فصل 3

نوع ها، نگهداری و متغییر ها

* یک برنامه C# مجموعه ای از نوع های تعریف شده است
* نوع یک الگو است
* نمونه سازی از یک نوع
* داده های عضو و توابع عضو
* نوع های از پیش تعریف شده
* نوع های تعریف شده توسط کاربر
* Stack و Heap
* نوع های مقداری و نوع های ارجاعی
* متغییر ها
* نوع های static و کلمه کلیدی dynamic
* نوع های تهی

یک برنامه C# مجموعه ای از نوع های تعریف شده است

اگر قرار باشد که کد منبع برنامه های C و C++ را به طور کلی توصیف کنید، ممکن است بگویید که یک برنامه C مجموعه ای از توابع و نوع های داده ای است و برنامه C++ مجموعه ای از توابع و کلاس ها است . یک برنامه C# مجموعه ای از نوع های تعریف شده است.

* کد منبع یک برنامه C# یا DLL، مجموعه ای از یک یا چند نوع تعریف شده است.
* برای اجرای یکی از نوع های تعریف شده، باید کلاسی وجود داشته باشد که متد Main را شامل شود.
* فضای نام راهی برای دسته بندی و نام گذاری مجموعه ای از نوع های تعریف شده است. از آنجایی که یک برنامه با مجموعه ای از نوع های تعریف شده در ارتباط است، به طور کلی نوع های این برنامه در داخل یک فضای نام تعریف می شود.

به عنوان مثال، کد زیر نشان دهنده برنامه ای است که شامل سه نوع تعریف شده است. این سه نوع در داخل فضای نام MyProgram تعریف شده است.

|  |
| --- |
| namespace MyProgram // ایجاد فضای نام جدید  {  *DeclarationOfTypeA*  //تعریف نوع  *DeclarationOfTypeB*  // تعریف نوع  class C // تعریف نوع  {  static void Main()  {  ...  }  }  } |

فضاهای نام با جزئیات بیشتری در فصل 21 بررسی می شوند.

نوع یک الگو است

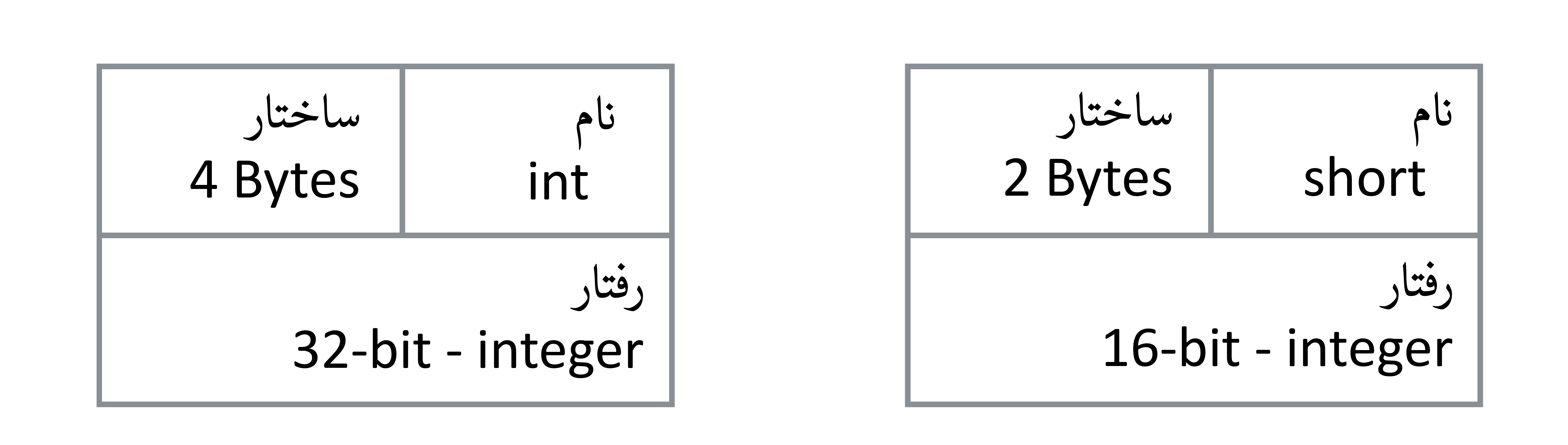
از آنجایی که یک برنامه C# مجموعه ای از نوع های تعریف شده است، یادگیری C# شامل یادگیری نحوه ایجاد و استفاده از نوع ها است. بنابراین، اولین چیز که شما برای یادگیری نیاز دارید این است که بدانید یک نوع چیست.

می توان تصور کرد که یک نوع همانند الگویی برای ایجاد ساختمان های داده است. این به خودی خود ساختمان داده نیست، اما خصوصیات اشیا ساخته شده از الگو را مشخص می کند.

یک نوع به وسیله عناصر زیر تعریف می شود:

* نام
* ساختمان داده شامل عضو های داده مربوط به خود است
* رفتار ها و محدودیت ها

به عنوان مثال، شکل 1-3 اجزای دو نوع short و int را نشان می دهد.



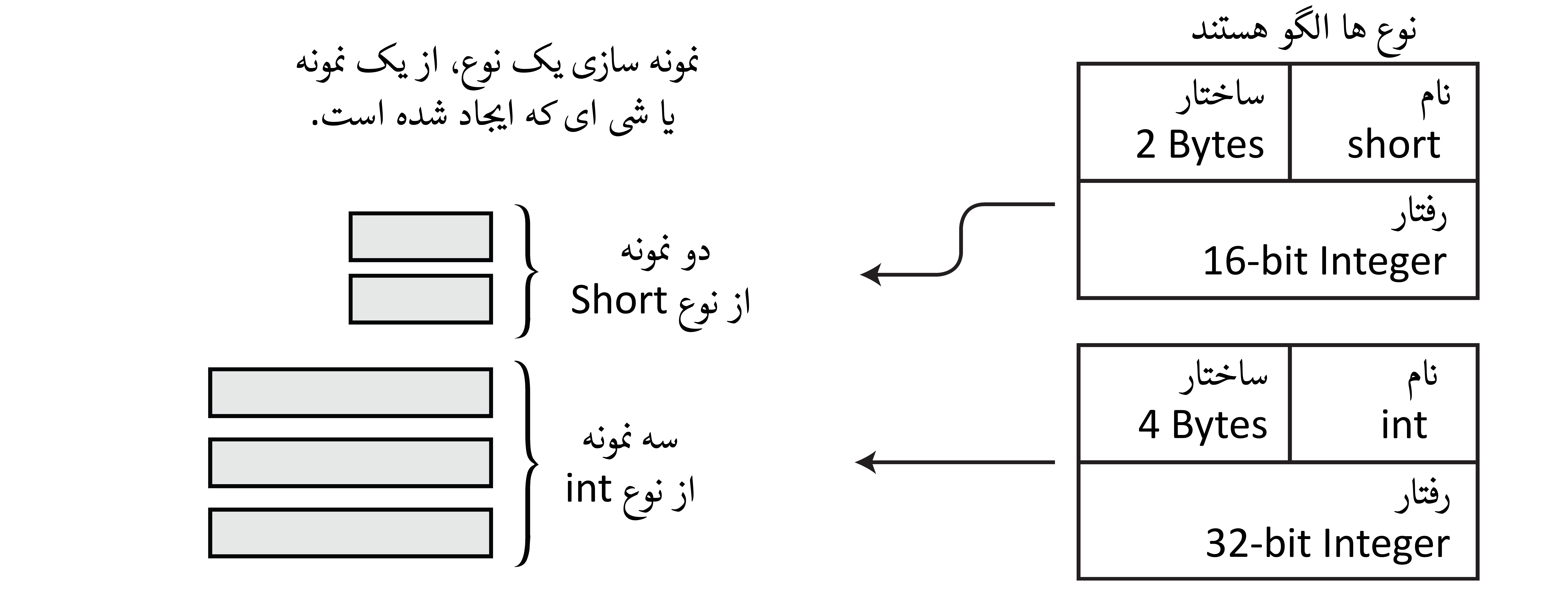
شکل 1-3. نوع یک الگو است

نمونه سازی یک نوع

ایجاد شی واقعی از یک نوع را نمونه سازی آن نوع می گویند.

* شی هایی که با نمونه سازی از یک نوع ایجاد می شوند را نمونه ای از آن نوع می گویند.
* هر آیتم داده در برنامه C# نمونه ای از یک نوع است. یک نوع یا توسط زبان ارائه شده است (توسط BCL یا توسط دیگر کتابخانه ها)، یا توسط برنامه نویس تعریف شده است.

شکل 2-3 نمونه ای از اشیا، دو نوع از پیش تعریف شده است را نشان می دهد.



شکل 2-3. نمونه سازی یک نوع، از نمونه ای که ایجاد شده است.

داده های عضو و توابع عضو

برخی از نوع ها از جمله short و int و long نوع های ساده نامیده می شوند و می توانند تنها یک آیتم را ذخیره کنند.

نوع های دیگر می توانند چندین آیتم را ذخیره کنند. برای مثال آرایه نوعی است که می تواند آیتم های متعددی از همان نوع را ذخیره کند. آیتم های منحصر به فرد عنصر نامیده می شوند و توسط یک عدد به آن مراجعه می شود که index نام دارد. در فصل 12 جزئیات آرایه ها بررسی می شود.

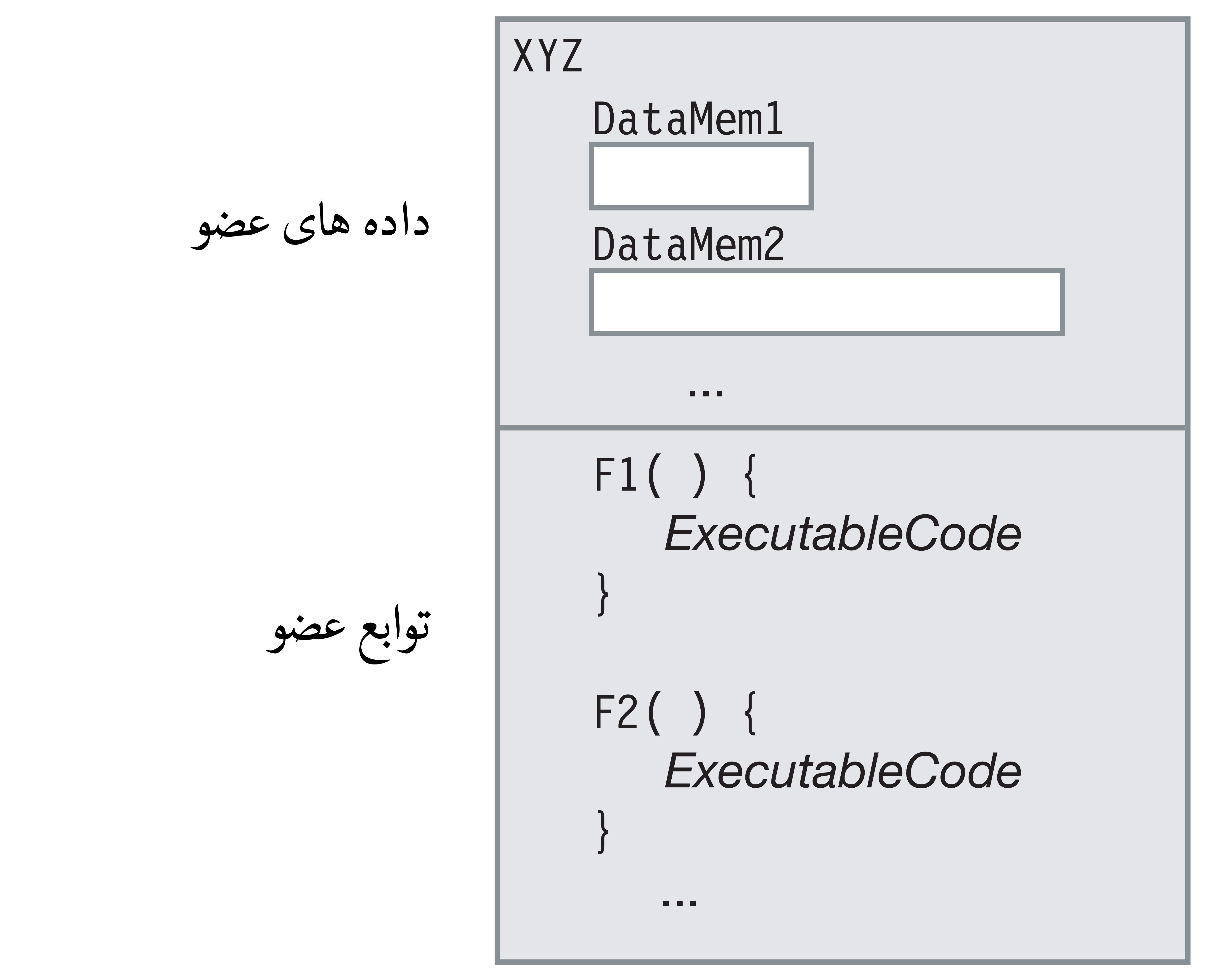
عضو های نمونه

نوع های دیگر می توانند شامل آیتم هایی از نوع ها مختلف باشند. عناصر منحصر به فرد در این نوع، عضو نامیده می شود و برخلاف آرایه ها، که با عدد به هر عضو مراجعه می شد، در این نوع ها، هر عضو نام مشخصی دارد.

دو نوع عضو وجود دارد: داده های عضو و توابع عضو.

* داده های عضو، که داده های مربوط به یک شی از کلاس را ذخیره می کنند.
* توابع عضو، کد های اجرایی هستند. توابع عضو تعیین می کنند که یک نوع چگونه عمل می کند.

به عنوان مثال، شکل 3-3 برخی از داده های عضو و توابع عضو از نوع XYZ را نشان می دهد. و شامل دو داده عضو و دو تابع عضو است.



شکل 3-3. مشخص کردن داده های عضو و توابع عضو

نوع های از پیش تعریف شده

C#، 16 نوع از پیش تعریف شده را فراهم می کند، که در شکل 4-3 نشان داده شده است و در جدول های 1-3 و 2-3 ذکر شده اند. آنها شامل 13 نوع ساده و 3 نوع غیر ساده هستند.

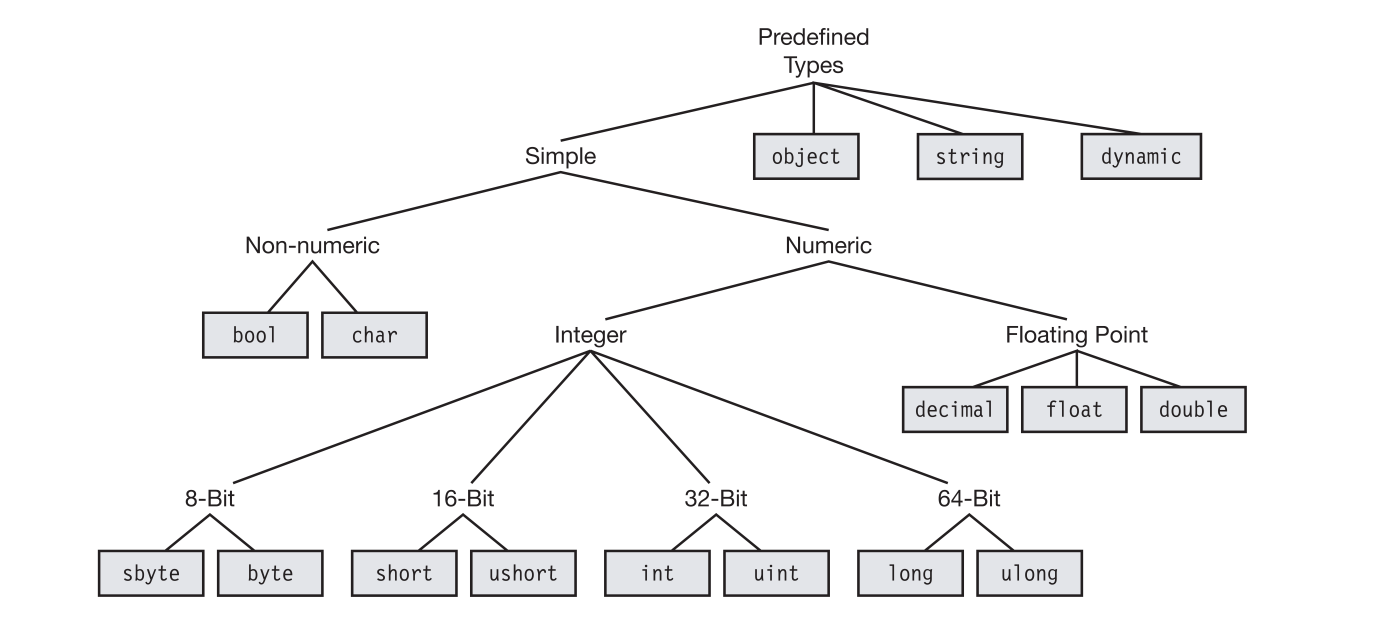
کاراکتر های تمام نوع های از پیش تعریف شده، از حروف کوچک تشکیل شده اند. نوع های ساده از پیش تعریف شده عبارتند از :

* یازده نوع عددی، شامل موارد زیر:
  + طول های مختلفی از نوع های صحیح علامت دار و بدون علامت.
  + نوع های اعشاری – float و double .
  + دقت بالا از نوع اعشاری به نام decimal . برخلاف float و double ، نوع decimal می تواند کاملا کسر اعداد اعشاری را نمایش دهد. که اغلب برای محاسبات پولی استفاده می شود.
* نوع کاراکتری یونیکد به نام char .
* نوع بولی به نام bool . نوع bool یک مقدار Boolean را نمایش می دهد و باید یکی از دو مقدار true ( صحیح ) یا false ( غلط ) باشد.

|  |
| --- |
| * توجه بر خلاف C و C++ ، در C# مقادیر عددی تعبیری از Boolean ندارند. |

سه نوع غیر ساده به شرح زیر است:

* نوع string که آرایه ای از کاراکتر های یونیکد است.
* نوع object مبنای تمام نوع های دیگر است.
* نوع dynamic، در هنگام استفاده از اسمبلی های نوشته شده به زبان پویا استفاده می شود.



شکل 4-3. نوع های از پیش تعریف شده

توضیحاتی بیشتر درباره نوع های از پیش تعریف شده

تمام نوع های از پیش تعریف شده به طور مستقیم به نوع های .NET نگارش می شوند. نام نوع ها C#، فقط نام های مستعار برای نوع های .NET است، بنابراین استفاده از نام های .NET در C# از لحاظ دستوری درست عمل می کند، ولی این کار توصیه نمی شود. در داخل برنامه C# ، باید از نام های C# به جای نام های .NET استفاده کرد.

نوع های ساده از پیش تعریف شده، آیتمی جدا از داده را نشان می دهد. آنها در جدول 1-3 همراه با مقادیر قابل قبولی که می توانند اختیار کنند ذکر شده اند.

جدول 1-3. نوع های ساده از پیش تعریف شده

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| نام | معنا | | محدوده | نوع .NET | | مقدار پیش فرض |
| sbyte | 8 بیت عدد صحیح علامت دار | -128 تا127 | | | System.SByte | 0 |
| byte | 8 بیت عدد صحیح بدون علامت | 0 تا 255 | | | System.Byte | 0 |
| short | 16 بیت عدد صحیح علامت دار | -32,768 تا 32,767 | | | System.Int16 | 0 |
| ushort | 16 بیت عدد صحیح بدون علامت | 0 تا 65,535 | | | System.UInt16 | 0 |
| int | 32 بیت عدد صحیح علامت دار | -2,147,483,648 تا 2,147,483,647 | | | System.Int32 | 0 |
| uint | 32 بیت عدد صحیح بدون علامت | 0 تا 4,294,967,295 | | | System.UInt32 | 0 |
| long | 64 بیت عدد صحیح علامت دار | -9,223,372,036,854,775,808 تا 9,223,372,036,854,775,807 | | | System.Int64 | 0 |
| ulong | 64 بیت عدد صحیح بدون علامت | 0 تا 18,446,744,073,709,551,615 | | | System.UInt64 | 0 |
| float | عدد اعشاری با دقت کم | 1.5×10-45 تا 3.4×1038 | | | System.Single | 0.0f |
| double | عدد اعشاری با دقت زیاد | 5×10-324 تا 1.7×10308 | | | System.Double | 0.0d |
| bool | Boolean | True و false | | | System.Boolean | false |
| char | کاراکتر یونیکد | U+0000 تا U+ffff | | | System.Char | \x0000 |
| decimal | عدد اعشاری با دقت 28 رقم اعشار | ± 1.0 X 1028 تا ±7.9 X 1028 | | | System.Decimal | 0m |

نوع های از پیش تعریف نشده تا حدی پیچیده تر است. جدول 2-3 نوع های از پیش تعریف نشده را نشان می دهد.

جدول 2-3. نوع های از پیش تعریف نشده

|  |  |  |
| --- | --- | --- |
| نام | معنا | نوع **.NET** |
| object | کلاس پایه ای که تمام نوع ها دیگر (از جمله نوع های ساده ) از آن مشتق شده است. | System.Object |
| string | کاراکتر های یونیکد متوالی | System.String |
| dynamic | نوع طراحی شده ای که با اسمبلی نوشته شده به زبان های پویا استفاده می شود. | وجود ندارد. |

نوع های تعریف شده توسط کاربر

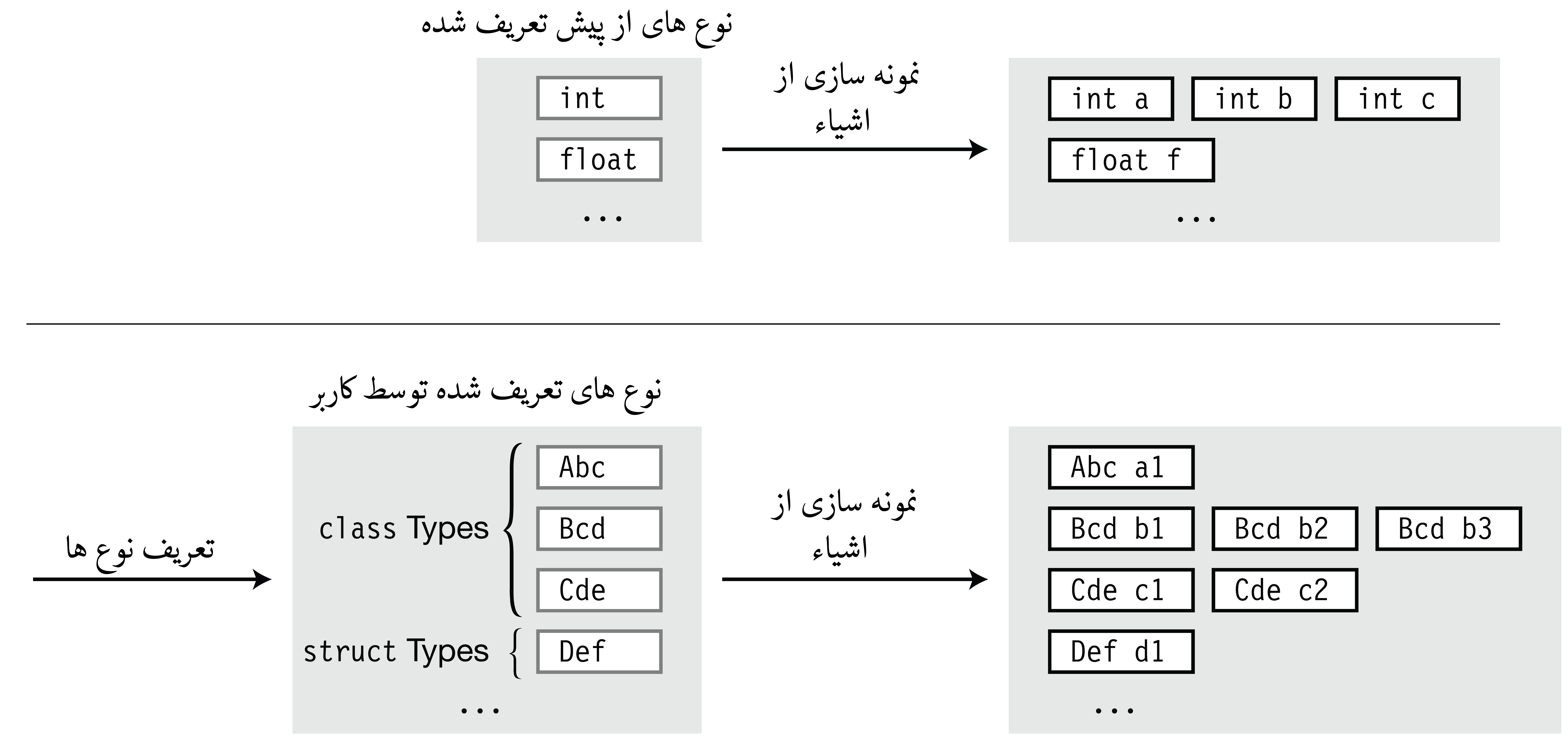
علاوه بر 16 نوع از پیش تعریف شده توسط C#، همچنین شما می توانید نوع ها تعریف شده خود را ایجاد کنید. شش نوعی که شما می توانید ایجاد کنید به شرح زیر است:

* class
* struct
* array
* enum
* delegate
* interface

زمانی که یک نوع با تعریف می شود، شامل اطلاعات زیر است:

* نوعی که برای تعریف شدن انتخاب شده است. (نوع های کلاس یا نوع های ساختار یا...).
* نامی که برای این نوع انتخاب شده است.
* تعریف کردن ( نام و مشخصات) هر یک از عضو های بجز برای نوع های آرایه و نماینده که عضو هایی برای تعریف کردن ندارند.

هنگامی که نوعی را تعریف می کنید، می توانید اشیایی از آن نوع را ایجاد و استفاده کنید، درست همانطوری که اگر آنها، از نوع های از پیش تعریف شده بودند. شکل 5-3 خلاصه ای از نمونه سازی نوع های از پیش تعریف شده و نوع های تعریف شده توسط کاربر را نشان می دهد. استفاده از نوع های از پیش تعریف شده، فرآیندی یک مرحله ای است که شما به سادگی اشیایی آن نوع را نمونه سازی می کنید. استفاده از نوع های تعریف شده توسط کاربر یک فرایند دو مرحله ای است که شما ابتدا نوع را تعریف می کنید و سپس اشیایی از آن نوع را نمونه سازی می کنید.



شکل 5-3. نوع های از پیش تعریف شده تنها نیاز به نمونه سازی دارند. در حالی که نوع های تعریف شده توسط کاربر نیاز به دو مرحله تعریف و نمونه سازی است.

Stack و Heap

اگر برنامه ای درحال اجرا باشد ،اطلاعات آن باید در حافظه نگه داری شود. مقدار حافظه مورد نیاز برای یک آیتم و مکان و چگونگی ذخیره سازی آن به نوع آن بستگی دارد.

یک برنامه درحال اجرا دو منطقه از حافظه را برای ذخیره داده ها استفاده می کند : stack و heap.

Stack

[[1]](#footnote-1)Stack آرایه ای از حافظه است که مانند یک last-in, first-out[[2]](#footnote-2) (LIFO) عمل می کند. این stack چندین نوع از داده ها را ذخیره می کند:

* مقادیر نوع های خاصی از متغییر ها
* محیط اجرایی برنامه جاری
* پارامتر های فرستاده شده به توابع

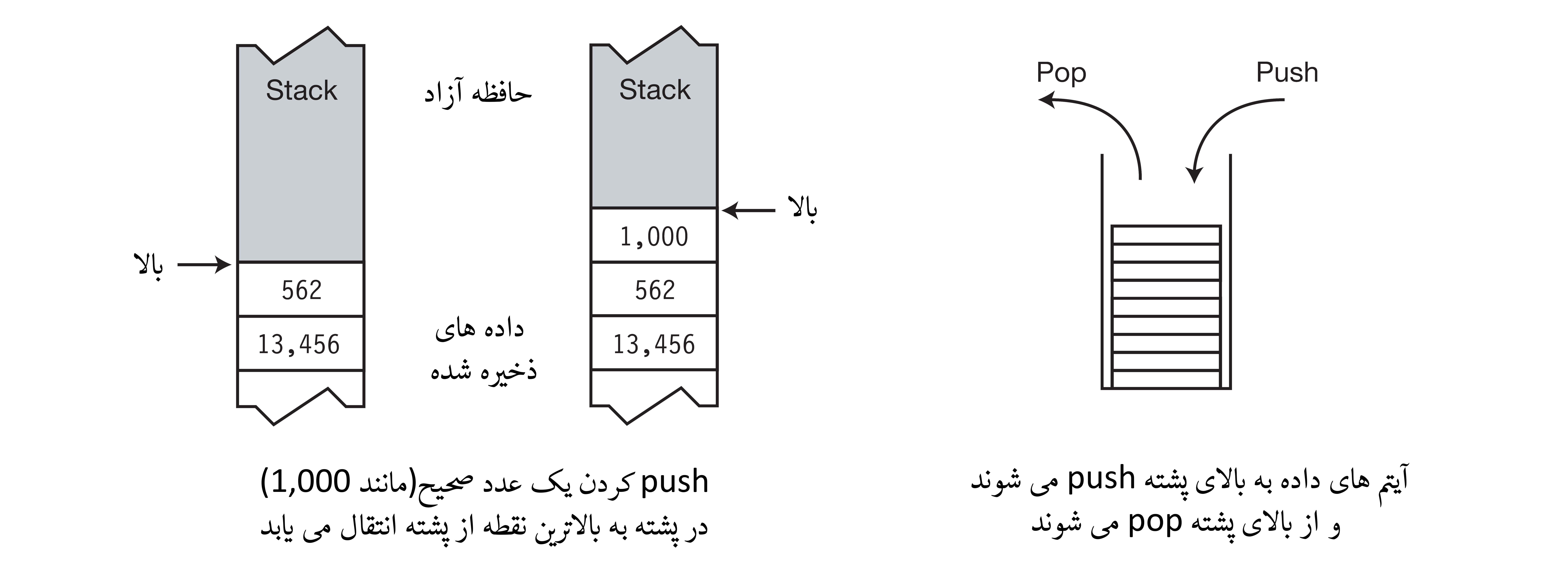
سیستم از دستکاری پشته مراقبت می کند. شما به عنوان برنامه نویس ، اجازه نخواهید داشت هر کاری را با این حافظه انجام دهید. اما فهم عملکرد آن، به شما درک بهتری از آن را می دهد.

واقعیتی درباره Stack

ویژگی های عمومی stack به شرح زیر است:

* داده ها می توانند فقط به بالای stack اضافه شوند و از بالای آن حذف شوند.
* قرار دادن یک آیتم به بالای stack را push کردن آن آیتم بر روی stack می گویند.
* حذف کردن یک آیتم از بالای stack را pop کردن آن آیتم از روی stack می گویند.

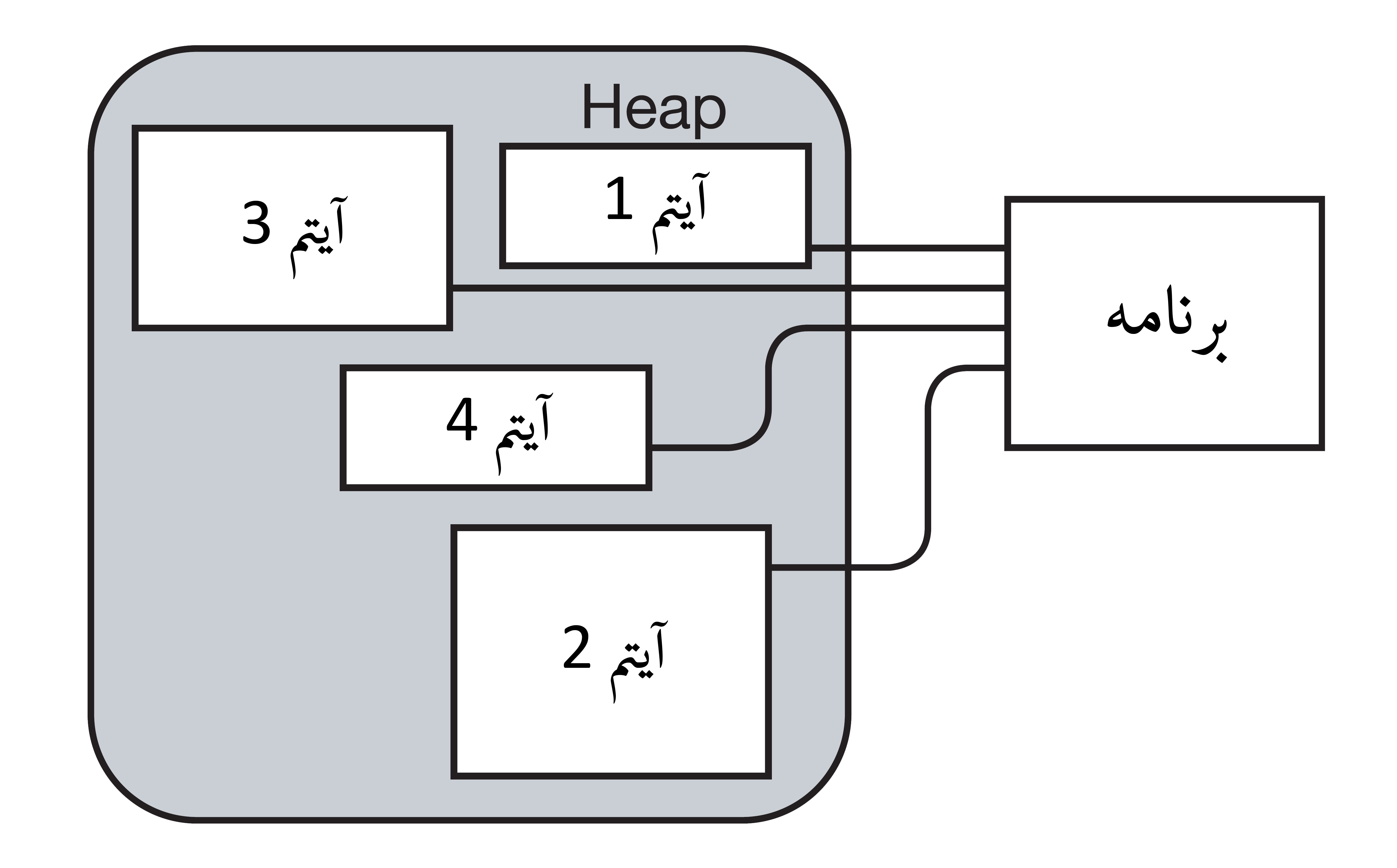
شکل 6-3 عملکرد و اصطلاحات پشته را نشان می دهد.



شکل 6-3. push وpop کردن در پشته

Heap

Heap[[3]](#footnote-3) منطقه ای از حافظه است، که در آن تکه هایی برای نگه داری نوع های خاصی از اشیا اختصاص داده می شود. برخلاف stack، آیتم های حافظه heap می توانند با هر ترتیبی ذخیره و یا حذف شوند. شکل 7-3 برنامه ای را نشان می هد که چهار آیتم را در heap ذخیره کرده است.



شکل 7-3. حافظه heap

اگر چه یک برنامه می تواند آیتم هایی را در heap ذخیره کند، اما نمی تواند به طور صریح آنها را حذف کند . در عوض گردآورنده­ زباله (GC) در CLR زمانی که تشخیص دهد برنامه دسترسی طولانی مدت به آن را ندارد، به طور خودکار اشیا بدون استفاده در heap را پاک می کند . این آزادی برای دیگر زبان های برنامه نویسی می تواند باعث به وجود آمدن خطا شود. شکل 8-3 روند جمع آوری زباله را نشان می هد.



شکل 8-3. جمع آوری زباله به صورت خودکار در heap

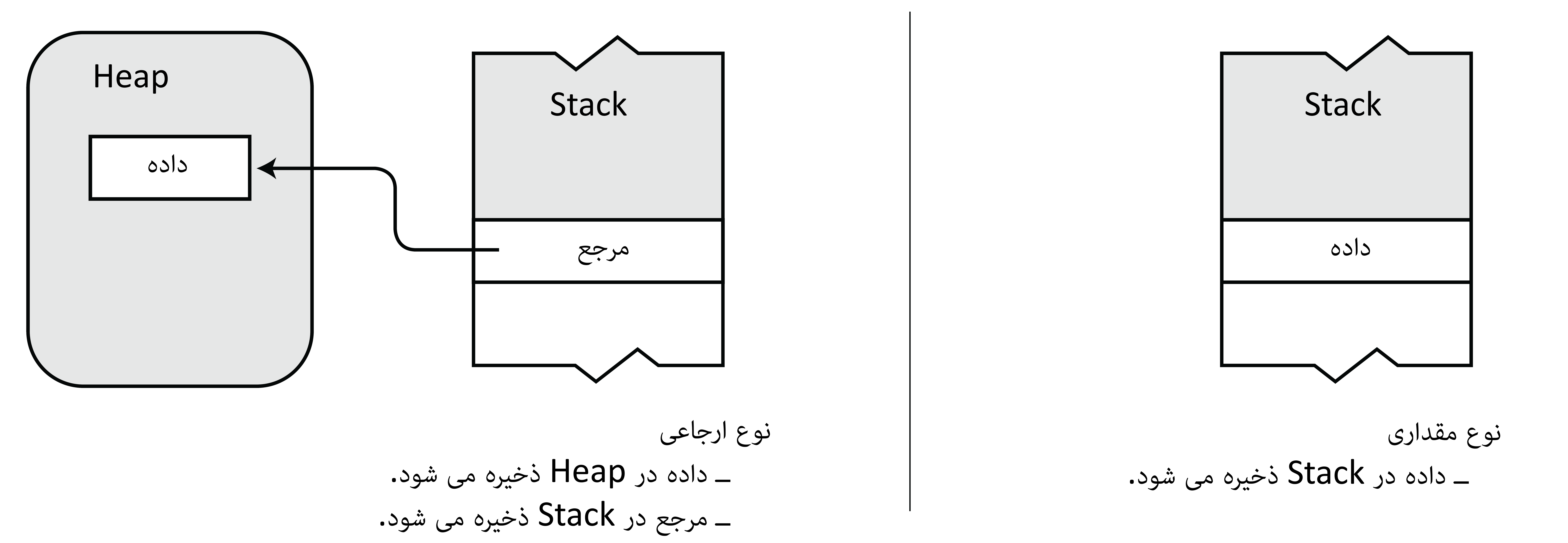
نوع های مقداری و نوع های ارجاعی

نوع یک آیتم مقدار حافظه مورد نیاز برای ذخیره سازی آن نوع و عضو هایی که شامل آن نوع است را مشخص می کند. همچنین نوع تعیین می کند که شی در حافظه stack و یا در حافظه heap ذخیره می شود.

نوع ها به دو دسته تقسیم می شوند: نوع های مقداری و نوع های ارجاعی. اشیا ایجاد شده از این نوع در حافظه به طور متفاوت ذخیره می شوند.

* نوع های مقداری تنها به یک بخش از حافظه نیاز دارند. که به طور واقعی ذخیره می شوند.
* نوع های ارجاعی به دو بخش از حافظه نیاز دارند:
  + اولین بخش شامل داده های واقعی است و همیشه در heap قرار دارد.
  + دومین بخش مرجعی است که به مکان ذخیره داده در heap، اشاره می کند.

شکل 9-3 چگونگی ذخیره نوع هایی از آیتم های داده را نشان می دهد. برای نوع های مقداری، داده ها در stack ذخیره می شوند و برای نوع های ارجاعی، داده های واقعی در heap ذخیره می شوند و مرجع ها در stack ذخیره می شوند.



شکل 9-3. ذخیره سازی داده ها

ذخیره سازی عضو های یک شی ارجاعی

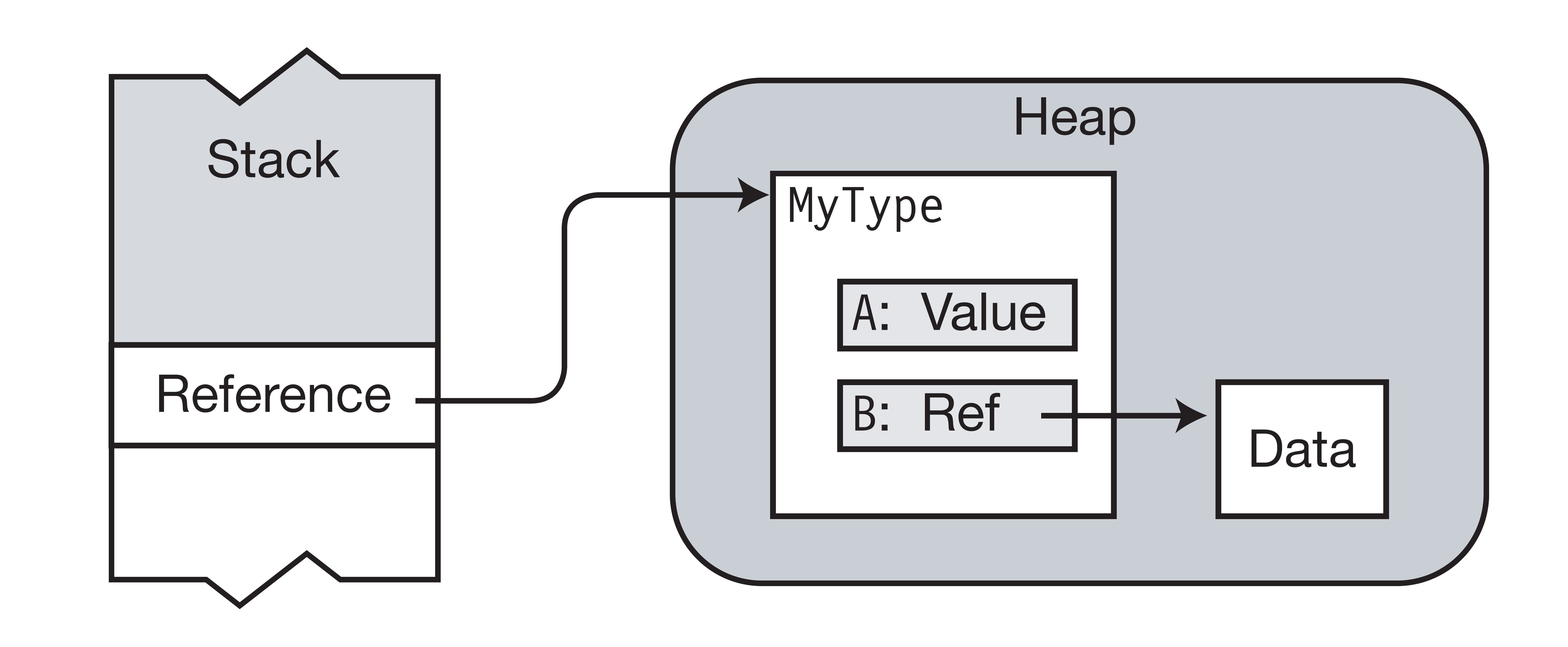
شکل 9-3 چگونگی ذخیره سازی آیتم های داده را زمانی که این عضو داده شامل نوع دیگری نباشد را نشان می دهد. هنگامی که این عضو شامل نوع های دیگر باشد، داده ممکن است کمی متفاوت تر ذخیره شود.

* بخش داده از یک شی ارجاعی همیشه در heap ذخیره می شوند، همانطور که در شکل 9-3 نشان داده شده است.
* یک شی مقداری، یا قسمت ارجاعی از نوع ارجاعی، بسته به شرایط می تواند در هر دو حافظه stack یا heap ذخیره شوند.

به عنوان مثال، فرض کنید که شما باید نمونه ای از نوع ارجاعی به نام MyType داشته باشید که دارای دو عضو، نوع مقداری و نوع ارجاعی است. چگونه این نوع ذخیره می شود؟ آیا مانند شکل 9-3، نوع مقداری در stack ذخیره می شود و نوع ارجاعی بین دو حافظه stack و heap تقسیم بندی می شود؟ پاسخ منفی است.

به یاد داشته باشید که برای یک نوع ارجاعی، داده همیشه در heap ذخیره می شود. از آنجایی که هر دو عضو، بخشی از شی هستند، هر دوی آنها در heap ذخیره می شوند، صرف نظر از اینکه آیا آنها از نوع مقداری هستند یا از نوع ارجاعی. شکل 10-3 نوع MyType را نشان می دهد.

* عضو A یک نوع مقداری است، و آن بخشی از داده MyType است و بنابراین با داده های شی در heap ذخیره می شود.
* عضو B یک نوع ارجاعی است، و بنابراین بخش داده ای آن همیشه در heap ذخیره می شود، همانطور که در شکل با مربع کوچکی با نام Data نشان داده شده است. تفاوت در این است که مرجع آن نیز در کنار بخش داده ای از شی MyType در heap ذخیره می شود.



شکل 10-3 ذخیره سازی داده ها به عنوان بخشی از نوع ارجاعی

|  |
| --- |
| * توجهبرای هر شی ارجاعی ، تمامی داده های آن عضو در heap ذخیره می شوند، صرف نظر از اینکه آنها از نوع مقداری هستند یا از نوع ارجاعی. |

طبقه بندی نوع های C#

جدول 3-3 تمام نوع های موجود در C# را نشان می دهد و آنها را با نوع های مقداری و نوع های ارجاعی دسته بندی کرده است. هر کدام از نوع های ارجاعی را بررسی خواهیم کرد.

جدول 3-3. نوع های مقداری و نوع های ارجاعی در C#

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | نوع های مقداری | | | | نوع های ارجاعی | |
| نوع های از پیش تعریف شده | sbyte | byte | float |  | object |  |
|  | short | ushort | double |  | string |  |
|  | int | uint | char |  | dynamic |  |
|  | long | ulong | decimal |  |  |  |
|  | bool |  |  |  |  |  |
| نوع های تعریف شده توسط کاربر | struct |  |  |  | class |  |
|  | enum |  |  |  | interface |  |
|  |  |  |  |  | delegate |  |
|  |  |  |  |  | array |  |

متغییر ها

یک زبان برنامه نویسی همه منظوره، باید اجازه دهد که برنامه داده ای را ذخیره و بازیابی کند.

* متغییر ها در زمان اجرای برنامه، داده ای را که در حافظه ذخیره شده است، نشان می دهد.
* C# چهار دسته از متغییر ها را ارائه کرده است. هر یک از آنها با جزئیات بیشتری مورد بررسی قرار می گیرند. این نوع ها در جدول 4-3 ذکر شده اند.

جدول 4-3. چهار نوع از متغییر ها

|  |  |
| --- | --- |
| نام | توضیحات |
| متغییر محلی | اطلاعات را به طور موقت در داخل محدوده ای از یک متد نگه داری می کند. عضوی از نوع نیست. |
| فیلد | دارای اطلاعات مرتبط با یک نوع یا نمونه ای از یک نوع است. عضوی از نوع است. |
| پارامتر | یک متغییر موقتی است که برای انتقال داده ها از یک متد به متد دیگر استفاده می شود. عضوی از نوع نیست. |
| عناصر آرایه | عضوی از یک مجموعه آیتم های هم نوع است. عضوی از نوع است. |

تعریف متغییر ها

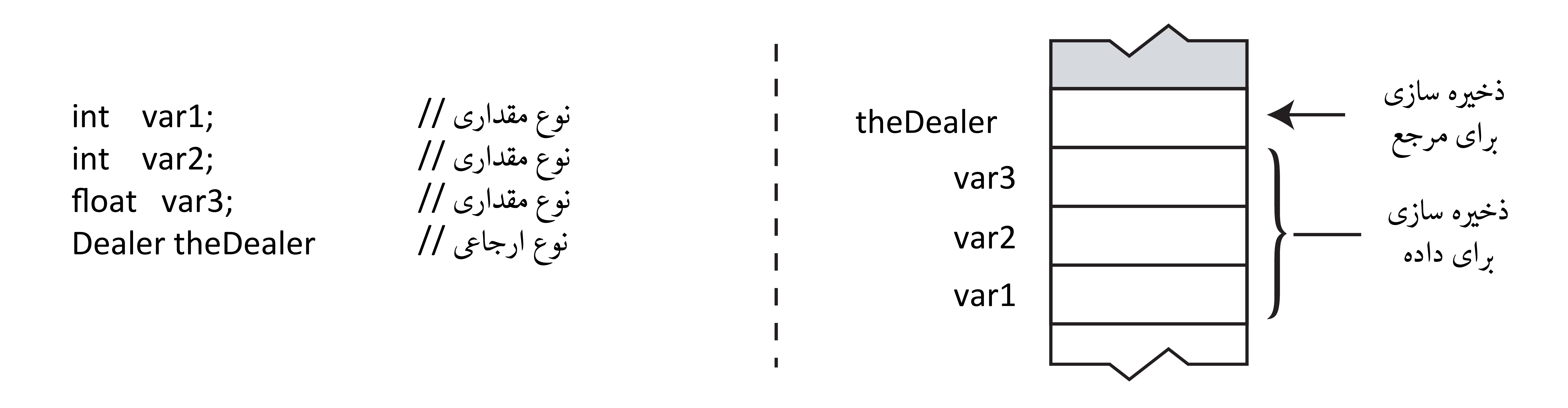
یک متغییر باید قبل از استفاده، تعریف شده باشد. در تعریف متغییر دو کار انجام می شود:

* به متغییر، نام و نوع آن تعیین می شود.
* این کار به کامپایلر اجازه تخصیص حافظه را می دهد.

تعریف کردن یک متغییر ساده حداقل به یک نوع و یک نام نیاز دارد. در تعریف زیر یک متغییر به نام var2 از نوع int تعریف شده است.

|  |
| --- |
| نوع  ↓  int var2;  ↑  نام |

به عنوان مثال، شکل 11-3 چهار متغییر را تعریف کرده است. و مکان آنها را در stack نشان می دهد.



شکل 11-3. تعریف متغییر هایی از نوع مقداری و نوع ارجاعی

مقدار دهی اولیه متغییر ها

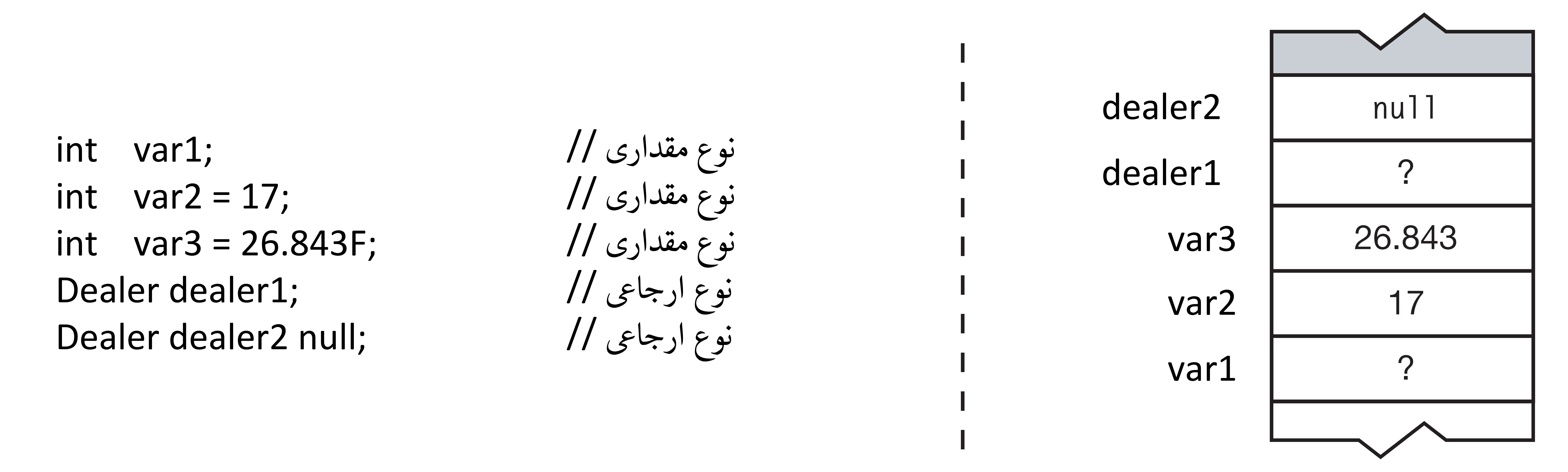
علاوه بر تعیین نام و نوع متغییر می توان به صورت اختیاری آن را با یک مقدار خاص، مقداردهی اولیه کرد.

مقداردهی اولیه متغییر از یک علامت مساوی و به دنبال آن، مقدار اولیه آن ( مقداری که هنگام تعریف متغییر به آن تخصیص داده می شود) را شامل می شود.

|  |
| --- |
| مقدار اولیه  ↓  int var2 = 17; |

متغییر های محلی که بدون مقدار اولیه تعریف می شوند تا زمانی که آنها را مقداردهی نکنیم نمی توانیم از آنها استفاده کنیم. تلاش برای استفاده از متغییر های محلی تعریف نشده ، باعث می شود که کامپایلر پیغام خطا تولید کند.

شکل 12-3 در سمت چپ تعدادی از متغییر های محلی تعریف شده و در سمت راست پیکر بندی stack آنها را نشان می دهد. برخی از متغییر ها مقدار اولیه دارند و برخی دیگر ندارند.



شکل 12-3. مقدار دهی اولیه متغییر ها

مقداردهی اولیه به صورت خودکار

بعضی از انواع متغییر ها اگر بدون مقدار اولیه تعریف شوند به صورت خودکار با یک مقدار پیش فرض مقداردهی می شوند. متغییر هایی که به صورت خودکار مقداردهی اولیه نمی شوند، تا زمانی که در برنامه به آنها مقداری اختصاص نگیرد دارای مقدار تعریف نشده هستند. جدول 5-3 انواعی از متغییر ها را نشان می دهد؛ و بررسی می کند که آیا به صورت خودکار مقداردهی اولیه می شوند یا خیر.

جدول 5-3. خلاصه ای از ذخیره سازی متغییر ها

|  |  |  |  |
| --- | --- | --- | --- |
| نوع متغییر | ذخیره می شود در | مقداردهی اولیه به صورت خودکار | استفاده می شود برای |
| متغییر محلی | Stack یا stack وheap | ندارد | محاسبه های محلی در داخل تابع |
| فیلد های کلاس | Heap | دارد | اعضای یک کلاس |
| فیلد های ساختار | Stack یا heap | دارد | اعضای یک ساختار |
| پارامتر ها | Stack | ندارد | ارسال مقادیر به داخل و خارج از متد |
| عناصر آرایه | Heap | دارد | اعضای یک آرایه |

تعریف متغییر های متعدد

می توان در یک دستور، متغییر های متعددی را تعریف کرد.

* همه متغییر های باید از یک نوع باشند.
* نام متغییر ها باید با کاما از یکدیگر جدا شوند. مقدار اولیه می تواند با نام متغییر همراه شود.

به عنوان مثال، کد زیر دو دستور معتبر از تعریف متغییر های متعدد را نشان می دهد. توجه کنید که می توان متغییر های مقداردهی شده را با متغییر های مقدار دهی نشده در یک دستور ایجاد کرد. برای این کار باید متغییر ها را با کاما از یک دیگر جدا کرد. آخرین دستور نشان داده شده نا معتبر است زیرا آن تلاش می کند که نوع های مختلفی از متغییر ها را در یک دستور تعریف کند.

|  |
| --- |
| // تعریف متغییر -- بعضی ها با مقدار اولیه و بعضی ها بدون مقدار اولیه  int var3 = 7, var4, var5 = 3;  double var6, var7 = 6.52;  نوع نوع متفاوت  ↓ ↓  int var8, float var9; /\* خطا ! نمی توان نوع های مختلف را در یک دستور به صورت مخلوط تعریف کرد  (int و float) \*/ |

استفاده از مقدار یک متغییر

نام یک متغییر مقدار ذخیره شده در آن متغییر را نشان می دهد. می توان با استفاده از نام متغییر از مقدار آن استفاده کرد.

به عنوان مثال، در دستور زیر مقدار var2 از حافظه بازیابی می شود و در موقعیت متغییر قرار داده می شود.

|  |
| --- |
| Console.WriteLine("{0}", var2); |

نوع های static و کلمه کلیدی dynamic

شما باید به یک مورد توجه داشته باشید، که تعریف کردن هر متغییری، نوعی از متغییر ها را شامل می شود که به کامپایلر اجازه می دهد، مقداری از حافظه و قسمت هایی که باید در stack و در heap ذخیره شوند را در زمان اجرا به آن اختصاص دهد. نوعی از متغییر ها که در زمان کامپایل حافظه آنها تعیین شده اند و در زمان اجرا قابل تغییر نیستند. این نوع های را نوع های static می نامند.

بعضی از زبان ها دارای نوع های static هستند. بسیاری از زبان های اسکریپت نویسی از جمله IronPython و IronPuby دارای نوع های پویا[[4]](#footnote-4) هستند. به عبارت دیگر، نوع یک متغییر ممکن است تا زمان اجرا تعیین نشود. از آنجایی که این زبان ها، تحت دات نت هستند ، برنامه ی C# قادر است که از اسمبلی های نوشته شده این زبان ها استفاده کند. مشکل این است، که C# باید در زمان کامپایل قادر باشد که نوع های اسمبلی را تجزیه و تحلیل کند، در حالی که این نوع ها تا زمان اجرا تجزیه و تحلیل نمی شوند.

برای حل این مشکل، C# کلمه کلیدی dynamic را فراهم کرده است، که یک نوع C# را مشخص می کند تا در زمان اجرا آنرا تجزیه و تحلیل کند.

در زمان کامپایل، کامپایلر متغییر های نوع dynamic را بررسی نمی کند. در عوض، هر گونه اطلاعاتی، درباره عملیات آن متغییر را بسته بندی می کند. در زمان اجرا، برای اینکه متغییر تجزیه و تحلیل شود، این اطلاعات بررسی می شود تا مطمئن شود که با نوع واقعی مطابقت دارد. اگر مطابقت نداشته باشد، در زمان اجرا استثنا رخ می دهد.

نوع های تهی[[5]](#footnote-5)

شرایطی وجود دارد بخصوص زمان کار با پایگاه داده جایی که شما می خواهید نشان دهید که یک متغییر مقدار معتبری را در خود نگه داری می کند. برای نوع های ارجاعی، شما به راحتی می توانید با تخصیص دادن مقدار تهی به متغییر این کار را انجام دهید. هنگامی که یک متغییر از نوع مقداری را تعریف می کنید ، اگر حافظه به آن اختصاص داده شده باشد یا اگر حافظه به آن اختصاص داده نشده باشد، محتویات آن در هر دو صورت معتبر است.

چگونه شما می توانید مقدار نوع بولین را با این وضعیت ارتباط دهید بنابراین زمانی که مقدار معتبر است مقدار آن true و زمانی که مقدار معتبر نیست مقدار آن false است.

نوع های تهی به شما اجازه می دهد که یک نوع را با مقدار معتبر ایجاد کنید، که می تواند معتبر یا نا معتبر بودن را مشخص کند به طوری که شما می توانید از معتبر بودن یک متغییر قبل از استفاده از آن اطمینان حاصل کنید. نوع های مقداری به طور منظم نوع های غیر تهی می نامند. جزئیات نوع های تهی را در فصل 25 بررسی خواهیم کرد تا آن زمان C# را بهتر درک خواهید کرد.

1. پشته [↑](#footnote-ref-1)
2. آخرین ورودی، اول از همه خارج می شود. [↑](#footnote-ref-2)
3. پشته [↑](#footnote-ref-3)
4. dynamic [↑](#footnote-ref-4)
5. Nullable Type [↑](#footnote-ref-5)