصلی پیدا نشود مقدار صفر برکشت داده می شود.

فرم کلی این متد به شکل زیر است

int __fastcall Pos(**const** AnsiString& subStr) **const**;

مثال :

```
AnsiString s1="ali arash";
int p=s1.Pos("arash");
```

در اینجا چون زیر رشته arash از موقعیت شماره ۵ در رشته ali arash شروع می شود مقدار ۵ به عنوان موقعیت شروع برگشت داده می شود

متد c_str

این متد رشته موجود را تبدیل به یک رشته منتهی به کاراکتر null کرده و به عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

char* __fastcall c_str() **const;**

مثال :

```
AnsiString s1="ali arash";
char *x=s1.c_str();
```

متد Length

این متد طول رشته موجود را عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

int __fastcall Length() **const;**

مثال : بدست آوردن طول رشته

```
AnsiString s1="ali arash";
int l =s1.Length();
```

متد SubString

این متد یک زیر مجموعه از رشته را جدا کرده و عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

AnsiString **__fastcall** SubString(**int** index, int count) **const;**

مثال : جداکردن قسمتی از رشته (جدا کردن کلمه name از رشته in the name of GOD)

```
AnsiString s1="in the name of GOD";
AnsiString s1 =s1.SubString(8,4) ;
```

متد Trim

این متد کلیه فاصله های خالی موجود در سمت چپ و راست رشته را حذف کرده و رشته جدیدی را به عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

```
AnsiString __fastcall Trim() const;
```

مثال : حذف فاصله های خالی موجود در سمت چپ و راست رشته

```
AnsiString s1="    in the name of GOD     ";
s1 =s1.Trim();
```

متد TrimLeft

این متد کلیه فاصله های خالی موجود در سمت چپ رشته را حذف کرده و رشته جدیدی را به عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

```
AnsiString __fastcall TrimLeft() const;
```

مثال : حذف فاصله های خالی موجود در سمت چپ رشته

```
AnsiString s1="    in the name of GOD     ";
s1 =s1.TrimLeft();
```

متد TrimRight

این متد کلیه فاصله های خالی موجود در سمت راست رشته را حذف کرده و رشته جدیدی را به عنوان خروجی برمی گرداند

فرم کلی این متد به شکل های زیر است

```
AnsiString __fastcall Trim() const;
```

مثال : حذف فاصله های خالی موجود در سمت راست رشته

```
AnsiString s1="    in the name of GOD     ";
s1 =s1.TrimRight();
```

متد WideChar

این متد کاراکترهای رشته موجود را تبدیل به یونی کد کرده و نتیجه را در یک بافر قرار می دهد

فرم کلی این متد به شکل های زیر است

```
wchar_t* __fastcall WideChar(wchar_t* dest, int destSize) const;
```

مثال :

```
AnsiString s1="آموزش زبان سی";
int n=s1.Length();
  wchar_t *x=new wchar_t[n];
  x=s1.WideChar(x,n);
```

نکته : امکان کار با رشته ها به صورت یک آرایه یک سطری هم وجود دارد برای این منظور از می توانیم از اپراتور [] استفاده کنیم . برای مثال اگر در مثال قبل بخواهیم به عنصر i ام رشته s1 دسترسی داشته باشیم از s1[i] استفاده کنیم

مثال : مقدار دهی یک آرایه کاراکتری با کاراکترهای یک رشته AnsiString

```
char  y[5];
for (int i=0;i<5;i++)
```

y[i]=s1[i];

کلاس AnsiString هم چنین دارای اپراتور های مقایسه ای > ، < ، =< ، => ، =! ، =+ ، == ، = ، + می باشد . که کاربرد هر کدام در جدول زیر آورده شده است

اپراتور	کاربرد	مثال
<	کوچکتر	S1<S2
>	بزرگتر	S1>S2
<=	کوچکتر با مساوی	S1<=S2
>=	بزرگتر با مساوی	S1>=S2
!=	نا مساوی	S1!=S2
==	برابری	S1==S2
=	انتساب	S1=S2;
+=	جمع رشته ای	S1+=S2;
+	جمع رشته ای	S1=S2+S2;

متد ToInt

این متد رشته عددی موجود را به یک عدد صحیح تبدیل می کند

فرم کلی این متد به شکل های زیر است

int __fastcall ToInt() **const;**

مثال :

```
AnsiString s1="1123456";
int n =s1.ToInt();
```

متد ToIntDef

این متد رشته عددی موجود را به یک عدد صحیح تبدیل می کند. در صورتی که رشته حاوی مقدار معتبری نباشد . مقدار پیش فرض که به عنوان آرگومان متد ارسال شده برگشت داده می شود

فرم کلی این متد به شکل های زیر است

int __fastcall ToIntDef**(int** defaultValue**) const;**

مثال :

```
AnsiString s1="56";
int n =s1. ToIntDef (0);
AnsiString s2="56.5";
int m =s2. ToIntDef (0);
```

نکته : با توجه به اینکه مقدار 56.5 یک مقدار معتبر نیست مقدار m برابر صفر می باشد.

متد ToDouble

این متد رشته عددی موجود را به یک عدد دابل تبدیل می کند

فرم کلی این متد به شکل های زیر است

double __fastcall ToDouble() **const**;

مثال :

```
AnsiString s1="456.657";
double n =s1.ToDouble();
```

بررسی کلاس TString

بررسی ویژگی های مهم

ویژگی CommaText

این ویژگی رشته های موجود در شی TString را به فرمت SDF(system dada format) لیست می کند.

فرم کلی این ویژگی به شکل زیر است

__property AnsiString CommaText = {**read**=GetCommaText, **write**=SetCommaText};

مثال : فرض کنید شی TString دارای رشته های زیر باشد.

Stri,ng 1
Stri"ng 2
String 3
String4

در این صورت خروجی CommaText به صورت زیر خواهد بود. در خروجی ارائه شده توسط CommaText در صورتی که رشته مورد نظر حاوی کاما ، فاصله خالی و یا علامت دابل کوتیشن باشد در داخل علامت دابل کوتیشن محصور می شود. و سپس هر کدام از رشته ها توسط علامت کاما از یکدیگر جدا می شوند.

"Stri,ng 1","Stri""ng 2","String 3",String4

در موقع مقدار دهی CommaText با یک رشته به فرمت SDF ستونهای داده ای با علامت کاما / فاصله خالی مشخص می شوند . بنابراین اگر رشته

"Stri,ng 1", "Stri""ng 2" , String 3,String4

را به CommaText نسبت دهیم در این صورت شی TString به صورت زیر خواهد بود

Stri,ng 1
Stri"ng 2
String
3
String4

نکته : افزودن یک کاما به دنباله رشته باعث افزودن یک رشته خالی به لیست می شود. برای مثال اگر رشته SDF

"String1, String 2, String 3,"

را به CommaText نسبت دهیم در این صورت شی TString به صورت زیر خواهد بود

String1
String2
String3
<Blank>

ویژگی Count:

این ویژگی تعداد رشته های موجود در لیست را برمی گرداند. این ویژگی یک ویژگی فقط خواندنی است

فرم کلی آن به شکل زیر است

__property int Count = {**read**=GetCount, nodefault};

ویژگی DelimitedText

این ویژگی یک رشته شامل تمامی رشته های موجود در شی TStringبر می گرداند که در آن رشته های موجود در لیست توسط یک کاراکتر جدا کننده از هم متمایز شده اند.

فرم کلی آن به شکل زیر است

```
__property AnsiString DelimitedText = {read=GetDelimitedText,
write=SetDelimitedText};
```

در صورتی که کاراکتر جداکننده کاما و کارکتر محصور کننده دابل کوتیشن انتخاب شود ، عملکرد این ویژگی با عملکرد CommaText یکی است. کاراکتر جداکننده توسط ویژگی Delimiter و کاراکتر محصور کننده توسط ویژگی QuoteChar مشخص می شود

ویژگی Delimiter

این ویژگی کاراکتر جداکننده مورد استفاده توسط ویژگی DelimitedText را تعیین می کند

فرم کلی آن به شکل زیر است

```
__property char Delimiter = {read=GetDelimiter, write=SetDelimiter};
```

ویژگی QuoteChar

این ویژگی کاراکتر محصور کننده رشته را که مورد استفاده توسط ویژگی DelimitedText می باشد تعیین می کند

فرم کلی آن به شکل زیر است

```
__property char QuoteChar = {read=GetQuoteChar, write=SetQuoteChar};
```

ویژگی Text

این ویژگی یک رشته شامل تمامی رشته های موجود در شی Tstringبر می گرداند که در آن رشته های موجود در لیست توسط یک کاراکتر های CRو LF از هم متمایز شده اند.

نکته: رشته هایی که توسط کاراکتر های CRو LF از هم جدا شده اند نسبت به خروجی های مربوط به CommaText و DelimitedText واضح تر بوده و دارای ابهام کمتری می باشند

فرم کلی آن به شکل زیر است

```
__property AnsiString Text = {read=GetTextStr, write=SetTextStr};
```

ویژگی Strings:

به کمک این ویژگی می توان به رشته های ذخیره شده در لیست دسترسی داشت. Index موقعیت رشته مورد نظر در لیست را مشخص می کند که صفر برای موقعیت اول و ۱ برای موقعیت دوم و ...

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Strings[int Index] = {read=Get, write=Put};
```

ویژگی های Names و Values

در اکثر مواقع خصوصا در فایل های ini عموما رشته ها به صورت Name=Value می باشند. کلاس TString دارای دو ویژگی برای کار با رشته های دارای قالب Name=Value می باشد که عبارت است از Names و Value که اولی نام و دومی مقدار مرتبط با یک نام را برمی گرداند. در زیر چند نمونه رشته با این قالب آورده شده است

```
DisplayGrid=1
SnapToGrid=1
GridSizeX=8
GridSizeY=8
```

نکته : در قالب فوق فرض بر این است که قبل و بعد از علامت مساوی فاصله خالی وجود ندارد. مگر اینکه فاصله مورد نظر مربوط به مقدار/ نام باشد

ویژگی Names:

این ویژگی نام مربوط به رشته های با قالب Name=Value را بر می کرداند فرم کلی این ویژگی به شکل زیر است

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Names[int Index] = {read=GetName;}
```

نکته۱ : مقدار Index برای عنصر اول لیست صفر و برای دومی ۱ و .... می باشد.

نکته۲ : چنانچه رشته موجود در موقعیت Index به صورت قالب Name=Value نباشد مقدار رشته خالی برگشت داده می شود.

ویژگی Values :

این ویژگی مقدار مربوط به رشته های با قالب Name=Value را بر می کرداند

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Values[AnsiString Name] = {read=GetValue, write=SetValue};
```

نکته : چنانچه رشته معادل با نام مورد نظر در لیست پیدا نشود مقدار رشته خالی برگشت داده می شود.

بررسی متدهای مهم

متد Add

این متد یک رشته را دریافت و به انتهای لیست رشته ای اضافه می کند و شماره موقعیت رشته اضافه شده در لیست را به عنوان خروجی برمی کرداند

فرم کلی این متد به شکل زیر است

```
virtual int __fastcall Add(const AnsiString S);
```

متد AddStrings

این متد کلیه رشته های موجود در یک شی TString را به انتهای لیست رشته ای اضافه می کند و شماره موقعیت رشته اضافه شده در لیست را به عنوان خروجی برمی کرداند

فرم کلی این متد به شکل زیر است

virtual void __fastcall AddStrings(TStrings* Strings);

متد Append

عملکرد این متد شبیه متد Add است با این تفاوت که موقعیت رشته اضافه شده به انتهای لیست را برنمی گرداند

فرم کلی این متد به شکل زیر است

void __fastcall Append(**const** AnsiString S);

متد Clear

این متد رشته های موجود در لیست را از حافظه لیست حذف و لیست را خالی می کند.

فرم کلی این متد به شکل زیر است

virtual void __fastcall Clear(**void**) = 0;

متد CompareStrings

این متد دو رشته را با هم مقایسه می کند و در صورتی که رشته ها با هم برابر باشند مقدار صفر و در صورتی که رشته S1<S2 باشد مقدار کمتر ار صفر و در غیر این صورت (S1>S2) یک مقدار مثبت به عنوان نتیجه برمی کرداند. از این متد TString به صورت داخلی برای یافتن رشته مورد نظر و بر گرداندن نام و یا موقعیت رشته استفاده می کند. (در این متد عمل حساس به متن نیست)

فرم کلی این متد به شکل زیر است

virtual int __fastcall CompareStrings(**const** AnsiString S1, **const** AnsiString S2);

متد Delete

این متد رشته موجود در موقعیت خاصی از لیست را حذف می کند. . (موقعیت ها عبارتند از صفر برای رشته اول و ۱ برای رشته دوم . ..)

فرم کلی این متد به شکل زیر است

virtual void __fastcall Delete(**int** Index) = 0;

متد Exchange

این متد رشته های موجود در دو موقعیت خاص از لیست را با هم جا به جا می کند. . (موقعیت ها عبارتند از صفر برای رشته اول و ۱ برای رشته دوم . ..)

فرم کلی این متد به شکل زیر است

virtual void __fastcall Exchange(**int** Index1, **int** Index2);

متد ExtractName

این متد قسمت مربوط به نام را در رشته ورودی که به فرمت Name=Value باشد را جدا کرده و به عنوان خروجی برمی گرداند، چنانچه رشته ورودی به فرمت فوق نباشد این متد یک رشته خالی را به عنوان خروجی برمی گرداند

فرم کلی این متد به شکل زیر است

AnsiString **__fastcall** ExtractName(**const** AnsiString S);

متد Equals

این متد قسمت محتویات رشته های موجود در یک شی TStrings را با یک شی دیگر از همین نوع مقایسه می کند و در صورتی که هر دو معادل هم باشند. نتیجه true و در غیر این صورت مقدار false را به عنوان نتیجه مقایسه برمی گرداند

فرم کلی این متد به شکل زیر است

bool __fastcall Equals(TStrings* Strings);

متد GetText

این متد یک بافر کاراکتری از حافظه را اختصاص داده و تمامی رشته های موجود در لیست را به در یک بافر کاراکتری ذخیره و آدرس مربوط به آن را به عنوان خروجی بر می گرداند.

فرم کلی این متد به شکل زیر است

virtual char * __fastcall GetText(**void**);

متد IndexOf

این متد موقعیت رشته ورودی را در لیست رشته ای پیدا کرده و برمی گرداند. در صورتی که رشته ورودی در لیست پیدا نشوند مقدار ۱- را به عنوان خروجی برمی گرداند.اگر رشته مورد جستجو در لیست بیش از یک بار وجود داشته باشد. این متد فقط موقعیت اولین تکرار آن را برمی گرداند

فرم کلی این متد به شکل زیر است

virtual int __fastcall IndexOf(**const** AnsiString S);

متد IndexOfName

این متد موقعیت نام مورد نظر را در لیست رشته ای که به صورت Name=Value پیدا کرده و برمی گرداند. در صورتی که نام مورد نظر در لیست پیدا نشوند مقدار ۱- را به عنوان خروجی برمی گرداند.اگر نام مورد جستجو در لیست بیش از یک بار وجود داشته باشد. این متد فقط موقعیت اولین تکرار آن را برمی گرداند

فرم کلی این متد به شکل زیر است

virtual int __fastcall IndexOfName(**const** AnsiString Name);

مثال : بروز رسانی یک لیست رشته ای از روی یک لیست دیگر

```
void MergeStrings(TStrings *Dest, TStrings *Source)

{
  for (int i = 0; i < Source->Count; i++)
  {
    if (Source->Strings[i].Pos("=") > 1)
    {
      int DI = Dest->IndexOfName(Source->Names[i]);
      if (DI > -1)
        Dest->Values[DI] = Source->Values[i];
    }
  }
}
```

متد Insert

این متد یک رشته و یک موقعیت را دریافت و رشته مورد نظر را در موقعیت مشخص شده لیست رشته ای درج می کند .
شماره موقعیت ها عبارتند از از صفر برای ابتدای لیست ، ۱ برای موقعیت دوم و ...
فرم کلی این متد به شکل زیر است

void __fastcall Insert(**int** Index, **const** AnsiString S) = 0;

متد LoadFromFile

این متد لیست را توسط خطوط متن موجود در یک فایل متنی پر می کند . این متد از متد Add برای افزودن متن موجود در فایل به لیست استفاده می کند
فرم کلی این متد به شکل زیر است

virtual void __fastcall LoadFromFile(**const** AnsiString FileName);

متد SaveToFile

این متد رشته های موجود در لیست را در یک فایل متنی ذخیره می کند . هر رشته در یک خط جداگانه در فایل نوشته می شود.
فرم کلی این متد به شکل زیر است

virtual void __fastcall SaveToFile(const AnsiString FileName);

متد SetText

این متد رشته های موجود در لیست را از یک بافر کاراکتری پر می کند . این متد از کاراکترهای CR و LF به عنوان جداکننده ستون های رشته ای استفاده می کند
فرم کلی این متد به شکل زیر است

virtual void __fastcall SetText(char * Text);

متد TStrings

این متد سازنده کلاس TStrings بوده و یک شی از نوع TString را ایجاد کرده و آدرس آنرا به عنوان خروجی برمی گرداند.
فرم کلی این متد به شکل زیر است

__fastcall TStrings(void);

نکته : نمی توان از این متد برای ایجاد یک Instance از شی TStrings استفاده نمود. به جای آن از TStringList استفاده نمایید

مثال:

TStringList *s=new TStringList;

بررسی کلاس TStringList

بررسی ویژگی های مهم

ویژگی CaseSensitive :

این ویژگی تعیین می کند آیا عملیات قابل انجام بر روی رشته به صورت حساس به متن باشد/ خیر (بین حروف کوچک و بزرگ تفاوت قایل شود / خیر)

فرم کلی این ویژگی به شکل زیر است

```
__property bool CaseSensitive = {read=FCaseSensitive, write=SetCaseSensitive,
nodefault};
```

از CaseSensitive برای تعیین اینکه آیا عملیات قابل انجام بر روی رشته به صورت حساس به متن هست / نه استفاده کنید.

جنانچه می خواهید عملیات قابل انجام بر روی رشته ها حساس به متن باشد. مقدار CaseSensitive را به true تنظیم کنید. در غیر ابنصورت مقدار این ویژگی را به false تغییر دهید.

ویژگی Duplicates :

این ویژگی تعیین می کند که آیا رشته های موجود در یک لیست مرتب می توانند تکراری باشند/ خیر

فرم کلی این ویژگی به شکل زیر است

```
__property TDuplicates Duplicates = {read=FDuplicates, write=FDuplicates, nodefault};
```

مقدار این ویژگی با توجه به جدول زیر تعیین می شود

مقدار	مفهوم
dupIgnore	از درج موارد تکراری در لیست صرفنظر می کند
dupError	در صورت درج رشته تکراری خطای/ استثنای EStringListError را ایجاد می کند که می توان به کمک try - catch آن را کنترل کرد
dupAccept	رشته های موجود در لیست می توانند تکراری باشند

نکته : مقدار این ویژگی را قبل از افزودن رشته به لیست مقدار دهی کنید . ویژگی dupIgnore و یا dupError کنترلی بر روی موارد تکراری که قبلا به لیست اضافه شده باشد

ویژگی Count:

این ویژگی تعداد رشته های موجود در لیست را برمی گرداند. این ویژگی یک ویژگی فقط خواندنی است

فرم کلی آن به شکل زیر است

```
__property int Count = {read=GetCount, nodefault};
```

ویژگی Sorted:

این ویژگی تعیین می کند که آیا رشته های موجود در لیست به صورت خودکار مرتب شوند / نه

```
__property bool Sorted = {read=FSorted, write=SetSorted, nodefault};
```

در صورتی که مقدار Sorted برابر true باشد . رشته های موجود در لیست به صورت خودکار مرتب می شوند.

ویژگی Strings:

به کمک این ویژگی می توان به رشته های ذخیره شده در لیست دسترسی داشت. Index موقعیت رشته مورد نظر در لیست را مشخص می کند که صفر برای موقعیت اول و ۱ برای موقعیت دوم و ...

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Strings[int Index] = {read=Get, write=Put};
```

ویژگی های Names و Values

در اکثر مواقع خصوصا در فایل های ini عموما رشته ها به صورت Name=Value می باشند. کلاس TString دارای دو ویژگی برای کار با رشته های دارای قالب Name=Value می باشد که عبارت است از Names و Value که اولی نام و دومی مقدار مرتبط با یک نام را برمی گرداند. در زیر چند نمونه رشته با این قالب آورده شده است

```
DisplayGrid=1
SnapToGrid=1
GridSizeX=8
GridSizeY=8
```

نکته : در قالب فوق فرض بر این است که قبل و بعد از علامت مساوی فاصله خالی وجود ندارد. مگر اینکه فاصله مورد نظر مربوط به مقدار/ نام باشد

ویژگی Names:

این ویژگی نام مربوط به رشته های با قالب Name=Value را بر می کرداند فرم کلی این ویژگی به شکل زیر است

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Names[int Index] = {read=GetName;}
```

نکته۱ : مقدار Index برای عنصر اول لیست صفر و برای دومی ۱ و .... می باشد.

نکته۲ : چنانچه رشته موجود در موقعیت Index به صورت قالب Name=Value نباشد مقدار رشته خالی برگشت داده می شود.

ویژگی Values :

این ویژگی مقدار مربوط به رشته های با قالب Name=Value را بر می کرداند

فرم کلی این ویژگی به شکل زیر است

```
__property AnsiString Values[AnsiString Name] = {read=GetValue, write=SetValue};
```

نکته : چنانچه رشته معادل با نام مورد نظر در لیست پیدا نشود مقدار رشته خالی برگشت داده می شود.

ویژگی های زیر از کلاس TStrings به TStringList به ارث رسیده است

CommaText , DelimitedText ,Delimiter ,Names ,QuoteChar ,StringsAdapter ,Text ,Values

بررسی متد های مهم

متد های زیر از کلاس TStrings به TStringList به ارث رسیده است

AddStrings , Append , Equals, GetText, IndexOfName
IndexOfObject, LoadFromFile , LoadFromStream , Move
SaveToFile , SaveToStream , SetText,

کامپوننت ها و کنترل ها

تعریف کامپوننت :

کامپوننت شی است که در زمان طراحی کاربر می تواند ویژگی های آن را تغییر دهد .

تعریف کنترل:

کنترل شی است که در زمان اجرا بر روی صفحه نمایش ظاهر شده و کاربر می تواد با آن تعامل داشته باشد.

رویداد ها

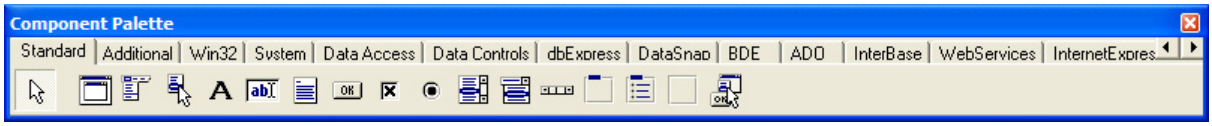
رویداد های قابل بررسی در کنترل ها عموما شامل رویدادهای موس که عبارتند از کلیک ، دابل کلیک ، فشردن موس ، رها کردن موس ، حرکت دادن موس و ... و نیز رویدادهای صفحه کلید شامل فشردن کلید ، رها کردن کلید و ... ، و نیز رویدادهای مربوط به دریافت و از دست دادن کنترل می باشند . هر کنترل دارای تعدای رویداد است و طراح و برامه نویس وظایف برنامه را بر حسب رویدادهای محیط برنامه نویسی و مشخص می کند .و مثلا برای یک دکمه تعیین می کند که اگر بر روی دکمه کلیک شود جه عملی باید انجام شود. یا مثلا وقتی که یک فرم فعال شود چه کاری باید انجام شود . و ...

بنابراین به جای اینکه رویداده های کنترل ها را بررسی کنیم باید بتوانیم بر روی رویدادها برنامه ریزی انجام دهیم که این موضوع به کمک متدها و ویژگی های مریوط به اشیاء ، کنترل ها و کامپوننت ها میسر می شود

کنترل فرم :

فرم ها یکی از پر کاربرد ترین کنترل هستند. فرم ها پنجره هایی هستند که سایر کنترل ها را در برمی گیرند و باعث سازماندهی سایر کنترل ها می شوند. به اصلی ترین فرم یک نرم افزار که سایر فرم ها در داخل آن قرار می گیرند (ظاهر می شوند) فرم اصلی می گویند

در سی بیلدر به منظور سهولت استفاده از کنترل ها آن ها را به شکل دسته بندی شده در یک Component Palette قرار داده اند . این دسته بندی کمک می کند تا برنامه نویس به راحتی بتواند کنترل مورد نظرش را از میان کنترل های موجود پیدا کند .



تصویر Component Palet

همان طوری که در تصویر Component Palette مشاهده می کنید . تعداد زیادی کنترل / کامپوننت وجود دارد .در این درس ما تعدادی از این کنترل ها را مورد بررسی قرار می دهیم.

بررسی کنترل فرم :

بررسی ویژگی های مهم

نکته : تمامی این ویژگی ها از طریق پنجره Object Inspector قابل تنظیم است.

ویژگی Caption

این ویژگی عنوان فرم را مشخص می کند . فرم کلی این ویژگی به شکل زیر می باشد

__property WideString Caption = {**read**=GetText, **write**=SetText};

ویژگی Active

از این ویژگی برای تعیین اینکه آیا فرم مورد نظر ما فرم فعال است / نه استفاده می کنیم. فرم فعال فرمی است که تمامی ورودی های صفحه کلید به آن ارسال می شود. فرم کلی این ویژگی به شکل زیر می باشد

__property bool Active = **{read**=FActive**, nodefault};**

ویژگی BorderIcons

این ویژگی آیکون هایی / دکمه هایی که بر روی عنوان فرم ظاهر می شود را تعیین می کند .. فرم کلی این ویژگی به شکل زیر می باشد

enum TBorderIcon = { biSystemMenu, biMinimize, biMaximize, biHelp };
typedef Set<TBorderIcon, biSystemMenu, biHelp> TBorderIcons;
__property TBorderIcons BorderIcons = {**read**=FBorderIcons, **write**=SetBorderIcons, **stored**=IsForm, default=7}

مقدار این ویژگی می تواند ترکیبی از مقادیر مجموعه شمارشی BorderIcons باشد که مفهوم هر کدام در جدول زیر آورده شده است.

مقدار	مفهوم
biSystemMen	نمایش منوی کنترلی/ سیستم بر روی نوار عنوان فرم
biMinimize	نمایش دکمه minimize بر روی نوار عنوان فرم
biMaximize	نمایش دکمه maximize بر روی نوار عنوان فرم

biHelp	نمایش دکمه راهنما بر روی نوار عنوان فرم

مثال : نمایش دکمه های حداکثر ، حداقل و راهنما بر روی نوار عنوان فرم

```
TBorderIcons tempBI ;
tempBI.Clear();
tempBI << biMaximize<< biMinimize<< biHelp;
this->BorderIcons = tempBI;
```

مثال : حذف دکمه های حداکثر ، حداقل از روی نوار عنوان فرم

```
TBorderIcons tempBI= this->BorderIcons;
tempBI >> biMaximize>> biMinimize;
this->BorderIcons = tempBI;
```

ویژگی BorderStyle

این ویژگی این ویژگی شکل ظاهری و رفتار فرم را مشخص می کند . فرم کلی این ویژگی به شکل زیر می باشد

__property TFormBorderStyle BorderStyle = {read=FBorderStyle, write=SetBorderStyle, stored=IsForm, default=2};

مقدار این ویژگی می تواند ترکیبی از مقادیر مجموعه شمارشی BorderStyle باشد که مفهوم هر کدام در جدول زیر آورده شده است.

مقدار	مفهوم
bsDialog	اندازه فرم قابل تغییر نیست ؛حاشیه جعبه گفتگوی استاندارد
bsSingle	اندازه فرم قابل تغییر نیست ؛حاشیه یک خطی
bsNone	اندازه فرم قابل تغییر نیست ؛حاشیه ندارد
bsSizeable	اندازه فرم قابل تغییر است ؛حاشیه استاندارد دارد
bsToolWindow	همچون bsSingle ولی نوار عنوان نازکتر است
bsSizeToolWin	همچون bsSizeable ولی نوار عنوان نازکتر است

ویژگی Position

این ویژگی موقعیت فرم در صفحه نمایش را مشخص می کند . فرم کلی این ویژگی به شکل زیر می باشد

enum TPosition { poDesigned, poDefault, poDefaultPosOnly, poDefaultSizeOnly, poScreenCenter, poDesktopCenter, poMainFormCenter, poOwnerFormCenter };

__property TPosition Position = {read=FPosition, write=SetPosition, default=poDesigned};

مقدار این ویژگی می تواند ترکیبی از مقادیر مجموعه شمارشی TPosition باشد که مفهوم هر کدام در جدول زیر آورده شده است.

مقدار	مفهوم
poDesigned	فرم در همان مکانی ظاهر می شود که در زمان طراحی در آن موقعیت قرار داده شده باشد
poDefault	موقعیت و اندازه فرم در زمان نمایش توسط سیستم عامل تعیین می شود
poDefaultPosOnly	فقط موقعیت فرم در زمان نمایش توسط سیستم عامل تعیین می شود و سیستم عامل حق تغییر اندازه فرم را ندارد
poDefaultSizeOnly	اندازه فرم در زمان نمایش توسط سیستم عامل تعیین می شود و سیستم عامل حق تغییر

	موقعیت فرم را ندارد
poScreenCenter	موقعیت فرم در زمان نمایش توسط سیستم عامل و در وسط صفحه نمایش تعیین می شود و سیستم عامل حق تغییر اندازه فرم را ندارد.(نکته : یک صفحه نمایش می تواند به اندازه چند صفحه مانیتور باشد.)
poDesktopCenter	موقعیت فرم در زمان نمایش توسط سیستم عامل و در وسط صفحه دسک تاپ تعیین می شود و سیستم عامل حق تغییر اندازه فرم را ندارد.
poMainFormCenter	موقعیت فرم در زمان نمایش توسط سیستم عامل و در وسط فرم اصلی تعیین می شود
poOwnerFormCenter	موقعیت فرم در زمان نمایش توسط سیستم عامل و در وسط فرمی که به عنوان Owner این فرم مشخص شده باشد تعیین می شود

ویژگی Height

این ویژگی اندازه ارتفاع فرم در صفحه نمایش را مشخص می کند . فرم کلی این ویژگی به شکل زیر می باشد

```
__property int Height = {read=FHeight, write=SetHeight, nodefault};
```

ویژگی Width

این ویژگی اندازه پهنای فرم در صفحه نمایش را مشخص می کند . فرم کلی این ویژگی به شکل زیر می باشد

```
__property int Width = {read=FWidth, write=SetWidth, nodefault};
```

ویژگی Top

این ویژگی اندازه مختصات محور Y مربوط به نقطه گوشه سمت چپ بالای فرم ر صفحه نمایش را مشخص می کند . فـرم کلی این ویژگی به شکل زیر می باشد

```
__property int Top = {read=FTop, write=SetTop, nodefault};
```

ویژگی Left

این ویژگی اندازه مختصات محور X مربوط به نقطه گوشه سمت چپ بالای فرم ر صفحه نمایش را مشخص می کند . فـرم کلی این ویژگی به شکل زیر می باشد

```
__property int Left = {read=FLeft, write=SetLeft, nodefault};
```

بررسی متد های مهم

متد Close

فرم را می بندد. فرم کلی این متد به شکل زیر می باشد

```
void __fastcall Close(void);
```

مثال :

```
This->Close();
```

متد Show

فرم را بر روی صفحه نمایش می دهد. فرم کلی این متد به شکل زیر می باشد

```
HIDESBASE void __fastcall Show(void);
```

مثال :

```
Form1->Show();
```

متد ShowModal

همچون متد Show فرم را بر روی صفحه نمایش می دهد با این تفاوت که فرم های دیگر نمی تواند بر روی فرم فعلی ظاهر شـوند . در واقع اجرای بقیه برنامه منوط به بست شدن فرم فعلی است.. فرم کلی این متد به شکل زیر می باشد

```
virtual int __fastcall ShowModal(void);
```

مثال :

```
Form1-> ShowModal ();
```

متد SetFocus

کنترل را به فرم مورد نظر منتقل می کند. فرم کلی این متد به شکل زیر می باشد

```
virtual void __fastcall SetFocus(void);
```

مثال :

```
Form1-> SetFocus ();
```

بررسی کنترل لیبل :

از لیبل ها برای برچسب گذاری استفاده می شود . معمولا این کنترل همراه کنترل هایی همچون کادر متن و .. بکار می رود تا کـاربر در زمان کار با برنامه بداند که چه مقادیر/ داده هایی را باید به کنترل ها نسبت دهد

بررسی ویژگی های مهم

ویژگی های Left ، Top ، Height ، Width ، Name که قبلا توضیح آن در کنترل فرم داده شده است

نکته : تمامی این ویژگی ها از طریق پنجره Object Inspector قابل تنظیم است.

ویژگی Caption

این ویژگی عنوان برچسپ را مشخص می کند. این عنوان بر روی صفحه نمایش قابل روئیت است . فرم کلی این ویژگی بـه شکل زیر می باشد

__property TCaption Caption = {**read**=GetText, **write**=SetText};

که در آن TCaption از نوع کلاس Widestring است

typedef Widestring TCaption;

ویژگی Alignment

از این ویژگی برای تعیین تراز افقی متن برچسب استفاده می کنیم.. فرم کلی این ویژگی به شکل زیر می باشد

__property Classes::TAlignment **Alignment = {read**=FAlignment**, write**=SetAlignment**,** default=0**};**

مقدار این ویژگی می تواند یکی از مقادیر نوع شمارشی زیر باشد

enum TAlignment { taLeftJustify, taRightJustify, taCenter };

که در آن taLeftJustify برای تراز از چپ ، taRightJustify برای تراز از راست و taCenter برای تراز از وسط می باشد

سئوال: در صورتی که بخواهیم متن موجود در یک برچسب به صورت وسط چـین نمـایش داده شـود . مقـدار ویژگـی Alignment آن را به چه مقداری باید نتظیم کنیم.

جواب : مقدار ویژگی Alignment آن را به taCenter تنظیم می کنیم

به منظور جلوگیری از افزایش حجم جزوه در ادامه از ارائه فرم کلی ویژگی ها و متدها تا حد امکان خودداری می شود

وبژگی AutoSize

این ویژگی تعیین می کند که آیا اندازه کنترل برچسب متناسب با اندازه متن موجود در Caption آن قابل تغییر باشد / خیر . در صورتی که مقدار این ویژگی برابر True باشد . اندازه کنترل برچسب متناسب با اندازه متن موجود در Caption آن تغییر می کند . در غیر این صورت اندازه برچسب با تغییر متن عنوان برچسب تغییر نمی کند

ویژگی Align : این ویژگی نحوه قرارگیری کنترل مورد نظر را در داخل یک پنجره نسبت به سایر کنترل ها مشخص می کند . مقدار این ویژگی با توجه به جدول زیر می تواند انتخاب شود

مقدار	مفهوم
alNone	محل کنترل همان محلی است که در زمان طراحی در آن قرار گرفته باشد. این مقدار به عنوان پیش فرض ویژگی Align مورد استفاده قرار می گیرد

مقدار	مفهوم
alTop	کنترل در بالاترین نقطه پنجره والدش قرار می گیرد و پهنایش به قدری بزرگ می شود که تمام پهنای کنترل را بپوشاند
alBottom	کنترل در پایین ترین نقطه پنجره والدش قرار می گیرد و پهنایش به قدری بزرگ می شود که تمام پهنای کنترل را بپوشاند
alLeft	کنترل در سمت چپ پنجره والدش قرار می گیرد و ارتفاعش به قدری بزرگ می شود که تمام ارتفاع کنترل را بپوشاند
alRight	کنترل در سمت راست پنجره والدش قرار می گیرد و ارتفاعش به قدری بزرگ می شود که تمام ارتفاع کنترل را بپوشاند
alClient	کنترل به قدری بزرگ می شود تا فضای باقیمانده پنجره والدش را تصاحب کند.

ویژگی BiDiMode

برای تعیین جهت دوسویه کنترل بکار می رود که می تواند یکی از چهار مقادیر زیر را به خود بگیرد

مقدار	مفهوم
bdLeftToRight	ترتیب خواندن از چپ به راست است ، Alignment تغییر نمی کند ، نوار مرور گر عمودی در سمت راست کنترل ظاهر می شود
bdRightToLeft	ترتیب خواندن از راست به چپ است ، Alignment تغییر می کند ، نوار مرور گر عمودی در سمت چپ کنترل ظاهر می شود
bdRightToLeftNoAlign	ترتیب خواندن از راست به چپ است ، Alignment تغییر نمی کند ، نوار مرور گر عمودی در سمت چپ کنترل ظاهر می شود
bdRightToLeftReadingOnly	فقط ترتیب خواندن از راست به چپ است ، Alignment و نوار مرورگر عمودی تغییر نمی کند

ویژگی Enabled

فعال / غیر فعال بودن کنترل را مشخص می کند

نکته : جنانچه عنصری غیرفعال باشد نمی تواند به رویدادهای صفحه کلید ، موس و تایمر پاسخ دهد.

ویژگی Font

فونت مورد استفاده برای کنترل را مشخص می کند

ویژگی Hint

به کمک این ویژگی می توان یک متن کمکی برای زمانی که کاربر موس را بر روی کنترل نگه می دارد را به عنصر مورد نظر نسبت داد. این متن در صورتی که مقدار ویژگی ShowHint برابر True باشد قابل نمایش است.

ویژگی ShowHint

نمایش / عدم نمایش متن کمکی موجود در ویژگی Hint کنترل مورد نظر را مشخص می کند، مقدار این ویژگی می تواند True و یا False باشد

ویژگی Transparent

تعیین می کند که آیا کنترل از رنگ زمینه اش استفاده کند / خیر. در صورتی که مقدار این ویژگی برابر True باشد . رنگ زمینه کنترل شفاف/ بی رنگ فرض می شود

ویژگی Visible

نمایش / عدم نمایش کنترل را بر روی صفحه نمایش مشخص می کند. در صورتی که مقدار این ویژگی False باشد کنترل مورد نظر نمایش داده نمی شود

بررسی کنترل دکمه فشاری / TButton

از این کنترل برای ایجاد دکمه های کنترلی در برنامه استفاده می کنیم

بررسی ویژگی های مهم

ویژگی های Left ، Top ، Height ، Width ، Name که قبلا توضیح آن در کنترل فرم داده شده است

ویژگی های BiDiMode ، Enabled ، Font ، Visible ، Hint ، ShowHint که قبلا توضیح آن در کنترل لیبل داده شده است

ویژگی Cancel

جنانچه که این ویژگی برابر True باشد . در صورت فشرده شدن دکمه ESC رخداد کلیک مربوط به دکمه مورد نظر فراخوانی می شود . معمولا در هر فرم یک دکمه (دکمه خروج) با خاصیت True برای این ویژگی قرار می دهند.

ویژگی Default

جنانچه که این ویژگی برابر True باشد . در صورت فشرده شدن دکمه Enter رخداد کلیک مربوط به دکمه مورد نظر فراخوانی می شود . معمولا در هر فرم یک دکمه (دکمه ثبت) با خاصیت True برای این ویژگی قرار می دهند.

ویژگی TabOrder

این ویژگی ترتیب بدست آوردن کنترل و فوکوس را در صورتی که کاربر از دکمه Tab برای حرکت بین عناصر استفاده کند تعیین می کند . مقدار این ویژگی از صفر تا n-1 است که n برابر تعدادکنترل هایی است که می توانند این ویژگی را داشته باشند . با افزودن هر کنترل مقدار این ویژگی به طور خودکار تنظیم می شود . طراح می تواند این ترتیب را به دلخواه تغییر دهد

ویژگی TabStop

این ویژگی تعیین می کند که آیا کاربر می تواند با پرش توسط دکمه Tab به کنترل مورد نظر دسترسی داشته باشد. در صورتی که مقدار این ویژگی برابر False باشد کاربر نمی تواند با استفاده از دکمه Tab و حرکت بین کنترل ها به کنترل مورد نظر دسترسی داشته باشد.. مقدار پیش فرض این ویژگی برابر True می باشد

بررسی رویداد های مهم

رویداد های قابل بررسی در کنترل ها عموما شامل رویدادهای موس که عبارتند از کلیک ، دابل کلیک ، فشردن موس ، رها کردن موس ، حرکت دادن موس و ... و نیز رویدادهای صفحه کلید شامل فشردن کلید ، رها کردن کلید و ... ، و نیز رویدادهای مربوط به دیافت و از دست دادن کنترل می باشند . هر کنترل دارای تعدای رویداد است و طراح و برامه نویس وظایف برنامه را بر حسب رویدادهای محیط برنامه نویسی و مشخص می کند .و مثلا برای یک دکمه تعیین می کند که اگر بر روی دکمه کلیک شود جه عملی باید انجام شود. یا مثلا وقتی کع یک فرم فعال شود چه کاری باید انجام شود . و ...

بنابراین به جای اینکه رویداده های کنترل ها را بررسی کنیم باید بتوانیم بر روی رویدادها برنامه ریزی انجام دهیم که این موضوع به کمک متدها و ویژگی های مریوط به اشیاء ، کنترل ها و کامپوننت ها میسر می شود

بررسی کنترل متن / TEdit

بررسی ویژگی های مهم

ویژگی های Left ، Top ، Height ، Width ، Name که قبلا توضیح آن در کنترل فرم داده شده است

ویژگی های AutoSize ، BiDiMode ، Color ، Enabled ، Font ، Visible ، Hint ، ShowHint که قبلا توضیح آن در کنترل لیبل داده شده است

ویژگی MaxLength

این ویژگی حداکثر تعداد کاراکترهای قابل ذخیره در کنترل را مشخص می کند که کاربر می تواند وارد کند در صورتی که مقدار این کنترل برابر صفر باشد ، محدودیتی در تعدا کاراکترها اعمال نمی شود

ویژگی ReadOnly

این ویژگی تعیین می کند که آیا متن موجود در کنترل توسط کاربر قابل تغییر هست / خیر

ویژگی Text

این ویژگی حاوی متن موجود در کنترل است . این ویژگی از نوع کلاس AnsiString بوده و بنابراین تمامی مواردگفته شده در مورد کلاس فوق در مورد این ویژگی صدق می کند

ویژگی TabOrder

این ویژگی ترتیب بدست آوردن کنترل و فوکوس را در صورتی که کاربر از دکمه Tab برای حرکت بین عناصر استفاده کند تعیین می کند . مقدار این ویژگی از صفر تا n-1 است که n برابر تعدادکنترل هایی است که می توانند این ویژگی را داشته باشند . با افزودن هر کنترل مقدار این ویژگی به طور خودکار تنظیم می شود . طراح می تواند این ترتیب را به دلخواه تغییر دهد

ویژگی TabStop

این ویژگی تعیین می کند که آیا کاربر می تواند با پرش توسط دکمه Tab به کنترل مورد نظر دسترسی داشته باشد. در صورتی که مقدار این ویژگی برابر False باشد کاربر نمی تواند با استفاده از دکمه Tab و حرکت بین کنترل ها به کنترل مورد نظر دسترسی داشته باشد.. مقدار پیش فرض این ویژگی برابر True می باشد

بررسی متد های مهم

مثد CanFocus

این مند تعیین می کند که آیا یک کنترل می تواند به عنوان کنترل فعال انتخاب شود / خیر . استفاده از این متد قبل از استفاده از متد SetFocus امری حیاتی است چرا که در صورتی که یک برنامه بخواهد فوکوس را به یک کنترل غیرفعال بفرستد با خطا مواجه می شود

مثال

```
If(Edit1->CanFocus() )  Edit1->SetFocus();
```

متد Clear

متن موجود در کنترل را ار بین می برد

مثال : حذف متن موجود در ویژگی Text مربوط به کنترل Edit1

```
Edit1->Clear();
```

متد ClearSelection

باعث یاک شدن متن انتخاب شده موجود می شود

مثال : حذف متن انتخاب شده موجود در ویژگی Text مربوط به کنترل Edit1

```
Edit1->ClearSelection();
```

متد ClearUndo

باعث یاک شدن بافر مربوط به Undo می شود . اجرای این متد باعث می شود تا تغییرات اعمال شده در متن قبل از اجرای این متد دائمی فرض شود و امکان برگشت تغییرات با فشردن کلید Ctrl+Z میسر نباشد

مثال : حذف بافر Undo

```
Edit1->ClearUndo();
```

متد CopyToClipboard

باعث کپی شدن متن انتخاب شده در کنترل به حافظه کلیپ برد می شود

مثال :

```
Edit1->CopyToClipboard();
```

متد CutToClipboard

باعث برش متن انتخاب شده و کپی شدن آن به حافظه کلیپ برد می شود

مثال :

```
Edit1->CutToClipboard();
```

متد PasteFromClipboard

باعث انتقال متن موجود در حافظه کلیپ برد به کنترل می شود

مثال :

```
Edit1->PasteFromClipboard();
```

متد SelectAll

کل متن موجود در کنترل را مورد انتخاب قرار می دهد

مثال :

```
Edit1->SelectAll();
```

تمرین :

۱- یک فرم شامل دو کنترل متن ، و چهار دکمه برای انجام عملیات SelectAll ، Copy ، Cut و Paste ایجاد کنید.

به طوری که کاربر بتواند متنی را در کنترل متن شماره ۱ وارد کند و سپس

۱- در صورت فشردن دکمه Select All کل متن موجود در کنترل متن شماره ۱ انتخاب شود

۲- در صورت فشردن دکمه Copy متن انتخاب شده به حافظه کلیپ برد کپی شود

۳- در صورت فشردن دکمه Cut متن انتخاب شده از کنترل متن شماره ۱ حذف و در حافظه کلیپ برد کپی شود

۴- در صورت فشردن دکمه Paste متن موجود در کنترل شماره ۲ حذف شود و سپس متن موجود در کلیپ برد به کنترل متن شماره ۲ منتقل شود.

۲-تمرین قبل را طوری تغییر دهید که

- دکمه Select All زمانی فعال باشد که متنی در کادر متن شماره ۱ وجود داشته باشد
- دکمه Cut ,Copy زمانی فعال باشند که متن موجود در کادر متن شماره ۱ مورد انتخاب واقع شده باشد
- دکمه paste زمانی فعال باشد که متنی در حافظه کلیپ برد وجود داشته باشد

بررسی کنترل توضیحات / TMemo

بررسی ویژگی های مهم

ویژگی های Left ، Top ، Height ، Width ، Name که قبلا توضیح آن در کنترل فرم داده شده است

ویژگی هایAlignment, Align, BiDiMode ، Color ، Enabled ، Font ، Visible ، Hint ، ShowHint که قبلا توضیح آن در کنترل لیبل داده شده است

ویژگی TabOrder , TabStop , Text, MaxLength,ReadOnly که قبلا در کنترل متن توضیح داده شده است

ویژگی Lines

این ویژگی از نوع کلاس TStrings و در برگیرنده متن موجود در کنترل می باشد. تمامی خصوصیات ومتد های گفته شده در مورد TStrings در مورد این ویژگی صدق می کند

ویژگی WordWrap

این ویژگی تعیین می کند که در صورتی که طول یک یاراگراف متن بیشتر از پهنای کادر کنترل توضیحات باشد، کنترل پاراگراف را طوری بشکند که عرض پاراگراف در پهنای کادر جای گیرد.

ویژگی ScrollBars

این ویژگی نمایش / عدم نمایش نوار های مرورگر افقی ، عمودی و یا هر دونوار با هم را برای کنترل Memo در صورتی که ابعاد متن موجود در کنترل بیشتر از ابعاد کادر کنترل باشد را مشخص می کند.

مقدار این ویژگی می تواند یکی از مقادیر ssNone برای عدم نمایش ، ssVertical برای نمایش نوار عمودی ،ssHorizontal برای نمایش نوار افقی و ssBoth برای نمایش هر دو نوار باشد

نکته : در صورتی که مقدار ویژگی WordWrap برابر True باشد از نمایش نوار مرورگر افقی صرفنظر می شود

بررسی متد های مهم

متد ها همان هایی است که در رابطه با کنترل TEdit بررسی شد به اضافه متد های به ارث برده شده از کلاس Tstrings که مربوط به ویژگی Lines کنترل توضیحات می شود

تمرین :

۱- برنامه ای بنویسید که بتواند محتویات یک فایل را در یک کنترل Memo نمایش دهد.

۲- برنامه فوق را طوری تغییر دهید که بتواند یک متن دلخواه را در داخل محتویات فایل جستجو کند