جزوه درس

طراحی و پیاده سازی زبانهای برنامه سازی

تهیه کننده

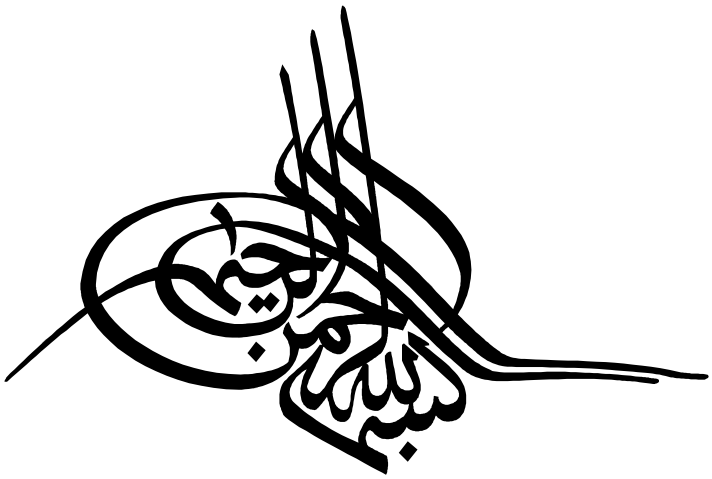
علی چوداری خسروشاهی

مرجع

اصول طراحی و پیاده سازی زبانهای برنامه سازی

تالیف :ترنسدبلیو .پرات – مارون وای .زیلکوویتز

ویژه دانشجویان کارشناسی کامپیوتر – نرم افزار



بنام خدا

پیشگفتار

این جزوه تلاش شده است تمام سرفصلهای درس پوشش داده شود. در مورد این جزوه یادآوری این نکته مهم، ضروری است که این جزوه جهت کمک به دانشجویان در کاهش یادداشت برداری، ترسیم شکلها، مثالها در هنگام تدریس بوده است. بنابر این در کنار این جزوه، هر دانشجو مطابق با ذوق و سلیقه خود، جزوه ای دست نویس خواهد داشت که حاوی یادداشت ها به منظور تکمیل و تفهیم این جزوه است.

این جزوه، ادعای جایگزینی مراجع اصلی این درس را ندارد بلکه تنها خلاصه ای از مطالب مهم از مراجع درس است. د ر گردآوری این جزوه از کتاب " اصول طراحی و پیاده سازی زبانهای برنامه سازی" تالیف ترنسدبلیو .پرات و مارون وای زیلکوویتز استفاده شده است.

در اینجا لازم است از تلاش تمام دانشجویانی که اینجانب را در تدوین این جزوه یاری نموده اند کمال تشکر را داشته باشم. همچنین در این جزوه احتمال اشتباه وجود دارد، به این جهت در مطالعه مطالب جزوه دقت کافی را داشته باشید.

از دانشجویان عزیز تقاضا می‌شود در مطالعه مطالب آن دقت کافی را داشته باشند و وجود هرگونه ایراد ، و همچنین نظرات و پیشنهادات خود را به آدرس پست الکترونیکی اینجانب اطلاع دهند.

Akhosroshahi@IAUT.ac.ir
Akhosroshahi@Gmail.com

باتشکر

علی چوداری خسروشاهی

فهرست مطالب

فصل اول

۱- نکاتی در مورد طراحی زبان

۱-۱- چرا زبانهای برنامه سازی را مطالعه می کنیم؟

- برای بهبود توانایی خود در جهت توسعه الگوریتم های کارآمد .
- استفاده بهینه از زبان برنامه نویسی موجود.
- می توانید با اصطلاحات مفید ساختار های برنامه نویسی آشنا شوید.
- انتخاب بهترین زبان برنامه سازی .
- آموزش زبان جدید ساده می شود .
- طراحی زبان جدید ساده می شود.

۱-۲- صفات یک زبان خوب :

وضوح ، سادگی ، یکپارچگی : هر زبان یک چهار چوبی را برای فکر کردن راجع به الگوریتم ها برای برنامه نویس تدارک می بیند ، باید این زبان مفاهیم واضح ، ساده و یکپارچه را برای برنامه نویس تدارک ببیند تا اینکه برای طراحی الگوریتم استفاده شود ، پس می توان گفت قابلیت خوانایی برای نحو یک زبان syntax یک اصل اساسی محسوب می شود.

قابلیت تعامد (Orthogonality) : منظور از تعامد این است که بتوانیم ویژگی های مختلف یک زبان را با همدیگر ترکیب کنیم و ترکیب حاصل با معنی باشد در این حالت ، حالت های استثنا و موارد خاص کاهش پیدا می کند.

طبیعی بودن برای کاربرد ها : هر زبانی که داری نحو(syntax) نز دیک به الگوریتم باشد یعنی اجازه دهد که برنامه ساختار منطقی مر بوط به الگوریتم را منعکس کند مناسب تر خواهد بود.

پشتیبانی از انتزاع (Abstraction): یعنی اینکه بتوانیم از اعمال موجود در یک زبان برای نوع های داده ای جدید استفاده کنیم، ایده آل می باشد که یک زبان به کاربر اجازه دهد که کاربر بتواند نوع های داده ای جدید را تعریف نموده و اعمال تو کار سیستم را برای آنها استفاده کند.

سهولت در بازرسی برنامه: یکی از مواردی که در برنامه نویسی اتفاق می افتد بازرسی برنامه نوشته شده می باشد تااینکه ببینیم آیا نتیجه خواسته شده ما را می دهد یا نه؟

زبانی مناسب می باشد که قابلیت بازرسی راحتی را داشته باشد.

محیط برنامه نویسی: یک محیط برنامه نویسی قدرتمند برای یک زبان یک مزیت محسوب می شود ممکن است که زبان ساختار قوی داشته باشد اما این زبان اگر محیط برنامه نویسی قوی نداشته باشد کمتر انتخاب می شود.

قابلیت حمل برنامه: برنامه ای که به سیستم خاصی وابسته نباشد قابلیت حمل را خواهد داشت و مناسب تر می باشد.

هزینه استفاده: هزینه استفاده در سه گروه مطرح می شود.

- *هزینه اجرای برنامه*
- *هزینه ترجمه برنامه*

- *هزینه نگهداری* :که شامل هزینه ترمیم خطا ، هزینه تغییر نیازمندی ها ،و غیره می باشد که در طول عمر برنامه به کثرات اتفاق می افتد،برخی از کامپایلر ها برای کارهای دانشگاهی طراحی می شوند در اینگونه کامپایلرها هزینه ترجمه برنامه فاکتور مهمی می باشد چون اینگونه برنامه های دانشجویی بیشتر از اینکه اجرا شوند کامپایل می شوند اما زبان هایی که برای کاربردهای عملی طراحی می شوند چون تعداد دفعات اجرا زیاد می باشد باید هزینه اجرایی کمتری داشته باشد.

۱-۳- نحو و معنای زبان

syntax یک زبان برنامه نویسی، ساختار ظاهری آن زبان را مشخص می کند، توسط syntax مشخص می شود که دستورات ، اعلان ها و سایر ساختارهای زبان چگونه نوشته شوند اما این ساختارها ی نوشته شده یکه معنا و مفهوم را خواهند داشت (semantic معنای زبان)

۱-۴- مدل های زبان

چهار نوع مدل زبان وجود دارد که عبارتند از:

- زبانهای دستوری(imperative)
- زبانهای تابعی(applicative)
- زبانهای قانونمند(rule based)
- برنامه نویسی شی گرا(object oriented)

۱-۴-۱- زبا نهای دستوری

زبانهای مبتنی بر فرمان یا دستور گرا می باشند دستورات به ترتیب قرار گرفته و اجرا ی هر دستور ، وضعیت زبان را تغییر خواهد داد ، این تغییرات دنبال می شود تا نتیجه خواسته شده بدست آید. مانند زبان c ,pascal

۱-۴-۲- زبان های تابعی

در اینگونه زبانها بجای مشاهده تغییر حالت ، عملکرد برنامه دنبال می شود برای این منظور بجای در نظر گرفتن داده های موجود نتیجه مطلوب را در نظر می گیریم برخی توابع اولیه وجود دارد که این توابع ترکیب شده تا توابع جدیدی ایجاد شود مانند زبان LISP,SCHEME

۱-۴-۳- زبانهای قانونمند

در اینگونه زبانها شرایط بررسی می شود و در صورت بر قرار بودن شرط فعالیت های خاصی انجام می شود ، این زبان ها شبیه زبان های دستوری می باشد با این تفاوت که در این زبان ها ترتیب وجود ندارد مانند PROLOG

۱-۴-۴- زبان های برنامه نویسی شی گرا

در اینگونه زبان ها، اشیائ داده ای پیچیده تولید می شود و مجموعه ای از توابع برای کارکردن کاربر طراحی می شوند هرشی می تواند برای تولید اشیائ دیگر نیز استفاده شود . به عنوان مثال C++,JAVA نمونه ای از این زبان ها می باشند.

۱-۵- استاندارد سازی زبان

هنگام مشاهد یک دستور در یک زبان برای پی بردن به معنای دستور و خروجی آن سه راه حل وجود دارد:

۱. مراجعه به مستندات زبان
۲. تایپ و اجرا روی کامپیوتر
۳. مراجعه به استاندارد زبان ها

استاندارد ها معمولا دو دسته می باشند :

۱. استاندارد خصوصی
۲. استاندارد عمومی

استاندارد خصوصی: استانداردی می باشد که توسط خود شرکت طراحی کننده زبان ارائه می شود در این حالت هیچ الزامی برای تبعیت کردن سایر شرکت ها از این استاندارد وجود ندارد.

استاندارد عمومی: استانداردی می باشد که توسط سازمان های مختلفی ارائه می شود و کلیه پیاده سازی های زبان باید از آن باید تبعیت کند.برای استفاده از استاندارد سه مسئله مهم وجود دارد:

۱- **زمان شناسی (timelines)** :یعنی چه زمانی، زبان را استاندارد کنیم قبل از ارائه زبان ، در اوایل ارائه زبان و مدت ها بعد از ارائه زبان که در حالت اول مطلوب تر می باشد.

۲- **اطاعت و پیروی (Comformance)**: یعنی اینکه برنامه ها باید از استاندارد پیروی کنند کامپایلر پیرو ،کامپایلری می باشد که وقتی یک برنامه استاندارد به آن ارائه شود نتیجه مشخصی را (درست)می دهد . ممکن است زبان ها امکانات اضافی علاوه بر استاندارد داشته باشد که هیچ نتیجه ای برای آن مشخص نمی شود.

۳- **کهنگی (obsolescence)**: ممکن است با گذشت زمان برخی از مفاهیم یک زبان کاربرد خود را از دست بدهند(کهنه شوند) استاندارد های جدیدی که برای یک زبان ارائه می شود باید از استاندارد های قبلی خود پشتیبانی کند، اخیراً مفهوم کهنگی و مستهلک شدن در استاندارد ها استفاده می شود و به این مفهوم می باشد که آن عبارت به احتمال زیاد تا ۱۰-۲۰ سال آینده از زبان حذف خواهد شد.

فصل دوم

۲- اثرات معماری کامپیوتر

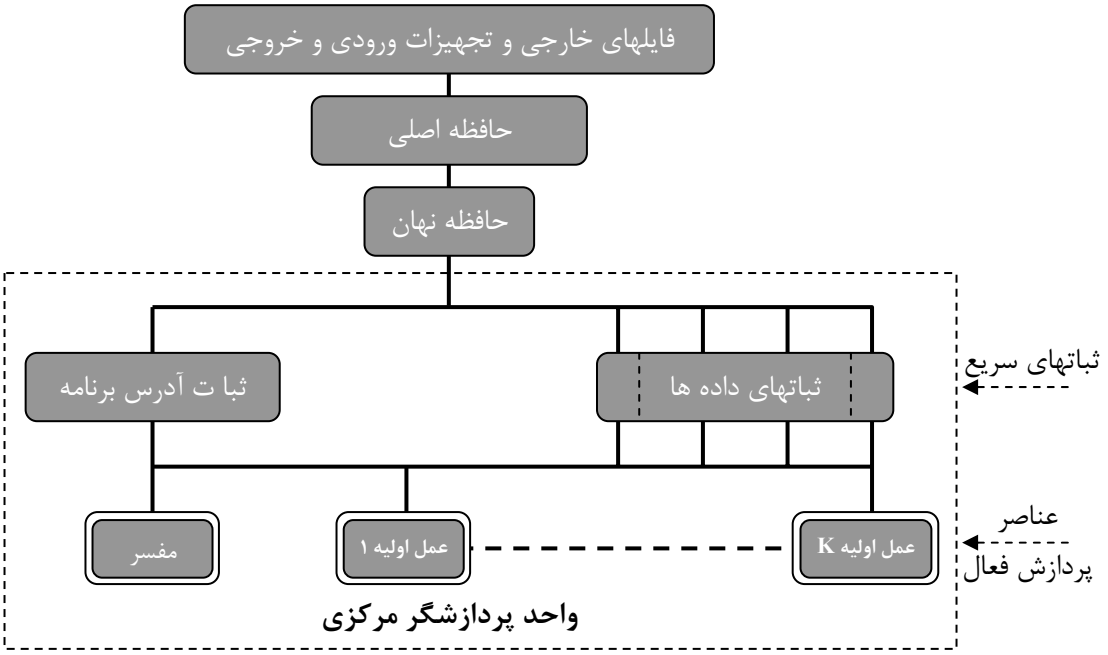
۲-۱- علملکرد کامپیوتر

کامپیوتر مجموعه ای از الگوریتم ها و ساختمان داده ها است که قابلیت ذخیره سازی و اجرای برنامه را دارد.هر کامپیوتر از ۶ جزء زیر تشکیل شده است:

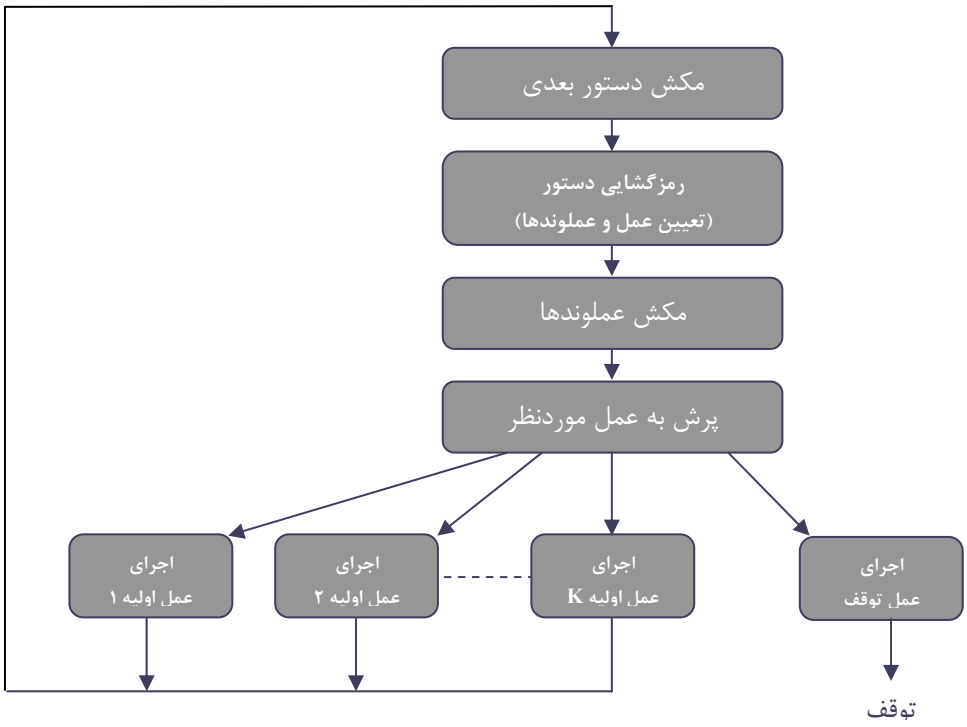
- **داده ها(data)** :هر کامپیوتر باید انواع مختلفی از عناصر داده ای و ساختمان داده ها را در اختیار کاربر قرار دهد.
- **اعمال اولیه(primitive operation)**:هر کامپیوتر باید یک سری اعمال اولیه را به منظور دستکاری داده ها در اختیار کاربر قرار دهد.
- **کنترل ترتیب(sequence control):** هر کامپیوتر باید مکانیزمی را به منظور کنترل ترتیب اجرای دستورات داشته باشد.
- **دستیابی به داده ها(data control)**: هر کامپیوتر باید مکانیزمی برای کنترل دسترسی به داده های تولید شده توسط دستورات داشته باشد.
- **مدیریت حافظه(Storage management)**: هر کامپیوتر باید مکانیزمی برای کنترل تخصیص حافظه به برنامه ها داشته باشد.
- **محیط عملیاتی (Operating environment):** هر کامپیوتر باید مکانیزمی رابرای برقراری ارتباط با محیط خارج داشته باشد(امکانات ورودی خروجی)

۲-۲- سخت افزار کامپیوتر

واحد پردازشگر مرکزی(Cpu): از بخش های یک کامپیوتر می باشد این واحد از ثبات های سریع و عناصر پردازش فعال تشکیل شده است. ثبات هایی که وجود دارند، ثبات های آدرس و ثبات های داده ای می باشند، ثبات های آدرس برای آدرس دهی کردن داده ها و دستورات روی حافظه استفاده می شوند و ثبات های داده ای هم برای ذخیره سازی داده های مورد نیاز و نتایج حاصل شده از اعمال اولیه استفاده می شوند. هر دستور العمل روی حافظه اصلی ،مشخص کننده یک هدف می باشد که این عمل توسط مفسر cpu ترجمه شده (عملیات کد گشایی) و دستورات لازم به بخش های مختلف داده می شود تا اینکه عمل اولیه بر روی داده ها انجام شود. عناصر پردازش یک cpu تشکیل شده از اعمال اولیه ای برای آن مشخص شده است این اعمال اولیه ممکن است در پردازشگرهای مختلف باشد.



سازمان یک کامپیوتر معمولی



تفسیر و اجرای برنامه

۲-۲-۱- کامپیوترهای سخت افزار (hardware)

کامپیوتر سخت افزار کامپیوتری است که کاملاً از اجزاء سخت افزاری و مدارات الکتریکی ساخته شده است. وضعیت ابتدایی برنامه مطابق با initial state کامپیوتر و نتیجه نهایی برنامه دقیقاً final state کامپیوتر است.

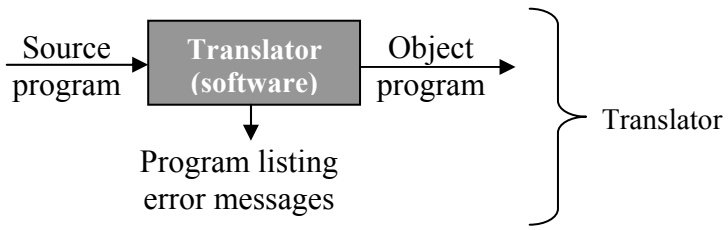
۲-۲-۲- کامپیوترهای میان افزار (firmware)

کامپیوتر میان افزار توسط ریز برنامه ای (microprogram) شبیه سازی می شود که بر روی کامپیوتر سخت افزار قابل ریزبرنامه نویسی اجرا می گردد. زبان ماشین آن مجموعه بسیار سطح پایین از ریز دستورات است که انتقال داده ها را بین حافظه اصلی و ثباتها، بین خود ثباتها و از ثباتها، از طریق پردازنده ها انجام می دهد.

۲-۲-۳- مفسرها و معماری های مجازی

زمانیکه یک برنامه به یک زبان نوشته می شود سوالی که ایجاد می شود این است که این برنامه زبان سطح بالا چگونه در یک کامپیوتر واقعی صرف نظر از زبان ماشین آن اجرا می شود برای این منظور دو راه حل وجود دارد:

۱- **ترجمه یا کامپایل کردن**: می توان مفسری را طراحی نمود که یک برنامه زبان سطح بالا را دریافت کرده و به یک زبان ماشین ترجمه کند.مفسر برنامه یک زبان سطح بالا یا پایین را به عنوان ورودی دریافت نموده و به برنامه مقصد تبدیل می کند تحت شرایطی که کارایی آنها باهم برابر می باشد.

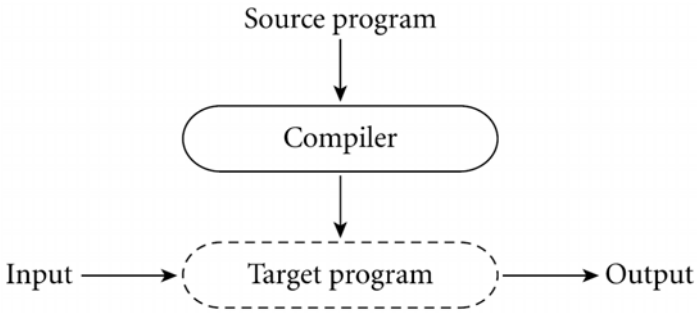


انواع مفسرها عبارتند از:

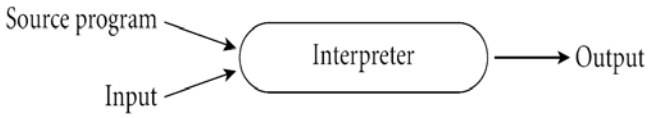
- *اسمبلر(assembler)* : مفسری می باشد که زبان منبع آن برنامه زبان اسمبلی و زبان مقصد آن زبان ماشین برای برنامه واقعی می باشد.
- *کامپایلر (compiler)* : مفسری می باشد که زبان منبع آن یک زبان سطح بالا و زبان مقصد آن ، زبان نزدیک به زبان ماشین می باشد.
- *بارکننده (loader)* : مفسری می باشد که زبان مقصد آن زبان کد ماشینی واقعی و زبان منبع آن مانند ورودی است و معمولا شامل برنامه های زبان ماشین به شکل جابجا پذیر می باشد.
- *پیوند دهنده(Linker)*: این مفسر بخش های مختلف برنامه را دریافت نموده آنها را سر هم بندی کرده و برنامه خروجی تقریبا شبیه برنامه ورودی به شکل کامل تر تولید میکند.
- *پیش پردازنده یا پردازنده ماکرو(preprocessor)* : مفسری می باشد که زبان منبع آن شکل توسعه یافته ای از یک زبان سطح بالا مانند C++ می باشد و زبان مقصد آن شکل استانداری از همان ربان می باشد (همان برنامه C)

۲. **شبیه ساز نرم افزاری (تفسیر نرم افزاری)**[1] : در این روش به جای ترجمه برنامه سطح بالا به برنامه زبان ماشین معادل ، می توانیم از شبیه ساز استفاده کنیم که از این طریق روی کامپیوتر میزبان اجرا می شود این شبیه سازی با نرم افزار که روی کامپیوتر میزبان اجرا می شود ، انجام می شود این نرم افزار یک کامپیوتر زبان سطح بالا را ایجاد می کند در غیر اینصورت مجبور هستیم از سخت افزار واقعی یعنی ماشینی که زبان آن سطح بالا باشد استفاده کنیم این شبیه ساز مستقیما برنامه گرفته شده را اجرا می کند.

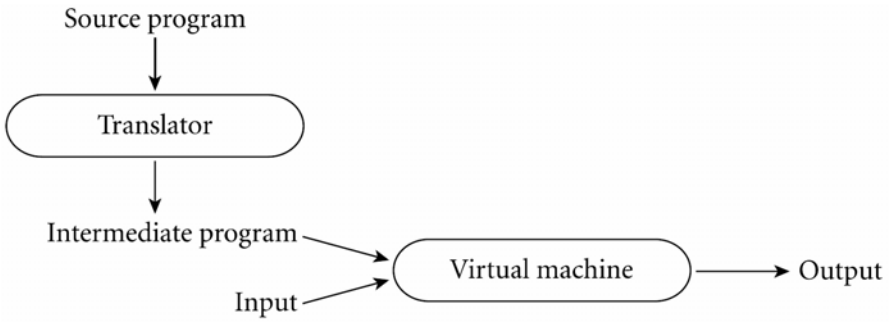
معمولا از روش ترکیبی نیز استفاده می شود یعنی برنامه سطح بالا با استفاده از یک مترجم به یک برنامه ای میانی تبدیل می شود که این برنامه میانی می تواند با استفاده از یک ماشین مجازی روی کامپیوتر حقیقی اجرا شود.



کامپایلر



تفسیر نرم افزاری



روش ترکیبی

[1]Software simulation (Software interpreter)

استفاده از مفسر باعث می شود برنامه نهایی تولیده شده سرعت اجرای بالایی نسبت به شبیه سازی داشته باشد اما مزیت شبیه سازی این است که برنامه قابلیت اجرا بر روی هر ماشینی را خواهد داشت (وابسته به ماشین نیست) اما در حالی که برنامه تولید شده توسط مفسر فقط روی ماشینی اجرا خواهد شود که برنامه به زبان آن ماشین تولید شده است.

روش های ساخت کامپیوتر عبارتند از :

- *از طریق سخت افزار(Through a hard ware realization)* :ساختمان داده ها و الگوریتم ها به صورت سخت افزاری پیاده سازی می شود.
- *از طریق میان افزار (Through firmware realization):* ساختمان داده ها و الگوریتم ها از طریق ریز برنامه نویسی برای سخت افزار مناسب ایجاد می شود.
- *ازطریق ماشین مجازی(Through saft ware realization):* در این حالت ساختمان داده ها و الگوریتم ها از طریق برنامه نویسی و زبان های دیگر نمایش داده می شوند.
- *از طریق ترکیبی (Through a hard ware realization):* در این تکنیک بخش های مختلف کامپیوتر مستقیما در سخت افزار و یا به وسیله شبیه سازی نرم افزاری نمایش داد می شود.

۲-۳- کامپیوترهای مجازی و پیاده سازی های زبان

سه عامل منجر بر تفاوتهایی در بین پیاده سازیهای یک زبان می شود:

- پیاده سازی های مختلف از کامپیوتر مجازی که به طور ضمنی در تعریف زبان وجود دارد، درک های متفاوتی اند.
- تفاوتهایی در امکاناتی که توسط کامپیوتر میزبان ارائه می شود که زبان برنامه سازی باید بر روی آن پیاده سازی شود.
- تفاوتها در انتخابهایی که توسط پیاده ساز صورت می گیرد تا عناصر کامپیوتر مجازی را با استفاده از امکاناتی که توسط کامپیوتر مربوط ارائه می شود پیاده سازی کند. علاوه بر این ساخت مترجم برای پشتیبانی از این انتخابهای نمایش کامپیوتر مجازی

۲-۴- سلسه مراتب ماشین های مجازی :

کامپیوترمجازی تعریف شده توسط برنامه نویس (پیاده سازی توسط زبان سطح بالا)
کامپیوترمجازی زبان سطح بالا (پیاده سازی توسط برنامه ها و اجرا توسط OS)
کامپیوترمجازی سیتم عامل (با برنامه های که بر روی کامپیوترمجازی یا میان افزاری اجرا می شوند پیاده سازی می گردد)
کامپیوترمجازی میان افراز (با دستورات زمان ماشین پیاده سازی شده که با ریز دستورات توسط کامپیوتر واقعی اجرا می شود)
کامپیوترسخت افزار واقعی (توسط اجرا فیزیکی پیاده سازی شده است)

شکل بالا سلسله مراتب ماشین های مجازی را نشان می دهد در پایین تر ین سطح آن کامپیوتر سخت افزار واقعی قرار می گیرد. در این شکل هر لایه توسط لایه پایین تر به اجرا در می آید .در صورت استفاده از مترجم ، لایه دوم بالایی وجود نخواهد داشت و یا به صورت ضعیف وجود خواهد داشت اما در صورت استفاده از شبیه سازی نرم افزاری همه لایه ها به صورت کامل وجود خواهند داشت.

۲-۵- انقیاد (binding) و زمان انقیاد

انقیاد یک عنصر برنامه به ویژگی یا صفت خاصه مثل انتخاب یک صفت از مجموعه ای از صفات است.زمانیکه در حین فرموله کردن یا پردازش برنامه این انتخاب صورت می گیرد زمان انقیاد آن صفت برای آن عنصر نام دارد زمان های انقیاد به گروه های زیر دسته بندی شده اند:

۱. **زمان اجرا**: در این زمان انقیاد های زیادی صورت می گیرد مانند انقیاد متغیرها به مقادیرشان ، انقیاد متغیر به محل های خاص حافظه این نوع انقیاد ها به دو دسته تقسیم بندی می شود :
   a) هنگام ورود به زیر برنامه یا بلوک
   b) در نقطه خاصی از اجرای برنامه
۲. **زمان ترجمه (زمان کامپایل)**: در این زمان سه دسته انقیاد وجود دارد :
   a) انقیادی که *توسط برنامه نویس* انتخاب می شود به عنوان مثال اسامی متغیر ها، نوع متغیر ها ...
   b) انقیادی که *توسط مترجم* انجام می شود مانند مشخص کردن محل نسبی شی داده ای در حافظه
   c) انقیادهای که *توسط بارکننده* صورت می گیرد مثل انقیاد متغیر ها به آدرس های آنها
۳. **زمان پیاده سازی زبان**:بعضی از جنبه های تعریف یک زبان ممکن است برای برنامه هایی که با استفاده از آن زبان اجرا می شوند یکسان باشند اما ممکن است مابین پیاده سازی ها متفاوت باشد اگر برنامه ای، بصورتی پیاده سازی شود که از انقیاد های زمان پیاده سازی استفاده کنیم ممکن است در پیاده سازی های دیگر اجرا نشود ویا اجرا شده اما نتیجه متفاوتی را بدهد.
۴. **زمان تعریف زبان** به عنوان مثال شکلهای مختلف دستورات،انواع ساختمان داده ها وغیره

به عنوان مثال فرض کنید برنامه x:=x+10 در زبان L نوشته شده است برای این برنامه انقیاد ها و زمان انقیاد عناصر به صورت زیر بحث می شود:

۱. مجموعه ای از انواع ممکن برای متغیر x : می توان دارای نوعی مانند real ، integer ، و یا Boolean این انقیاد در زمان تعریف در زبان مشخص می شود.
۲. نوع متغیر : نوع داده ای x در زمان ترجمه مشخص می شود و برای این منظور حتما باید متغیر اعلان شود .
۳. مجموعه ای از مقادیر ممکن برای x : این نوع binding در زمان پیاده سازی زبان اتفاق می افتد و ممکن است در پیاده سازی های مختلف متفاوت باشد.
۴. مقدار متغیر x : این نوع binding در زمان اجرا اتفاق می افتد با انتخاب مقدار به متغیر x صورت می گیرد.
۵. نمایش مقدار ثابت ۱۰: انتخاب نحوه نمایش مقدار عدد ۱۰ در زمان تعریف زمان انجام می شود علاوه بر این نحوه نمایش این مقدار به صورت دنباله ای از بیت ها در زمان پیاده سازی مشخص می شود .

۶. خواص عملکرد + : انتخاب نماد جمع برای نمایش عمل جمع در زمان تعریف زبان اتفاق می افتد .

در برخی از زبان ها مانند فرترن انقیاد در زمان ترجمه برنامه اتفاق می افتد به این گونه اتقیاد ***انقیاد زود رس*** گفته می شود و برخی دیگر از زبان ها مانند ML انقیاد در زمان اجرا صورت می گیرد به این گونه انقیاد ***انقیاد دیررس*** گفته می شود بسیاری از تفاوت ها ی زبان ها به زمان های انقیاد مربوط می شود. انقیاد زودرس باعث سریعتر شدن اجرای برنامه تولیدی می شود اما در حالی که انقیاد دیررس باعث ایجاد زبان منعطف تر می شود دیررس بودن ویا زودرس بودن باعث تقابل مابین سرعت در انعطاف پذیری خواهد شود که این دو مقوله با هم دیگر در تضاد خواهد بود.

فصل سوم

۳- انواع داده اولیه(Elementary Data Type E.D.T)

۳-۱- خواص انواع و اشیاء

هر برنامه صرف نظر از نوع زبان مجموعه ای از عملیات است که باید به ترتیب خاصی بر روی داده ها اجـرا شـوند. تفاوتهـای بین زبانها ناشی از انواع *داده ها*، *عملیات موجود* و مکانیزم *کنترل ترتیب* اجرای عملیات بر روی داده ها است.

۳-۱-۱- اشیای داده ، متغیرها و ثوابت

ناحیه ی ذخیره سازی داده بر روی کامپیوتر حقیقی،ساده می باشد(یک بایت،دو بایـت و...) امـا ذخیـره سـازی داده بـر روی کامپیوتر مجازی،پیچیده می باشد(مانند آرایه،صف و پشته و...) از اصطلاح Data object برای گروه بندی زمـان اجـرای یـک یـا چند قطعه از داده ها در کامپیوتر مجازی استفاده می کنیم.

اشیاء داده ی میتوانند Programmer defined یا System defined باشند. بعضی اشیا در حین اجرای برنامه توسط برنامه نویس تعریف شده اند. اشیائ داده ای که توسط برنامه نویس تعریف می شوند در حین اجرای برنامه وجود دارند و اینگونه اشـیاء قابل دسترس برای برنامه نویس هستند. مثل متغیرها، مقادیر ثابت، آرایه ها و فایلها.

System defined اشیاء داده ای هستند که توسط سیستم تعریف می شوند در حین اجرای برنامه به صورت خودکار، ایجـاد می شوند مانند پشته های زمان اجرا.و یا با فر فایلها، اینگونه اشیای داده ای،مستقیما در اختیار برنامه نویس قرار نمی گیرند.

توصیف اشیاء داده ای:

- یک شئ داده، **ظرفی**[1] برای مقادیر داده است. یعنی محلی که داده در آن ذخیره و از آن بازیابی می شوند.
- یک شئ داده، توسط مجموعه ای از **صفات**[2] مشخص می شودکه مهمترین آن نوع داده ای می شود.
- مقدار داده می تواند Single number، Character، Pointer یا ... باشد.
- یک شئ داده، توسط **الگوی**[3] خاصی از بیتها مشخص می شود. باید توجه داشت که اشیای داده و مقـادیر داده، بـا همدیگر اشتباه در نظر گرفته نشود.

اشیای داده ای مشخص کننده ی محلی از حافظه می باشد در حالیکه مقدار داده به صورت الگویی از بیتها مشخص می شود. مثال: فرض کنید یک شی داده ای با نام a، محلی از حافظه می باشد و همچنین فرض کنیـد یـک مقـدار داده برابر با عدد۱۴،با الگوی بیتی ۱۱۱۰وجود داشته باشد، هنگام bind کردن (مقید کردن) این مقدار بر روی شـئ داده ای به صورت زیر نمایش داده میشود.

A: 00001110

- هر شئ داده ای دارای **طول عمر :** است یعنی اینکه کی درحافظه ایجاد شده و کی ا ز حافظه حذف می شود.
- اگر شئ داده حاوی مقداری باشد که همیشه به عنوان یک واحد[1] دستکاری شود آن را **شئ داده اولیه** گویند.

[1] container
[2] Attribute
[3] Pattern

- اگر شی داده مجموعه ای از سایر اشیای داده باشد **ساختمان**[2] نامیده می شود.

انواع binding برای یک شئ داده:

یک شئ داده ای در طول عمر خود binding های مختلفی را می پذیرد. انواع binding هایی که برای یک شئ داده ای می تواند اتفاق بیفتد عبارت اند از:

- **نوع:** این پیوند در زمان ترجمه ی برنامه ایجاد می شود.
- **محل (location)**: انقیاد محل ذخیره سازی در حافظه توسط روال مدیریت حافظه ی کامپیوتر مجازی مشخص می شود.
- **مقدار**: این انقیاد (bind)نتیجه عمل انتساب می باشد.
- **نام (name):** انقیاد به یک یا چند نام که به وسیله ی آن به شیء داده ای در حین اجرای برنامه مراجعه می شود، معمولا توسط اعلانها مشخص و توسط فراخوانی زیر برنامه و برگشت از آن اصلاح میشود.
- **اجزاء (component):** Binding یک data object به یک یا چند data objectکه به عنوان اجزای آن است، توسط مقدار اشاره گر داده میشود و با تغییر اشاره گر اصلاح میشود.

۳-۱-۲- متغیرها و ثوابت

شی داده ای که توسط برنامه نویس تعریف و نامگذاری می شود متغیر (variable) نام دارد. شئ داده ای اولیه، متغیر ساده ای می باشد که مقدار متغیر می تواند در طول اجرای برنامه با استفاده از دستور انتساب تغییر پیدا کند.

ثابت (constant) یک شی داده با نام است که مقداری به آن نسبت داده می شود و مقدار آن در طول اجرای برنامه تغییر پیدا نمی کند.

ثابتها دو نوع می باشند:

۱. یک ثابت لیترال (literal constant) ثابتی است که نامش همان نمایش مقدارش است .
۲. ثابت تعریف شده توسط برنامه نویس ثابتی است که نامش در تعریف شئ داده توسط برنامه نویس انتخاب می شود.

۳-۱-۳- انواع داده (Data Type)

نوع داده طبقه ای از اشیای داده به همراه مجموعه ای از عملیات (manipulation) برای ایجاد و دستکاری آنها است. زبان برنامه سازی الزاماً با انواع داده هایی مثل دسته ای از آرایه ها ، مقادیر صحیح ، یا فایلها و عملیات مربوط به دستکاری آرایه ها ، مقادیر صحیح یا فایلها سروکار دارد.

هر زبانی مجموعه ای از *انواع دادهای اولیه* دارد که توسط زبان ساخته می شود و همچنین ممکن است زبان اجازه تعریف انواع دادهای جدید را به کاربر بدهد. یک نوع داده ای در دو سطح مختلف بررسی می شود:

۱. مشخصات (Specification)
۲. پیاده سازی (Implementation)

[1] unit
[2] struct

مشخصات (specification)

عناصر اصلی یک نوع داده درسطح Specification :

۱. صفات (Attributes) اشیای داده ای از آن نوع را متمایز می کند.
مثال آرایه: اندازه بعد آرایه – بازه اندیسهای آرایه – نوع داده ای عناصر آرایه
۲. مقادیری (Values) که داده ای از آن نوع می تواند داشته باشد .
مثال آرایه: (بستگی به نوع آرایه دارد) مجموعه ای از مقادیر معتبر برای عناصر آرایه می باشد .
۳. عملیاتی (Operations)که روش دستکاری آن شی داده ای را تعریف می کند .
مثال آرایه: الحاق – جمع و دسترسی به آرایه واندیس گذاری آرایه برای مراجعه به عناصر ، ایجاد آرایه – حذف آرایه

پیاده سازی (implementation)

عناصر اصلی یک نوع داده ای جهت پیاده سازی:

۱. نمایش حافظه ای
۲. روش دستکاری

۳-۱-۳-۱- مشخصات انواع داده اولیه

شئ داده اولیه حاوی یک مقدار است. دسته ای از چنین اشیائ داده که عملیات گوناگون بر روی آنها تعریف شده اند، **نوع داده اولیه** نامیده می شود. انواع اولیه متداول در زبان ها **عددی ، بولی ، کاراکتری، رشته های کاراکتری، شمارشی، زیر بازه** و **اشاره گر** است. انواع عددی نیز معمولاً شامل **صحیح ، ممیز شناور، موهومی** و **گویا** است.

صفات صفات اصلی هر شئ داده، مثل نوع داده و نام، معمولاً در طول عمر آن عوض نمی شود. توجه کنید که مقادیر یک صفت از شئ داده، متفاوت از مقادیر است که شئ داده حاوی آن است.

مقادیر انواع اشیا داده، مجموعه ای از مقادیر ممکن را برای آن اشیاء تعیین می کند. هر شئ داده اولیه، در هر نقطه از طول عمرش، یک مقدار از این مجموعه را دارا است.

مجموعه ای از مقادیر که توسط نوع داده اولیه تعریف می شود، معمولاً مجموعه مرتب می نامند، زیرا دارای بیشترین و کمترین مقدار است و برای هر دو مقدار، یکی کوچکتر و دیگری بزرگتر است.

عملیات مجموعه ای از عملیاتی که برای یک نوع داده ای تعریف شده است، تعیین می کند که آن نوع اشیاء داده چگونه باید دستکاری شوند. دو تقسیم بندی برای عملیاتها وجود دارد:

۱. عملیات اولیه (Primitive operation)
۲. Programmer defined operation

عملیات اولیه: از دید برنامه نویس به دو دسته تقسیم می شود

a) عملیات دودویی (Binary operation)که شامل دو عملوند می باشند.
b) عملیات یکانی (Unary operation)که شامل فقط یک عملوند می باشند.

Programmer defined operation : این عملیات به صورت زیر برنامه ها، پیاده سازی می شوند.

هر عملیات یک تابع ریاضی می باشد یعنی به ازای هر آرگومان ورودی، نتیجه ی مشخصی دارد. هر عملیات دارای یک دامنه و یک برد می باشد. دامنه مجموعه ای از مقادیر ورودی می باشد و برد مجموعه ای از نتایج ممکن می باشد. معمولاً برای نمایش مشخصات عملیات از نشانه گذاریهای ریاضی استفاده می شود

op name : arg type × arg type × … arg type → result type

چهار عامل موجب می شوند تا تعریف عملیات زبان برنامه سازی پیچیده شود:

۱. **عملیاتی که برای ورودیهای خاصی تعریف نشده اند.** ممکن است عملیاتی که برای یک دامنه تعریف شده برای ورودیهای خاص از آن دامنه تعریف نشده باشد. مانند عملیاتی که موجب ایجاد سرریز over flowو پاریز onderflowشود.

۲. **آرگومانهای ضمنی (implicit arguments)** : یک عملیات ممکن است برروی متغیرهای عمومی یا ارجاع به سایر شناسه های غیر محلی داشته باشد(آرگومانهای ضمنی) در این شرایط تعیین تمام داده هایی که تحت تاثیر این عملیات قرار می گیرند مشکل می باشد.

۳. **اثرات جانبی (نتایج ضمنی):** یک عملیات ممکن است علاوه بر وظایف اصلی خود وظایف دیگری را نیز انجام دهد، یا علاوه بر مقدار بازگشتی، آرگومانهای ورودی خود را نیز تصحیح کند. در چنین مواردی ،تعیین مشخصات برد، دشوار می باشد.

```
int sum(int &a, int b);
{
     return ++a + b++;
}
#include
int main()
{
    int a = 5, b = 10;
    cout << sum(a,b) << endl;
    cout << "a = " << a << ", b = " << b;
    return 0;
}
```

۴. **خود اصلاحی (Self-modification)** یک عملیات ممکن است حساس به سابقه اجرای خود باشد.یعنی در هر بار اجرا، خود را اصلاح کند.بنابراین نتایج حاصل از این گونه عملیات ها ممکن است در هر بار اجرای آن متفاوت باشد. نمونه مثالی از این عملیات ها، مولدعدد تصادفی می باشد این عملیات پارامتر ثابتی را می گیرد ولی نتایج متفاوتی را تولید می کند.

۳-۱-۳-۲- تعریف subtype وsupertype

اگر نوعی به عنوان بخشی از نوع بزرگتر باشد آن را زیر نوع (subtype) و نوع بزرگتر را ابر نوع (supertype) می گویند.

۳-۱-۳-۳- پیاده سازی انواع داده اولیه

پیاده سازی نوع داده اولیه ، شامل *نمایش حافظه* مربوط به اشیای داده و مقادیر آن نوع و همچنین مجموعه ای از *الگوریتم ها* یا *رویه ها* برای دستکاری حافظه ی آن می باشد.

نمایش حافظه حافظه مربوط به انواع داده اولیه تحت تاثیر کامپیوتری است که برنامه را اجرا می کند.

با صفات اشیای داده اولیه به طور مشابه برخورد می شود:

۱. برای کارایی، بعضی از زبانها طوری طراحی شده اند که صفات داده ها توسط کامپایلر تعیین شوند.
۲. صفات شی داده ممکن است در زمان اجرا دریک توصیف گر [۱] (descriptor) و به عنوان بخشی از شی داده ذخیره شود. چون سخت افزار نمایش حافظه ای را برای توصیفگر تدارک نمی بیند، خود توصیفگر و عملیات بر روی اشیای داده ای با توصیفگر، به طور نرم افزاری شبیه سازی می شوند. این کار در زبان هایی مثل LISP و Prolog انجام می شود که قابلیت انعطاف آن ها بالا است.

پیاده سازی عملیات هر عملیاتی که برای نوعی از اشیای داده تعریف شد ممکن است به یکی از سه روش زیر پیاده سازی شود:

۱. ***به صورت عملیات سخت افزاری***. به عنوان مثال جمع و تفریق به صورت سخت افزاری
۲. ***به صورت زیر برنامه رویه یا تابع***. اگر اشیای داده ای با استفاده از نمایش سخت افزاری نشان داده نشوند، باید عملیات به صورت نرم افزار نمایش داده شود. به عنوان مثال جذر گیری.
۳. ***به صورت دستوراتی در داخل برنامه نوشته شوند***. این روش نیز شبیه روش دوم می باشد. اما به جای استفاده زیر برنامه ،دستورات زیر برنامه داخل برنامه نوشته می شوند. به عنوان مثال:

Abs(x)=if x<0 then –x else x

۳-۱-۴- اعلانها [۲]

دستوری از برنامه است که نام و نوع اشیای داده و همچنین طول عمر آن را در حین اجرای برنامه مورد نیاز هستند مشخص می کند.

اعلان می تواند به یکی از دو صورت انجام شود

۱. **صریح (explicit)** در این حالت صراحتا نام و نوع شئ داده ای معرفی می گردد.
۲. **ضمنی (implicit)** در این حالت صراحتا اعلان صورت نمی گیرد بلکه از پیش فرضهای سیستم استفاده می شود به عنوان مثال در فرترن متغیری مانندindexبدون اعلان صریح می توان استفاده کرد و نوع آن صحیح در نظر گرفته می شود.

اطلاعاتی که توسط دستورهای اعلان آورده می شود :

۱. Nameنام

[۱] مجموعه ای از صفات یک شیء داده که آن را توصیف می کند، به نام **توصیفگر**خوانده می شوند. به عنوان مثال، در مورد نوع داده آ رایه ، توصیفگر می تواند شامل نام آرایه ، حد بالا ، حد پایین، نوع عناصر آرایه و اندازه هر عنصر آرایه با شد.

[2] declaration

۲. Type نوع
۳. Initial valueمقدار
۴. Attributeصفت

۳-۱-۴-۱- اعلان عملیات

عملیات نیز همانند متغیرها باید اعلان شوند. عملیات دو نوع می باشند:

۱. عملیاتی که توسط برنامه نویس تعریف شده : باید اعلان شود.
۲. عملیات اولیه : عملیات اولیه نیاز به اعلان ندارند.

اعلانها می توانند اطلاعاتی راجع به عملیات را برای مترجم زبان فراهم کنند. برای عملیات باید موارد زیر مشخص شود.

۱. تعداد پارامترها
۲. ترتیب پارامترها
۳. نوع پارامترها
۴. نتایج

۳-۱-۴-۲- اهداف اعلان:

- **انتخاب نمایش حافظه** (storage representation) مترجم می تواند بهترین نمایش حافظه را برای آن شئ داده ای انتخاب کند.
- **مدیریت حافظه** (storage management) اطلاعاتی که توسط اعلان راجع به طول عمر شئ داده ای مشخص می شود منجر به مدیریت حافظه ی بهتری در طول اجرای برنامه می شود.
- **عملیات چندریختی – چند شکلی** (generic operation) می توان با اعلان چندین شکل مختلف برای عملیات تعریف کرد برای مثال عملگر جمع چندین بار تعریف می شود چون فقط یک عمل را انجام نمی دهد.
- **کنترل نوع و تبدیل نوع** (type checking and type conversion) مهم ترین هدف اعلان ،کنترل نوع ایستا در مقابل کنترل نوع پویا می باشد تا در مواقع لزوم تبدیل نوع هم صورت بگیرد.

۳-۱-۵- کنترل نوع و تبدیل نوع

نمایش حافظه ای که در سخت افزار برای داده ها ساخته می شود، هیچ اطلاعاتی راجع به نوع ندارد و عملیات اولیه بر روی داده ها، کنترل نوع را انجام نمی دهند. بعنوان مثال در حین اجرای برنامه محتویات یک کلمه ممکن است به صورت زیر باشد:

1110011010001…100

این دنباله از بیتها ممکن است یک مقدار صحیح، یک مقدار اعشاری، دنباله ای از کاراکترها، یا یک دستور را نشان دهد. هیچ راهی برای تشخیص وجود ندارد. منظور از کنترل نوع این است که هر عملیاتی که در برنامه انجام می گیرد تعداد و نوع آرگومانهای آن درست باشد.

هنگام کنترل نوع تست می کنیم که ورودیها (آرگومان) صحیح می باشند یا نه؟

چون در سخت افزار کامپیوتر معمولی قادر به تشخیص نیست این کار به صورت نرم افزاری انجام می شود. تا اینکه تست شود تعداد و نوع آرگومانها درست هست یا نه؟

دو نوع کنترل نوع داریم :

۳-۱-۵-۱- کنترل نوع پویا[1]

کنترل نوع ممکن است در زمان اجرا صورت گیرد(کنترل نوع پویا). مانند لیسپ و پرولوگ. بلافاصله قبل از انجام عمل خاصی اجرا می شود در این حالت باید در هر شی داده ای یک بر چسب نوع شی داده ای وجود داشته باشد. همچنین باید برای نتایج خروجی ، نوعی نیز در نظر گرفته شود تا برای کنترل نوع های بعدی استفاده شود.

در اینگونه زبانها متغیرها اعلان نمی شوند، و هیچ پیشفرضی راجع به انواع متغیرها نمی شود.

مزایای کنترل نوع پویا:

۱. انعطاف پذیری زیاد در برنامه.
۲. اعلان برای نوع می تواند نیاز نباشد.
۳. عمل تبدیل نوع می تواند در زمان اجرا انجام شود.

معایب کنترل نوع پویا:

۱. اشکال زدایی (Debug) برنامه و حذف تمام خطا ها ی نوع دشوار است.چون کنترل نوع پویا، انواع داده ها را در زمان اجرای عملیات کنترل می کند، عملیاتی که اجرا نمی شوند، کنترل نخواهند شد. به طور کلی، در حین تست برنامه، تمام مسیرهای اجرایی را نمی توان تست کرد. هر مسیر اجرایی تست نشده، ممکن است حاوی خطا های نوع باشد که بعداً ظاهر شود.
۲. در کنترل نوع پویا لازم است اطلاعات مربوط به نوع(برچسب نوع) در زمان اجرا نگهداری شود و به این ترتیب حافظه بیشتری مصرف می گردد.
۳. کنترل نوع پویا باید به صورت نرم افزاری پیاده سازی شود، زیرا سخت افزار از آن به ندرت پشتیبانی می کند.چون قبل از اجرای هر عملی کنترل نوع صورت می گیرد، سرعت اجرای برنامه کاهش می یا بد (کاهش سرعت).

۳-۱-۵-۲- کنترل نوع ایستا[2]

کنترل نوع ممکن است در زمان ترجمه صورت گیرد(کنترل نوع ایستا) مانند زبان c و pascal.

مزایای کنترل نوع ایستا:

۱. سرعت اجرای برنامه معمولاً زیاد است.
۲. استفاده از حافظه بهینه است.
۳. هیچ اطلاعاتی برای ذخیره نوع در خود برنامه نگهداری نمی کنیم.
۴. تمام مسیرهای اجرای برنامه از جهت نوع چک می شود.

معایب کنترل نوع ایستا:

۱. انعطاف پذیری کم
۲. هنگام مجزا کامپایل کردن برنامه ممکن است مشکلاتی بوجود آید.

[1] Dynamic type checking (D.T.C)
[2] Static type checking (S.T.C)

برای این که کنترل نوع ایستا در زبانی صورت گیرد، به اطلاعاتی نیاز است که عبارت اند از:

۱. برای هر عمل تعداد ترتیب و نوع پارامتر ها و نتایج آن
۲. نوع هر متغیر
۳. نوع هر ثابت

در برخی زبانها به دلیل ساختار زبان کنترل نوع ایستا امکان پذیر نیست. در این زبان ها ممکن است به دو گونه عمل شود:

- کنترل نوع پویا
- عملیات کنترل نشوند.

۳-۱-۵-۳- تبدیل نوع[1] و تبدیل نوع ضمنی[2]

پس از کنترل و تست نوع ممکن است نوع مورد انتظار و نوع واقعی آرگومان یکسان نباشد ودر این حالت ممکن است موارد زیر اتفاق بیفتد:

۱. عدم تطابق نوع ممکن است به عنوان خطا اعلان شود و فعالیت مناسبی صورت گیرد.
۲. ممکن است تبدیل نوع ضمنی صورت گیرد تا نوع آرگومان واقعی به نوع درستی تغییر کند.

اغلب زبانها تبدیل نوع را به دو صورت انجام می دهند:

۱. به صورت مجموعه ای از توابع پیش ساخته که توسط برنامه نویس فراخوانی می شود تا بر تبدیل نوع اثر بگذارند.
۲. در مواردی که عدم تطابق نوع صورت گرفت تبدیل ضمنی به طور خودکار فراخوانی می شود.

تبدیل نوع را از جهاتی می توان به دو دسته تقسیم کرد:

- تبدیل نوع ارتقادهنده
- تبدیل نوع محدودکننده

در اغلب زبان های بر نامه سازی، تبدیل نوع معمولاً به دو صورت انجام می گیرد:

- تبدیل نوع صریح
- تبدیل نوع ضمنی (تبدیل یا خطا)

در کنترل نوع پویا در نقطه‌ای از زمان اجرا که عدم تطابق رخ دهد، تبدیل ضمنی صورت می گیرد. در کنترل نوع ایستا ، کد اضافی در برنامه ترجمه شده قرار می گیرد تا عملیات تبدیل نوع رادر نقطه خاصی از برنامه اجرا کند.

تبدیلات نوع ضمنی آزادی برنامه نویس را افزایش می دهند. اما در برخی موارد ممکن است این تبدیلات ضمنی محدود کننده باشند و بخشی از اطلاعات هنگام تبدیل از بین برود. به عنوان مثال: تبدیل flout به int

۳-۱-۶- انتساب و مقدار دهی اولیه[3]

انتساب عملیات اصلی برای تغییر binding یک مقدار به یک شی داده است. این تغییر، اثر جنبی عملیات است. این دستور به دو شکل زیر می تواند انجام شود:

[1] conversion
[2] coersion
[3] Assignment and Initialization

Assignment (:=): $Type_1 * Type_2 \rightarrow Void$
Assignment(=): $Type_1 * Type_2 \rightarrow Type_3$

در حالت اول عمل انتساب هیچ مقداری را برگشت نمی دهد

اما در حالت دوم عمل انتساب مقداری را برگشت می دهد و این مقدار می تواند به عنوان مقدار راست برای انتساب دیگر و یا سایر عملیات استفاده شود.

تعریف عملیات انتساب به صورت زیر :

۱. مقدار چپ اولین عبارت عملوند را محاسبه کن.
۲. مقدار راست دومین عبارت عملوند را محاسبه کن.
۳. مقدار راست محاسبه شده را به شئ داده مقدار چپ محاسبه شده نسبت بده.
۴. مقدار راست محاسبه شده را به عنوان نتیجه عملیات برگردان.

مقدار دهی اولیه: متغیر فاقد مقدار اولیه یا شئ داده فاقد مقدار اولیه، یک شی داده است که ایجاد شده ولی هنوز مقداری به آن داده نشده است. متغیر های فاقد مقدار اولیه عامل مهمی برای بروز خطا در برنامه نویسی هستند.

دو مقدار دهی اولیه وجود دارد:

۱. صریح
۲. ضمنی

۳-۲- انواع داده اسکالر[1]

اشیایی وجود دارند که برای اشیای داده خود فقط یک صفت دارند. به عنوان مثال، شئ نوع صحیح دارای یک مقدار صحیح است و هیچ اطلاعات دیگری راجع به آن شئ نمی توان بدست آورد. سپس داده مرکب را در نظر می گیریم که درآن یک شی می تواند چندین صفت داده باشد. به عنوان مثال، رشته کاراکتری حاوی دنباله ای از کاراکترها بعنوان داده خود است ولی ممکن است صفات دیگری مثل اندازه رشته داشته باشد.

بطور کلی اشیای اسکالر از معماری سخت افزار کامپیوتر پیروی می کنند. داده های مرکب معمولاً ساختار پیچیده ای هستند که توسط کامپایلر تولید می شوند و توسط سخت افزار پیاده سازی نشده اند.

۳-۲-۱- انواع داده عددی

۳-۲-۱-۱- انواع صحیح

مشخصات : یک شی داده از نوع صحیح، معمولاً صفتی غیر از نوع ندارد.

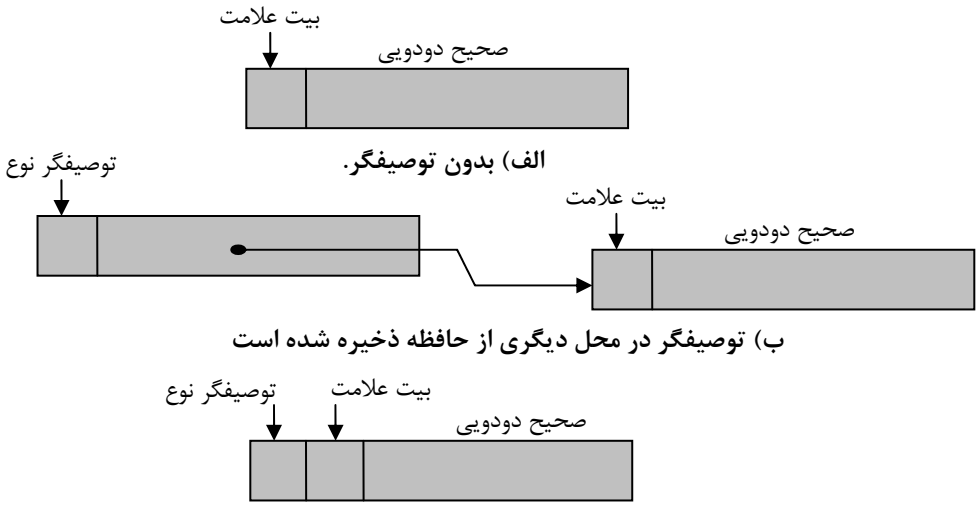
عملیات محاسباتی: عملیات بر روی اشیای داده صحیح شامل موارد زیر است:

- عملیات محاسباتی
- عملیات رابطه ای
- انتساب

[1] Scalar Data Type

- عملیات بیتی

پیاده سازی: شکل زیر سه نمایش حافظه برای مقادیر صحیح را نشان می دهد.



الف) بدون توصیفگر.

ب) توصیفگر در محل دیگری از حافظه ذخیره شده است

ج) توصیفگر و مقدار در یک کلمه ذخیره می شوند.

سه نمایش حافظه برای مقادیر صحیح

۳-۲-۱-۲- زیر بازه ها[1]

مشخصات: زیر بازه ای از نوع داده صحیح یا زیر نوعی از نوع داده صحیح است و شامل دنباله ای از مقادیر صحیح و بازه محدود است. مثل مقادیر صحیح در بازه ۱ تا ۱۰ یا بازه ۵- تا ۵.

پیاده سازی: انواع زیربازه دو اثر مهم در پیاده سازی دارد:

- نیاز به حافظه کمتر : چون بازه کمتری از مقادیر را شامل می شود، مقادیر زیربازه نسبت به مقادیر صحیح معمولی، بیتهای کمتری نیاز دارد.
- کنترل نوع بهتر : اعلان یک متغیر از نوع زیربازه باعث می شود کنترل نوع دقیقتری صورت گیرد. به عنوان مثال، اگر متغیر Month به این صورت باشد: Month: 1..12 ، آنگاه دستور زیر غلط است:

```
Month := 0
```

این خطا در زمان کامپایل تشخیص داده می شود. در بسیاری از موارد کنترل نوع زیربازه ممکن نیست. بعنوان مثال، انتساب زیر را در نظر بگیرید:

```
Month := Month + 1
```

۳-۲-۱-۳- اعداد حقیقی ممیز شناور[2]

مشخصات : معمولاً با صفت نوع داده مثل real در فرترن یا float در C مشخص می شود.

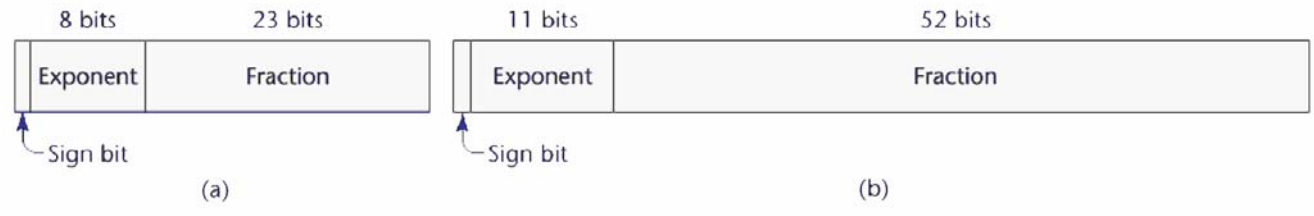
[1] subrange
[2] Floating Point Real Number

عملیات محاسباتی، رابطه‌ای و انتساب که برای مقادیر مطرح شد، در اعداد حقیقی نیز امکان پذیر است، ولی عملیات بولین گاهی محدود می شود. به دلیل اصول مربوط به گرد کردن کمتر اتفاق می افتد که دو مقدار حقیقی با هم مساوی باشند. به همین دلیل تساوی بین دو عدد حقیقی ممکن است توسط طراح زبان جلوگیری شود تا این نوع خطاها پیش نیاید.

پیاده سازی: نمایش های حافظه برای انواع آن معمولاً به سخت افزار بستگی دارد که در آن، محل حافظه به دو بخش مانتیس (ارقام با ارزش عدد) و توان تقسیم می شود. این مدل نشانه گذاری علمی را شبیه سازی می کند که در آن هر عدد N را می توان بصورت $N = m \times 2^k$ نشان داد، به طوری که m بین صفر و یک و k مقدار صحیح باشد. استاندارد IEEE 754 در بسیاری از پیاده سازیها تعریف قابل قبولی برای فرمت ممیز شناور است.

استاندارد IEEE 754

استاندارد IEEE 754 استاندارد ۳۲ و ۶۴ بیتی را برای اعداد ممیز شناور مشخص می کند.



استاندارد ۳۲ بیتی

اعداد شامل ۳ فیلد هستند:

بیت S: فیلد علامت یک بیتی. صفر به معنای مثبت است.

فیلد E: توان ظاهری ۸ بیتی با افزونی ۱۲۷ ، در بازه صفر تا ۲۵۵ که معادل توانهای ۲ از ۱۲۷- تا ۱۲۸ است.

فیلد M: مانتیس ۲۳ بیتی. چون اولین بیت در مانتیس در عدد نرمال شده همیشه یک است، می تواند حذف شود. و سخت‌افزار به طور خودکار آن را اضافه می کند تا بیت ۲۴ از دقت نیز به وجود آید.

S علامت را مشخص می کند. با توجه به مقادیر E و M ، مقدار دقت بصورت زیر است:

پارامترها	مقدار
E=255 and M≠0	عدد نامعتبر
E=255 and M=0	∞
0 < E < 255	$2^{E-127}(1.M)$
E=0 and M≠0	$2^{-126}.M$
E=0 and M=0	0

چند نمونه:

$+1 = 2^0 \times 1 = 2^{127-127} \times (1).0(\text{binary}) =$ 0 01111111 000000…

$+1.5 = 2^0 \times 1.5 = 2^{127-127} \times (1).1(\text{binary}) =$ 0 01111111 100000…

$-5 = -2^2 \times 1.25 = 2^{129-127} \times (1).01(\text{binary}) =$ 1 10000001 010000…

بازه ای که در اینجا مشخص می گردد 10^{-38} تا 10^{38} است. در فرمت ۶۴ بیتی، توان به ۱۱ بیت افزایش می یابد که بازه آن از ۱۰۲۲- تا ۱۰۲۳ است و اعداد حاصل در بازه 10^{-308} تا 10^{308} است.

۳-۲-۱-۴- اعداد حقیقی ممیز ثابت[1]

مشخصات: اغلب سخت افزارها شامل اشیائ داده صحیح و ممیز شناور هستند. برای برخی از داده های حقیقی اگر از ممیز شناور استفاده کنیم خطای گرد کردن اتفاق خواهد افتاد. می توان برای اینگونه داده ها از ممیز ثابت استفاده کرد.

پیاده سازی: ممکن است مستقیماً توسط سخت افزار پشتیبانی شود یا به صورت نرم افزاری شبیه سازی گردد. این گونه اعداد بصورت صحیح ذخیره می شوند و نقطه اعشار بعنوان صفت آن شی داده ای است. اگر مقدار X برابر ۱۰/۴۲۱ باشد، مقدار راست X برابر ۱۰۴۲۱ و شی X حاوی صفتی به نام فاکتور مقیاس[2] برابر ۳ است و معنایش این است که سه رقم بعد از نقطه اعشار قرار دارد، یعنی:

$$Value(X) = rvalue(X) \times 10^{-SF}$$

SF صرف نظر از مقدار راست X همواره برابر ۳ است.

۳-۲-۱-۵- سایر انواع داده عددی

اعداد موهومی: عدد موهومی متشکل از یک جفت از اعداد است که یکی از آنها بخش حقیقی و دیگری بخش موهومی را نشان می دهد.

اعداد گویا: عدد گویا خارج قسمت دو مقدار صحیح است.

۳-۲-۲- نوع شمارشی[3]

مشخصات: . مقادیر نوع شمارشی بر اساس تعریف برنامه نویس مشخص می شوند که لیست مرتبی از مقادیر مجزا است. مقادیر نوع شمارشی به نام ثوابت شمارشی نیز خوانده می شوند. برنامه نویس اسامی لیترالهایی را که باید برای مقادیر مورد استفاده قرار گیرند و همچنین ترتیب آنها را با استفاده از اعلانی مشخص می کند. نمونه ای از نوع شمارشی و تعریف متغیر هایی از آن نوع در#c به صورت زیر است:

مثال: enum StudentClass {Fresh, Soph, Jonior, Senior};

این تعریف لیترالهای Fresh ، Soph ، Jonior و Senior را نیز تعریف می کند که در هر جایی از برنامه قابل به کارگیری و استفاده می باشند.

عملیات اصلی روی نوع داده شمارشی عبارتند از عملیات رابطه ای (تساوی ، کچکتر از ، بزرگتر از ، و غیره)، انتساب و عملیات successor (بعدی) و predecessor (قبلی) که به ترتیب عناصر بعدی و قبلی را مشخص می‌کنند.

پیاده سازی: نمایش حافظه برای شی داده ای از نوع شمارشی ساده است. هر مقدار در دنباله شمارشی در زمان اجرا به وسیله مقادیر صحیح ۰ ، ۱ ، ۲ و ... قابل نمایش است. چون فقط مجموعه کوچکی از مقادیر در نوع شمارشی وجود دارد و مقادیر منفی نیستند ، نمایش آنها از نمایش مقادیر صحیح نیز ساده تر است. بعنوان مثال، نوع Class که در بالا تعریف شد، فقط چهار مقدار ممکن دارد که در زمان اجرا به صورت Fresh=0 ، Soph=1 ، Jonior=2 و Senior=3 نمایش داده می شود. در زبان C می توان این ترتیب را تغییر داد.

[1] Fixed point
[2] Scale Factor (SF)
[3] Enumeration

۳-۲-۳- نوع بولی[1]

مشخصات: متشکل از اشیای داده ای است که یکی از دو مقدار TRUE یا FALSE را می پذیرد.

متداولترین عملیاتهایی که بر روی نوع داده بولی عبارتند از : انتساب ، and ، or ، xor ، not ، not and و غیره.

پیاده سازی: نمایش حافظه برای شی داده بولی یک بیت از حافظه است، به شرطی که نیاز به توصیفگر برای نوع داده ای نباشد. چون یک بیت در حافظه قابل نمایش نیست، معمولاً از یک واحد قابل آدرس دهی مانند بایت یا کلمه استفاده می شود. مقادیر true و false به دو روش در این واحد حافظه نمایش داده می شوند:

- بیت خاصی برای این مقادیر استفاده می شود، به طوری که false=0 و true=1 و بقیه بیتهای مربوط به بایت یا کلمه بدون استفاده می مانند.
- مقدار صفر در کل واحد حافظه (بایت یا کلمه) نشان دهنده false و مقدار غیرصفر نشان دهنده true است.

بعضی از زبانها مثل Cفاقد نوع بولی اند.در این زبان،از نوع داده صحیح برای این منظور استفاده می شود.مقدار صفر نشان دهنده false و مقدار غیر صفر نشان دهنده true است.دستورات زیر را ببینید:

```
int flag;
flag = 7;
```

چون flag برابر با صفر نیست، به عنوان true استفاده می شود. اگر انتساب flag = 0; را داشته باشیم، flag دارای ارزش false خواهد بود. این روش استفاده از مقادیر صحیح به جای مقادیر بولی، اشکالاتی دارد.

به عنوان مثال، اگر به جای and منطقی (&&) از and بیتی (&) یا به جای or منطقی (| |) از or بیتی (|)، یا به جای نقیض منطقی (!) از نقیض بیتی (~) استفاده کنید، با مشکل مواجه خواهید شد.دستورات زیر را ببینید:

```
int found;
found = 12;
```

نمایش بیتی عدد ۱۲ در یک کلمه ۱۶ بیتی به صورت زیر است که ارزش درستی دارد:

Found:

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

اگر آن را نقیض بیتی کنید (~found) نمایش بیتی آن به صورت زیر است که بازهم ارزش درستی دارد:

~Found:

1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1

بنابراین نتیجه می گیریم که found و نقیض بیتی آن هر دو ارزش درستی دارند. در صورتی که بخواهید واقعاً نقیض found ارزش نادرستی داشته باشد، باید آن را نقیض منطقی کنید (found!) در این صورت نمایش بیتی found! به صورت زیر خواهد بود که ارزش نادرستی دارد:

!Found:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Boolean

۳-۲-۴- کاراکترها[1]

مشخصات : نوع داده کاراکتری اشیای داده ای را به وجود می آورد که مقدار آنها یک کاراکتر است. مجموعه ای از مقادیر کاراکتری ممکن معمولاً بصورت نوع شمارشی تعریف شده در زبان در نظر گرفته می شود.

پیاده سازی: مقادیر داده های کاراکتری همیشه توسط سخت افزار و سیستم عامل پشتیبانی می شوند. اگر نمایش کاراکتری که توسط زبان تعریف شد، با نمایش کاراکتری که توسط سخت افزار پشتیبانی می شود یکسان باشد، آنگاه عملیات رابطه ای نیز مستقیما در سخت افزار نمایش داده می شوند یا توسط کدهای نرم افزاری شبیه سازی خواهد شد.

۳-۳- انواع داده مرکب

۳-۳-۱- رشته های کاراکتری

مشخصات و نحو

نکات طراحی رشته

۱. آیا رشته ها باید به صورت آرایه ای از کاراکترها باشند یا به صورت نوع داده اولیه؟ دقت کنید که اگر آرایه ای از کاراکترها باشد، عملیات اندیس گذاری بر روی آن امکان پذیر است، ولی اگر به صورت نوع داده اولیه باشد، امکان پذیر نیست.
۲. طول رشته ها باید ایستا یا پویا باشد؟

با رشته های کاراکتری حداقل به سه روش رفتار می شود:

۱. طول ثابت
۲. طول متغیر با حد بالا
۳. طول نامحدود

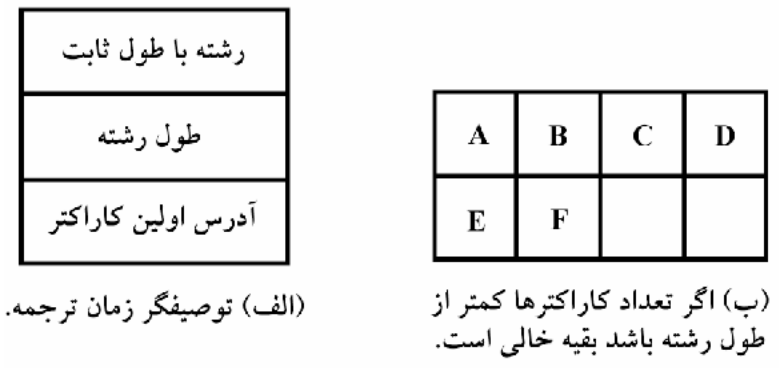
رشته های کاراکتری در زبان C کمی پچیده تر می باشند. در انتهای رشته باید کاراکتر تهی (‘\0’) قرار گیرد و برنامه نویس باید اطمینان حاصل کند که رشته ها به تهی ختم می شوند.

عملیات گوناگونی بر روی رشته ها انجام پذیر است که بعضی از آنها عبارتند از:

۱. الحاق
۲. عملیات رابطه ای در رشته ها
۳. انتخاب زیر رشته با استفاده از اندیس
۴. فرمت بندی ورودی – خروجی
۵. انتخاب زیر رشته با تطابق الگو[2]
۶. رشته های پویا

[1] Character
[2] Pattern-matching

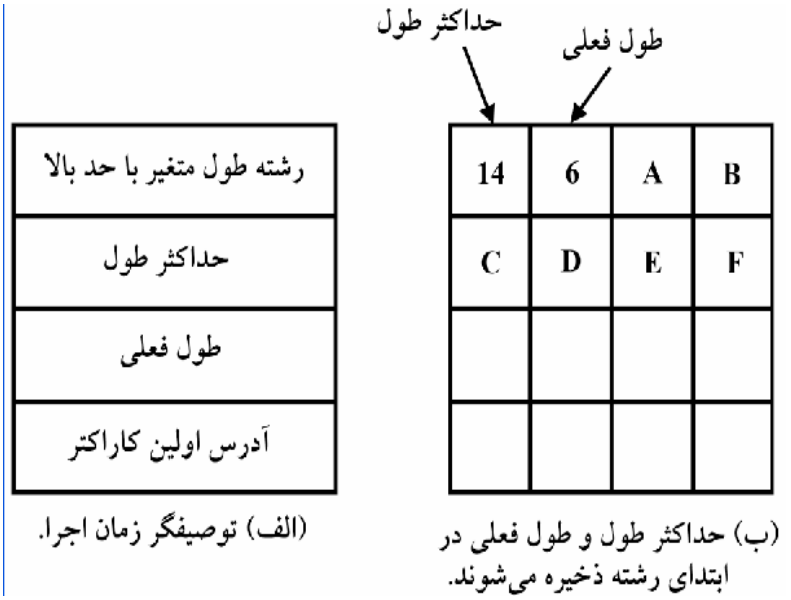
پیاده سازی: برای رشته ای با طول ثابت نمایش حافظه همان شکلی است که برای بـردار فـشرده ای از کاراکترهـا اسـتفاده شد. برای رشته طول متغیر با حد معین نمایش حافظه از توصیفگری استفاده می کند که حاوی حداکثر طول و طول فعلی رشته ذخیره شده در شی داده است. برای رشته های نامحدود می توان از نمایش حافظه پیوندی اشیا داده طول ثابت استفاده کرد.



(الف) توصیفگر زمان ترجمه.

(ب) اگر تعداد کاراکترها کمتر از طول رشته باشد بقیه خالی است.

پیاده سازی رشته با طول ثابت.

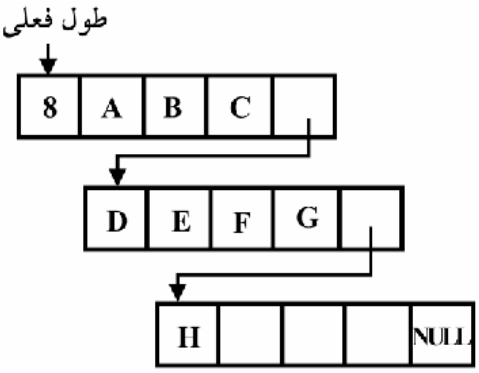


(الف) توصیفگر زمان اجرا.

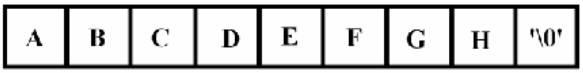
(ب) حداکثر طول و طول فعلی در ابتدای رشته ذخیره می‌شوند.

پیاده سازی رشته با طول متغیر با حد بالا.

(الف) توصیفگر زمان اجرا.



(ب) در هر بلوک ۴ کاراکتر قرار می‌گیرد.
طول رشته در ابتدای رشته است.



(ج) نمایش رشته طول متغیر به صورت آرایه
(در C و C++ استفاده می‌شود).

پیاده سازی رشته با طول متغیر.

۳-۳-۲- اشاره گرها و اشیای داده برنامه نویس

معمولاً در هر زبانی امکاناتی وجود دارد که بتوان با استفاده از اشاره گرها، ساختارهایی را ایجاد کرد که عناصر اشـیای داده را به یکدیگر متصل کرد. زبان باید ویژگیهای زیر را داشته باشد:

۱. نوع داده اولیه اشاره گر
۲. عمل ایجاد کردن برای اشیای داده طول ثابت، مثل آرایه ها، رکوردها و انواع اولیه.
۳. عملیات دستیابی به محتویات موجب می شود تا محتویات جایی که اشاره گر به آن اشاره می کند،دستیابی شود.

مشخصات: نوع داده اشاره گر دسته ای از اشیای داده را تعریف می کند که مقادیر آنها آدرسهای اشیای دیگری اند. بـا شـئ داده ای از نوع اشاره گر می توان به دو روش برخورد کرد:

۱. اشاره گر ها ممکن است فقط به یک نوع شی داده مراجعه کنند.
۲. اشاره گرها ممکن است به هر نوع شی داده مراجعه کنند.

عملیات مهمی که در مورد اشاره گر ها انجام می شوند، عبارت انداز:

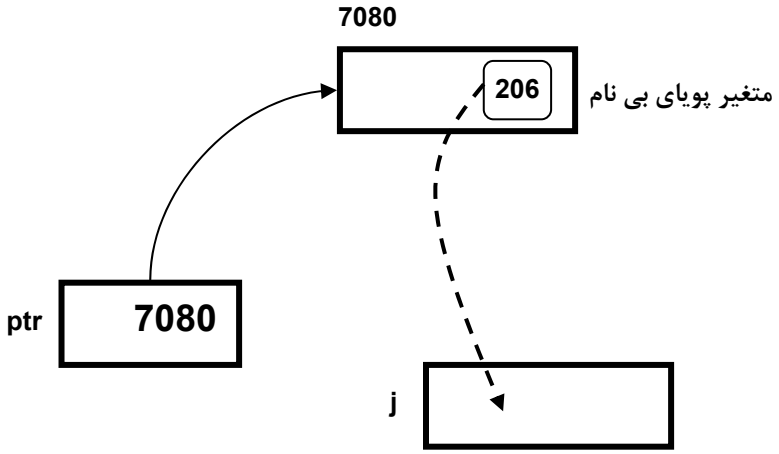
۱. عملیات انتساب

```
int  *p,  *q;
در C :      p = (int *) malloc(sizeof(int));
در C++ :    q = new int;
```

۲. عملیات دستیابی به محتویات

```
1.  int j;
2.  int *ptr;
```

3. *ptr = 206;
4. j = *ptr;



۳. عملیات محاسباتی و رابطه ای

```
int  *p,  *q;
…
p++;
q--;
if(p == q)
      ...
```

پیاده سازی: شی داده اشاره گر به صورت محلی از حافظه نمایش داده می شود که شامل آدرس محل دیگری از حافظه است.

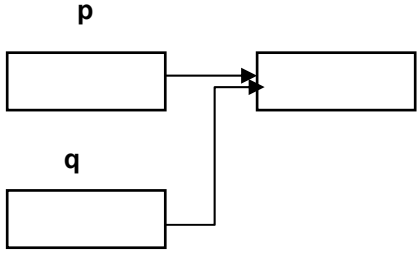
دو نمایش حافظه برای مقادیر اشاره گر استفاده می شود:

- آدرس مطلق
- آدرس نسبی

مشکلات اشاره گر ها

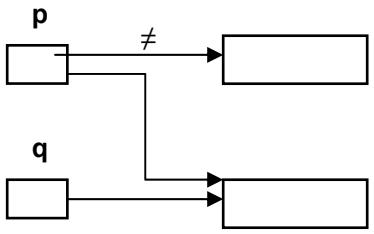
۱. **ارجاع معلق** ارجاع معلق یک مسیر دستیابی (اشاره گری) است که به شیء داده ای اشاره می کند که حافظه مربوط به آن شیء داده به heap برگردانده شده است . ارجاع معلق ، مسئله بسیار مهمی است ، زیرا ممکن است منجر به تغییر ناخواسته در شیء داده ها شود و نتایج غیر منتظره ای را به وجود آورد. به عنوان مثال، دستورات زیر را در C++ در نظر بگیرید:

1. int *p, *q;
2. p = new int;
3. q = p;
4. free(p);
5. *q = 500



۲. حافظه مازاد (زباله) زباله وقتی به وجود می آید که بخشی از حافظه که در اختیار برنامه نویس است، قابل دستیابی نباشد. به عبارت دیگر، اگر مسیر دستیابی به شیء داده ای از بین رفته باشد، حافظه آن زباله محسوب می شود. به عنوان مثال، دستورات زیر را در C++ در نظر بگیرید:

```
1. int *p, *q;
2. p = new int;
3. q = new int;
4. p = q;
```



۳-۳-۳- فایلها و ورودی - خروجی

فایل ساختمان داده ای با دو ویژگی است:

۱. بر روی حافظه ثانویه مثل دیسک یا نوار تشکیل می شود و ممکن است بسیار بزرگتر از سایر ساختمان داده ها باشد.

۲. طول عمر آن می تواند بسیار زیاد باشد.

انواع فایل

- متداول ترین فایلها، فایلهای ترتیبی اند.
- فایلهای دستیابی مستقیم (direct access)
- فایلهای ترتیبی شاخص دار (indexed sequential files)
- فایلهای متنی

متداول ترین فایل ها ، فایل های ترتیبی اند ، اما اغلب زبان ها ، از فایل های **دستیابی مستقیم** و **فایل ها ی ترتیبی شاخص دار** استفاده می کنند. دو کاربرد عمده فایل ها عبارت اند از: ورودی و خروجی داده ها در محیط عملیات خارجی و حافظه موقت در مواقعی که حافظه کافی وجود ندارد. عناصر فایل را **رکورد** گویند. ولی دراین جا ازاین اصطلاح صرف نظر می کنیم تا با ساختمان داده رکورد اشتباه نشود.

۳-۳-۳-۱-فایلهای ترتیبی

فایل می تواند در حالت خواندن یا در حالت نوشتن دستیابی شود. در هر دو حالت، یک اشاره گر موقعیت فایل وجود دارد که موقعیتی را قبل از اولین عنصر فایل، بین دو عنصر فایل ، یا بعد از آخرین عنصر فایل، تعیین می کند.در حالت نوشتن، اشاره گر موقعیت فایل، همیشه به بعد از آخرین عنصر فایل اشاره می کند و می توان عنصری را در آن محل نوشت و فایل را به اندازه یک عنصر بسط داد. در حالت خواندن، اشاره گر موقعیت فایل، می تواند در هر نقطه ای از فایل باشد ، می توان عمل خواندن را در آن محل انجام داد . در این حالت نمی توان عنصر جدیدی را اضافه کرد. در هر دو حالت، پس از عمل خواندن یا نوشتن، اشاره گر موقعیت فایل حرکت می کند تا به موقعیت فایل حرکت می کند تا به موقعیت عنصر بعدی اشاره نماید. اگر این اشاره گر بعد از آخرین عنصر فایل قرار گیرد، می گوییم که به انتهای فایل رسید یم.

مشخصات عملیات اصلی بر روی فایل های ترتیبی عبارت اند از:

- بازکردن

- خواندن
- نوشتن
- تست انتهای فایل
- بستن

۳-۳-۳-۲- فایل های دستیابی مستقیم

درفایل ترتیبی، عناصر به ترتیبی که درفایل قرار دارند بازیابی می شوند.گرچه عملیات محدودی برای جا به جایی اشاره گر موقعیت فایل وجود دارد، ولی در این فایل ها دستیابی تصادفی به عناصر، غیر ممکن است. فایل **دستیابی مستقیم** طوری سازمان دهی می شود که می توان به هر عنصر به طور تصادفی دست یافت(مثل آرایه و رکورد). اندیسی که برای انتخاب یک عنصر به کار می رود، کلید نام دارد که ممکن است یک مقدار صحیح یا شناسه دیگری باشد. اگر یک مقدار صحیح باشد، کلید شبیه اندیس معمولی است که برای تعیین عنصری از فایل به کار می رود. اما، چون فایل دستیابی مستقیم، در حافظه ثانویه ذخیره می شود، پیاده سازی فایل و عملیات انتخاب، متفاوت از آرایه است.

۳-۳-۳-۳- فایل ترتیبی شاخص دار

فایل ترتیبی شاخص دار، شبیه فایل دستیا بی مستقیم است. به طوری که امکان دستیابی ترتیبی، با شروع از عنصری که به طور تصادفی انتخاب شد، وجود دارد. به عنوان مثال، اگر عنصری با کلید ۲۷ انتخاب(خوانده) شود، عملیات خواندن بعدی ممکن است عنصر "بعدی" را به ترتیب انتخاب کند (به جای دادن مقدار کلید). این سازمان فایل، مصالحه ای را بین سازمان های ترتیبی محض و دستیابی مستقیم محض به وجود می آورد.

فصل چهارم

۴- بسته بندی

۴-۱- ساختمان داده

هر ساختمان داده یک شیء داده ای است که عناصر آن اشیاء داده ای دیگری می باشند. از جمله ساختمان داده های مهم عبارتند از :آرایه ها ، رکوردها، پشته ها، لیست ها و مجموعه ها. بعضی از ساختمان داده ها توسط برنامه نویس تعریف می شود و بعضی دیگر در حین اجرا توسط سیستم ایجاد میشود. انواع ساختمان داده نیز همانند انواع اولیه، اصولی مثل، مشخصات، بیان سازی ، اعلانها و کنترل نوع دارند.دو مورد از اینها مهم می باشند:

۱. چگونه باید اشیاء داده عنصر ساختمان داده و روابط بین آنها را نشان داده تا انتخاب عناصر از ساختمان داده آسان باشد.
۲. بسیاری از عملیات ساختمان داده ها مسئله مدیریت حافظه را بوجود می آورد که در مورد اشیای داده ای اولیه وجود ندارد.

۴-۱-۱- مشخصات انواع ساختمان داده

صفات اصلی مشخص کننده ساختمان داده ها عبارتند از :

۱. **تعداد عناصر** : بر اساس تعداد عناصر اندازهٔ ساختمان داده ممکن است ثابت یا متغیر باشد آرایه ها و رکودها انواع داده با اندازه ثابت ، پشته ها، لیست ها، مجموعه ها ، جدولها و فایلهای ساختمان داده با طول متغیر می باشند اشیای داده طول متغیر اغلب از نوع داده اشاره گر استفاده می کنند.
۲. **نوع هر عنصر** : اگر تمام عناصر ساختمان داده از یک نوع باشند آنرا همگن و گرنه آنرا غیر همگن می گویند. آرایه ها، رشته ها کاراکتری ، مجموعه ها و فایلها همگن و رکوردها و لیست ها ناهمگن می باشند .
۳. **اسامی برانتخاب عناصر** : ساختمان داده نیاز به مکانیزمی دارد که بتوان عناصر آنرا انتخاب کرد و آرایه های از اندیس استفاده میشود و در لیست ها این کار با استفاده از نام استفاده می شود.
۴. **حداکثر تعداد عناصر**
۵. **سازمان عناصر** : متداولترین سازمان دنباله خطی از عناصر میباشد (ترتیبی) حالت دیگر ساختار غیر ترتیبی می باشد عملیاتهائیکه بر روی ساختمان داده ها صورت می گیرد.

۴-۱-۲- عملیات ساختمان داده ها

مشخصات دامنه و برد عملیات در انواع داده ای ساختمان می تواند مانند انواع داده ای اولیه باشد دسته های دیگری از عملیات از اهمیت ویژه ای برخودراند.

۱. **عملیات انتخاب عناصر** : پردازش ساختمان داده ها مستلزم دستیابی به عناصر آنها می باشد . نوع عملیات انتخاب وجود دارد که به عناصر ساختمان داده ها دستیابی دارند. انتخاب تصادفی و انتخاب ترتیبی . در انتخاب

تصادفی عناصر دلخواهی دستیابی می شوند و در انتخاب ترتیبی عناصر به ترتیبی که از قبیل مشخص شده است دستیابی می شوند.

۲. **عملیات بر روی کل ساختمان** : عملیات ممکن است کل ساختمان را بعنوان آرگومان دریافت کرده و ساختمان داده جدیدی را تولید کند مانند جمع آرایه یا انتساب رکوردهایی به رکورد دیگر

۳. **درج و حذف عناصر**

۴. **ایجاد و حذف ساختمان داده** که عمدتاً برای مدیریت حافظه استفاده می شود.

نکته : باید انتخاب یا دستیابی به یک عنصر را با عملیات ارجاع تمییز داد. هر شیء داده دارای یک نام است مثل بردار زمانیکه می نویسیم A[4] تا به عنصر چهارم از آرایه A دسترسی پیدا کنیم، دو مرحله را طی می کنیم که مرکب از ***عملیات ارجاع و عملیات انتخاب*** است . عملیات ارجاع موقعیت فعلی نام A را تعیین می کند و نتیجه را بصورتی اشاره گر بر می گرداند که به آن عنصر داخل بردار اشاره می کند. عملیات انتخاب، اشاره گر به بردار را گرفته، همراه با اندیس ۴ ، سپس اشاره گری را بر می گرداند که به آن عنصر داخلی بردار اشاره میکند. در این بخش ما به عملیات انتخاب می پردازیم.

۴-۱-۳- پیاده سازی انواع ساختمان داده

پیاده سازی انواع ساختمان داده ها، موضوع وجود دارد که انتخاب نمایش حافظه را تحت تأثیر قرار می دهد انتخاب کارآمد عنصر از ساختمان و مدیریت حافظه کارآمد برای پیاده سازی زبان.

۴-۱-۳-۱- نمایش حافظه

نمایش حافظه برای ساختمان داده ها شامل

۱. حافظه ای برای عناصر ساختمان داده

۲. توصیف گر اختیاری آنها است.

دو نمایش حافظه اصلی وجود دارد که عبارتند از:

۱. **نمایش ترتیبی**[1]: در این حالت ساختمان داده در بلوک پیوسته ای از حافظه ذخیره می شود که شامل توصیف گر و عناصر است.

۲. **نمایش پیوندی**[2]: در حالت نمایش پیوندی، ساختمان داده در چندین بلوک ناپیوسته ذخیره می شود و بلوک با استفاده از اشاره گر به همدیگر پیوند داده می شوند.

نمایش حافظه ترتیبی برای ساختمان داده های طول ثابت و گاهی برای ساختمان داده های طول متغیر همگن بکار می رود و نمایش های پیوندی معمولا برای ساختمانهای طول متغیر مثل لیستها بکار می روند.

۴-۱-۳-۲- پیاده سازی عملیات ساختمان داده ها

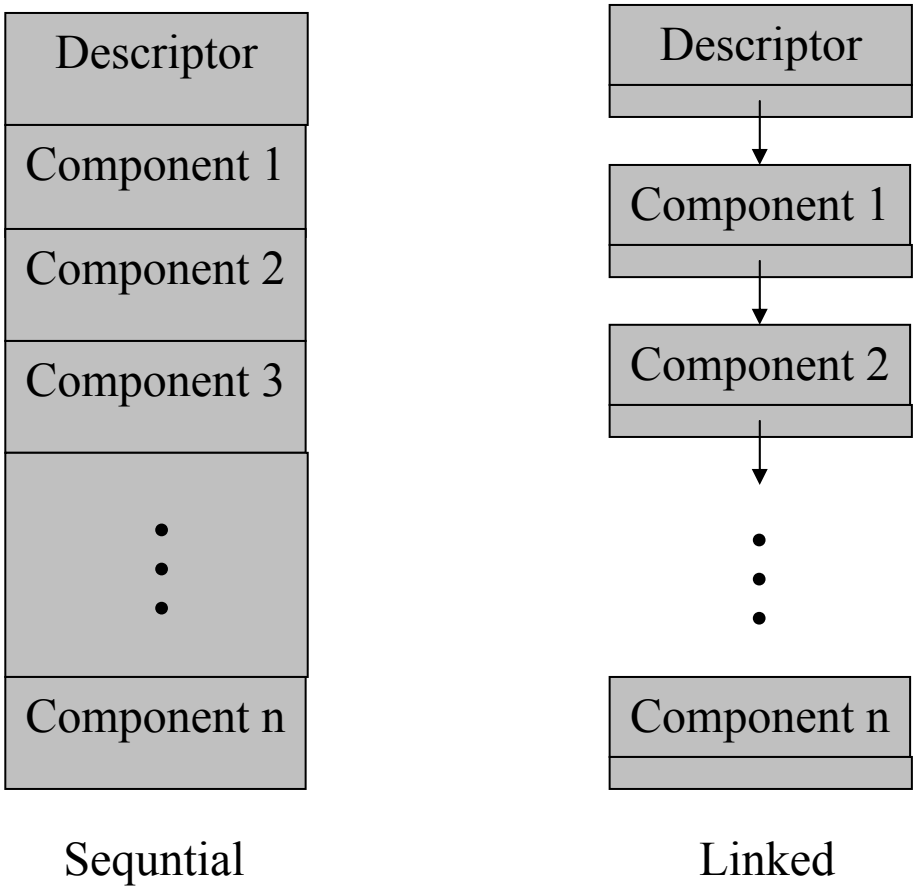
انتخاب عناصر ساختمان داده مهم ترین مسئله در پیاده سازی آن می باشد و همچنین کارآمد بودن عملیات انتخاب تصادفی و انتخاب ترتیبی ضروری است.

[1] sequential
[2] Linked

نمایش ترتیبی. در انتخاب تصادفی آدرس **پایه و آفست**[1] باید با استفاده از فرمول دستیابی محاسبه شود. محل نسبی عنصر انتخاب شده در بلوک ترتیبی، افست نام داده و محل شروع بلوک ***آدرس پایه*** نام دارد. فرمول دستیابی که با استفاده از نام یا اندیس عنصر مطلوب مشخص می شود، چگونگی محاسبه افست را تعیین می‌کند. سپس آفست به آدرس پایه اضافه می شود تا محل واقعی عنصر مورد نظر در حافظه مشخص گردد. برای ساختار همگنی مانند آرایه که بطور ترتیبی ذخیره می‌شود انتخاب دنباله ای از عناصر می تواند بصورت زیر انجام شود.

۱. برای دستیابی به اولین عنصر دنباله از محاسبه آدرس پایه ـ آفست استفاده می کنیم.
۲. برای دستیابی به عنصر بعدی دنباله اندازه عنصر فعلی را به موقعیت عنصر فعلی اضافه کنید.

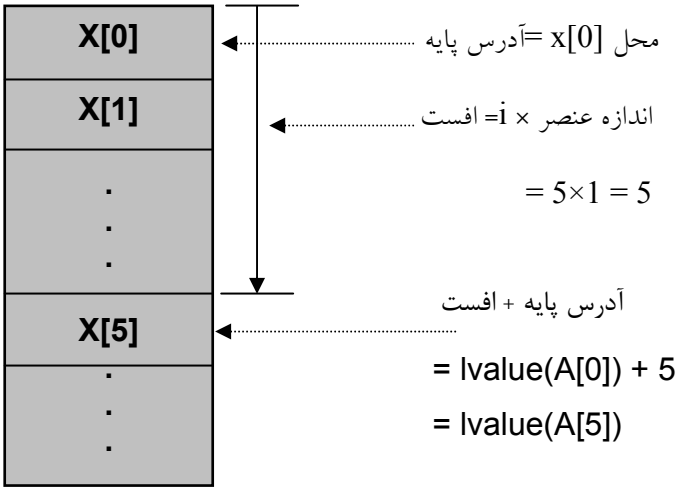
در این حالت انتخاب تصادفی خواهند بود.



نمایش حافظه برای ساختمان داده های خطی

نمایش پیوندی : در نمایش پیوندی به منظور انتخاب با یک زنجیره ای از اشاره گرها را از اولین بلوک موجود در ساختار تا رسیدن به عنصر مورد نظر دنبال کنیم. در این حالت پس از انتخاب اولین عنصر اشاره گر پیوندی را تا عنصر مورد نظر دنبال خواهیم کرد. در این حالت انتخاب ترتیبی خواهند بود.

[1] base-address-offset



دستیابی به آرایه کاراکتری در C

۴-۱-۳-۳-مدیریت حافظه و ساختمان داده ها

طول عمر هر شیء داده با انقیاد شی ء به محلی از حافظه شروع می‌شود. یعنی زمانیکه بلوکی از حافظه به آن شیء اختصاص داده شد، برای ساختمان داده های طول متغیر هر یک از عناصر طول عمر خاص خود را خواهد داشت. طول عمر شیء داده ای با حذف آن از روی حافظه پایان می یابد. زمانیکه شی ء داده ای ایجاد می شود یک ***مسیر دستیابی*** برای آن ایجاد می شود بطوری که بتوان به آن دسترسی پیدا کرد. ایجاد مسیر دستیابی یا با انتخاب نام برای آن انجام می شود یا از طریق اشاره گر به آن ساختمان صورت می گیرد. به علت تاثیر متقابل بین طول عمر شیء داده و مسیر دستیابی مسئله مهم در مدیریت حافظه بوجود می آید.

۱. **زباله**[1] **(حافظه زاید)** : وقتی مسیر دستیابی به یک شیء داده از بین برود ولی خود شیء داده وجود داشته باشد شی ء داده را ***زباله*** گویند.
۲. **ارجاع معلق**[2] : ارجاع معلق یک مسیر دستیابی است که پس از اینکه طول عمر شی ء داده خاتمه یافت وجود داشته باشد.

ارجاع های معلق مسئله مهمی برای مدیریتی حافظه است بطوریکه جامعیت ساختارهای زمان اجرا در حین اجرای برنامه خدشه دار می شود. بعنوان مثال مقداری به شیء داده نسبت داده می‌شود (با استفاده از ارجاع معلق) که آن شیء داده وجود ندارد.

۴-۱-۴- مفاهیم اصلی اعلانها و کنترل نوع برای ساختمان داده ها

مفاهیم اصلی اعلانها و کنترل نوع مربوط به ساختمان داده ها همانند انواع داده اولیه می‌باشد. بعنوان مثال در مورد آرایه ها باید نوع داده ای آرایه تعداد ابعاد آرایه، اندیس هر یک از ابعاد، تعداد عناصر و نوع هر عنصر بایستی مشخص شود. کنترل نوع

[1] garbage
[2] dandling refrence

برای ساختمان داده پیچیده تر می‌باشد. زیرا عملیات انتخاب عنصر نیز بایستی در نظر گرفته شود. دو مسئله در این مورد وجود دارد.

۱. **وجود مولفه انتخابی** : نوع آرگومان انتخاب ممکن است درست باشد اما عنصر مورد نظر در ساختمان وجود نداشته باشد. اگر عملیات انتخاب مقدار چپ نادرستی را تولید کند اثر آن مشابه خطای کنترل نوع است. قبل از اینکه فرمول تعیین مقدار چپ یک عنصر ارائه شود بایستی مشخص کنیم که عنصر وجود دارد یا نه.
۲. **نوع عنصر انتخابی** : کنترل نوع ایستا تضمین می‌کند اگر عنصر وجود داشته باشد، نوع آن درست است.

۴-۱-۵- بردارها و آرایه ها

بردار ساختمان داده مرکب از تعداد ثابتی از عناصر همنوع است که بصورت یک دنباله خطی سازمان دهی شده است. برای دستیابی به عناصر بردار از ***اندیس*** استفاده می‌شود. اندیس یک مقدار صحیح یا شمارشی است که موقعیت یک عنصر را در بردار مشخص می‌کند.

۴-۱-۵-۱- بردارها

۱. **تعداد عناصر** : معمولاً توسط بازه ای که برای اندیس مشخص میشود تعیین می شود.
۲. **نوع هر عنصر** : تمام عناصر بردار از یک نوع می باشد.
۳. **اندیس برای انتخاب هر عنصر** : معمولاً بصورت بازه ای از مقادیر صحیح مشخص می شود بطوریکه اولین مقدار صحیح به اولین عنصر بین مقدار صحیح به اولین عنصر و... را مشخص می کنند .

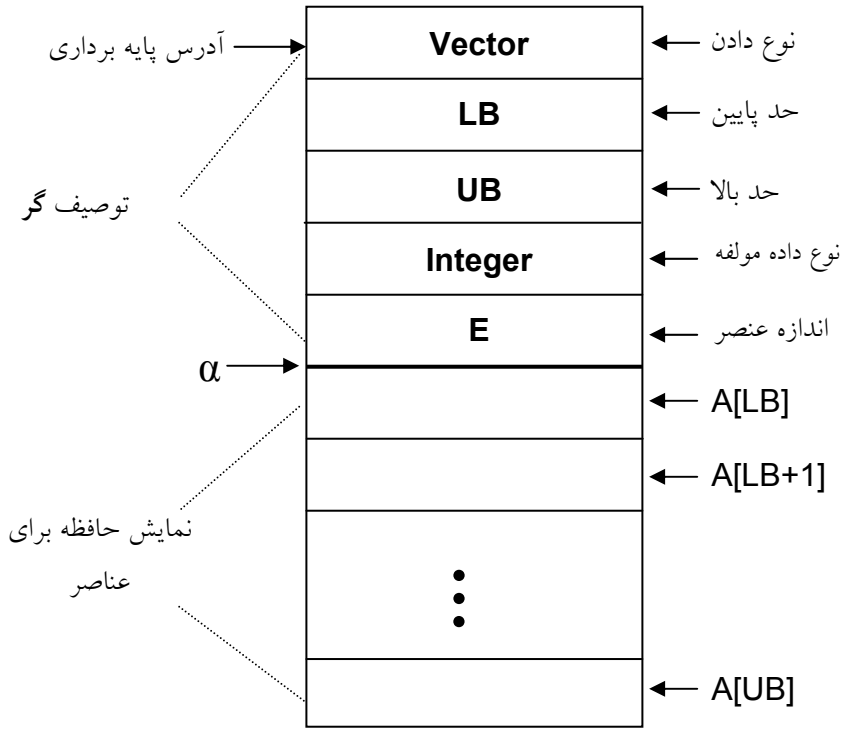
۴-۱-۵-۲- عملیات بر روی بردارها

یکی از عملیاتها عملیات **اندیس گذاری** می باشد، اندیس گذاری عملیاتی است که عنصری از بردار را انتخاب می‌کند. عمل اندیس گذاری مقدار چپ یا محل شیء داده را بر می‌گرداند.

عملیات دیگر بردار عبارتند از ایجاد و از بین بردن آنها، انتساب به عناصری از بردار و عملیاتی که اعمال محاسباتی را بر روی بردارهایی با طول یکسان انجام می دهند، مانند جمع بردار ، درج و حذف عناصر بر روی بردار امکان پذیر نیست و همچنین می توانیم مقدار هر عنصر را تغییر دهیم.

پیاده سازی : چون در روی بردارها معمولا عناصر همگن می باشند و اندازه بردار ثابت است بهترین نمایش حافظ استفاده از نمایش حافظه ترتیبی می باشد. در بردارها معمولاً طول ثابت. تعداد و موقعیت هر عنصر بردار در طول عمرش ثابت است. برای ذخیره سازی صفات بردار می توان از توصیف گر استفاده کرد، مخصوصا اگر اطلاعاتی تا زمان اجرا مشص نباشد. در صورتیکه لازم باشد بازه مقادیر اندیس محاسبه شده کنترل شود، حدود بالا و پایین بازه در توصیف گر ذخیره شوند.

شکل زیر نمایش حافظه برداری با توصیف گر کامل را نشان می دهد.



اگر اولین عنصر آرایه در محل α باشد مقدار چپ یک عنصر بردار بصورت زیر است:

$Lvalue(A[i]) = \alpha + (i-LB) \times E = (\alpha - LB \times E) + (I \times E)$

در این رابطه $(\alpha-LB\times E)$ مقدار ثابتی بوده و می توانیم آن را K در نظر بگیریم خواهیم داشت.

$Lvalue(A[i]) = K + I \times E$

در زبان C , LB همیشه برابر با صفر می باشد پس فرمول دستیابی در آرایه های زبان C برابر خواهد بود با .

$Lvalue(A[i]) = \alpha + I \times E$

که آرایه ، یک آرایه کاراکتری باشد فرمول دستیابی آن بصورت زیر خواهد بود.

$Lvalue(A[i]) = \alpha + I$

مبدأ مجازی : آدرس عنصری با اندیس صفر را مشخص می کنیم. فرمول دستیابی به صورت زیر در می آید:

$$Lvalue(A[0]) = (\alpha - LB \times E) + (0 \times E)$$
$$= (\alpha - LB \times E)$$
$$= K$$

K آدرس عنصر صفر بردار در صورت وجود می باشد. چون ممکن است عنصر صفر وجود نداشته باشد این آدرس ***مبدا مجازی*** (VO) نام دارد. بدین ترتیب فرمولی برای ساختن بردارها و تولید فرمول دستیابی ارائه می کند:

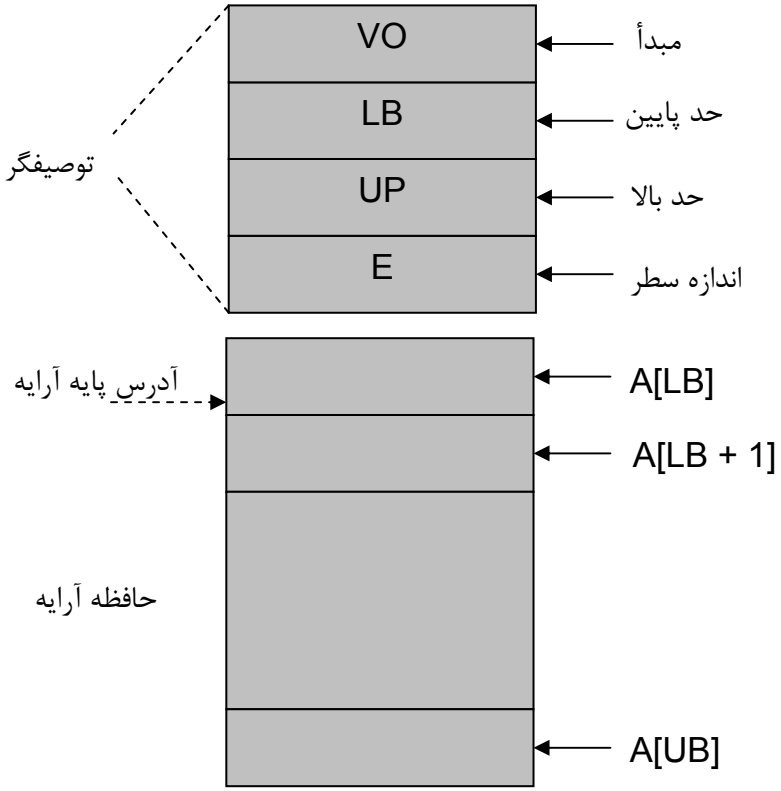
۱. **هنگام تخصیص حافظه برای بردار** حافظه ای برای n عنصر بردار به اندازه E و توصیف گری به اندازه D اختصاص بده $(D+N\times E)$ محل حافظه را تخصیص بده). آدرس محلی از حافظه را که اولین عنصر بردار ذخیره می شود α بنامید.

۲. **مبدا مجازی را محاسبه کن** : $VO = \alpha - LB \times E$

۳. **در دستیابی به عنصر بردار** مقدار چپ هر عنصر A[i] را بصورت زیر محاسبه می شود:

Lvalue(A[i]) = VO + i × E

اگر مبدأ مجازی در توصیف گر آرایه ذخیره شود آرایه واقعی لازم نیست با توصیف گر آن بطور پیوسته قرار داشته باشد. این شکل نشان می دهد که توصیفگر و آرایه پیوسته نیستند.



حافظه مجزا برای ناحیه های توصیفگر و عناصر

توصیف گرهای مربوط به پارامترهای آرایه می توانند به زیر برنامه ها ارسال شود ولی آرایه واقعی در جای دیگری ذخیره شده باشد.

نمایش های حافظه بصورت فشرده و غیر فشرده : در نمایش حافظه غیر فشرده عناصر یک بردار در حافظه فشرده می شوند و به این نقطه توجه نمی شود که هر عنصر باید از کلمه یا بایت آدرس پذیر شروع شود. این روش موجب صرفه جویی در حافظه می شود اما این حالت گران تمام می شود، زیرا نمی توانیم از فرمول دستیابی ساده ای که ارائه شد استفاده کرد بلکه محاسبات پیچیده ای نیاز دارد به همین دلیل عمدتاً از حالت غیر فشرده استفاده می شود و هر عنصر در مرز یک واحد آدرس پذیر قرار می گیرد و ممکن است بین هر جفت عنصر، حافظه استفاده نشده ای باقی بماند. با گسترش ماشین هایی که واحد آدرس دهی آنها بایت است این موضوع چندان مهم نیست.

عملیات بر روی کل بردار : عملیاتی که بر روی کل بردار انجام می شود از نمایش حافظه ترتیبی برای بردار استفاده می کند در انتخاب دو بردار با صفات یکسان محتویات بلوکی از حافظه که نشان دهنده یک بردار می باشد در بلوک دیگری از حافظه که نشان دهنده بردار دیگری است کپی می شود.

۴-۱-۵-۳- آرایه های چند بعدی

تفاوت آرایه های چند بعدی در بازه اندیس هر بعد است.

پیاده سازی: برای پیاده سازی آرایه های چند بعدی می توان با استفاده از ارایه ها با ابعاد کمتر این کار را انجام داد. به عنوان مثال برای پیاده سازی ماتریس می توان آن را برداری از بردارها در نظر گرفت. باید توجه داشته باشید که تعداد عناصر تمام بردارهای فرعی[1] و نوع همه ی عناصر باید یکسان باشد.

می توان ماتریس را بصورت ***ستون هایی از سطرها*** یا بصورت ***سطرهای از ستون ها*** در نظر بگیرید که اغلب بصورت ستونی از سطرها در نظر گرفته می شود. ماتریس بصورت برداری در نظر گرفته می شود که هر عنصر آن یک بردار فرعی است و یک سطر از ماتریس اصلی را نمایش می دهد. این نمایش را ترتیب سطری[2] می نامند. در نمایش ستونی ماتریس بصورت یک سطر از ستون ها در نظر گرفته می شود نمایش حافظه برای آرایه چند بعدی از بردار پیروی می کند.

به عنوان مثال، ماتریس x را در نظر بگیرید:

$$\mathbf{x} = \begin{bmatrix} \mathbf{3} & \mathbf{4} & \mathbf{7} \\ \mathbf{6} & \mathbf{2} & \mathbf{5} \\ \mathbf{1} & \mathbf{3} & \mathbf{8} \end{bmatrix}$$

در نمایش سطری، عناصر آرایه x به ترتیب زیر ذخیره می شوند:

$$\underbrace{3\ ,\ 4\ ,\ 7}_{\text{سطراول}}\ ,\ \underbrace{6\ ,\ 2\ ,\ 5}_{\text{سطردوم}}\ ,\ \underbrace{1\ ,\ 3\ ,\ 8}_{\text{سطرسوم}}$$

در نمایش ستونی،ابتدا ستون اول، سپس ستون دوم، ...و سپس ستون آخر ذخیره می شوند. بنابر این، ماتریس x در نمایش ستونی به صورت زیر ذخیره خواهد شد:

$$\underbrace{3\ ,\ 6\ ,\ 1}_{\text{ستون اول}}\ ,\ \underbrace{4\ ,\ 2\ ,\ 3}_{\text{ستون دوم}}\ ,\ \underbrace{7\ ,\ 5\ ,\ 8}_{\text{ستون سوم}}$$

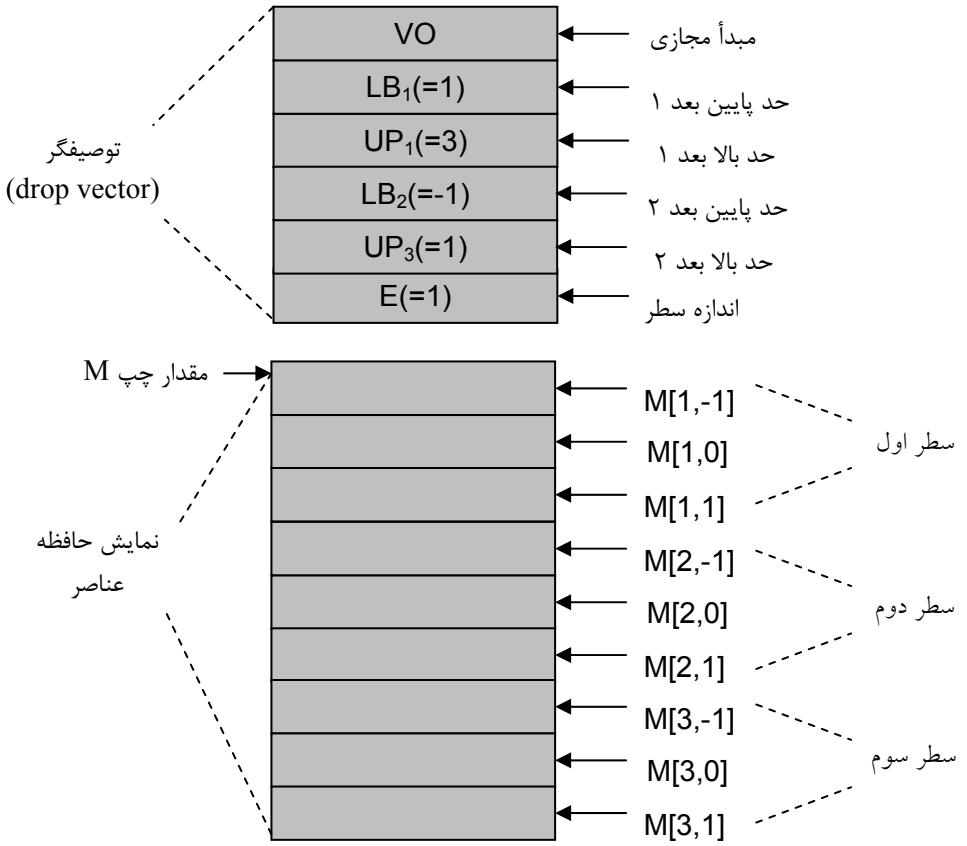
فرمول دستیابی به عنصر A[i,j] بصورت زیر خواهد بود.

$$Lvalue(A[i,j]) = \alpha + (i - LB_1) \times S + (j - LB_2) \times E$$

بنابراین خواهیم داشت :

$$S = (UB_2 - LB_2 + 1) \times E$$
$$VO = \alpha - LB_1 \times S - LB_2 \times E$$
$$Lvalue(A[i,j]) = VO + i \times S + j \times E$$

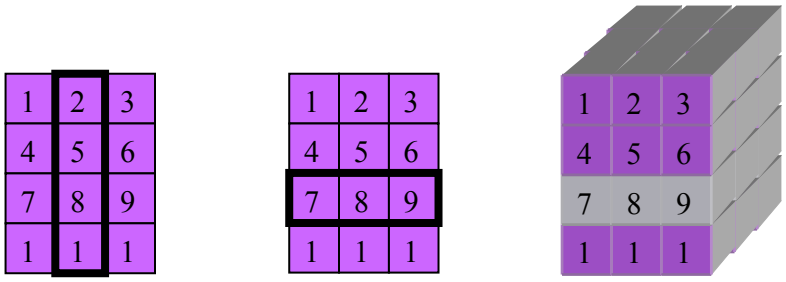
[1] subvector
[2] rowmajor



نمایش حافظه برای آرایه دو بعدی

۴-۱-۵-۴- برش آریه[1]

برش بخشی از آرایه است که خودش یک آرایه است برای پیاده سازی آن عمدتاً از توصیفگر استفاده می شود. استفاده از توصیفگر منجر به پیاده سازی کارآمد برش ها می شود.



۴-۱-۵-۵- آرایه های شرکت پذیر[2]

[1] slice
[2] associative arrags

یکی از روش ها برای دسترسی به عناصر آرایه استفاده از اندیس می باشد اما روش دیگر این است که از نام به عنوان انـدیس استفاده شود یعنی از طریق نام بتوان به اطلاعات دست یافت چون دنباله ای از اساس نا محدود هستند استفاده از نوع شمارشـی ممکن نیست بلکه مجموعه ای از اسامی به عنوان مجموعه شمارشی به کار گرفته می شود اگـر نـام جدیـدی اضـافه شـود ایـن شمارش گر افزایش می یابد چنین آرایه ای را آرایه شرکت پذیر می نامند.

۴-۱-۶- رکوردها

رکورد ساختمان داده مرکب از تعداد ثابتی از عناصر و از انواع مختلف است رکوردها و بردارها ساختمان داده های ***خطـی*** بـا ***طول ثابت*** هستند اما رکوردها از دو جهت متفاوت هستند.

۱. عناصر رکورد می تواند ناهمگن و از انواع مختلف باشد.
۲. عناصر رکورد دارای نام می باشند یعنی اینکه عناصر رکورد توسط اسامی سمبلیک مورد استفاده قرار می گیرند.

صفات رکورد عبارتند از :

۱. تعداد عناصر
۲. نوع هر عنصر
۳. نامگذاری هر عنصر

عناصر رکورد را معمولاً **فیلد** و نام هر عنصر را نام فیلد گویند. رکورد را معمولاً **ساختمان** نیز می نامند.

عملیات بر روی رکوردها: عملیاتی که بر روی رکوردها انجام می شود اندکند. به عنوان مثال عملیات بر روی هر عنصر کـه اینگونه عملیات ها وابسته به نوع هر عنصر می باشند. انتخاب هر عنصر مهم ترین عمل در رکورد است این کار همانند اندیس در آرایه ها می باشد ولی با این تفاوت که اندیس در رکورد نام یک عنصر است، یعنی یک مقدار محاسبه شده ای نیـست. عملیـاتی که بر روی کل رکورد انجام می شود اندک است و تنها assignment یک رکورد روی رکورد دیگری (با ساختار یکسان) است.

پیاده سازی: برای پیاده سازی رکوردها نمایش حافظه ای که استفاده می شود یک بلوک از حافظه است که عناصر در آن به ترتیب ذخیره می شود. معمولاً descriptor زمان اجرا برای هر یک از عناصر رکورد در نظر گرفته شود ولی برای کل کل رکورد نیازی به descriptor نیست. مکان هر مولفه به شکل زیر آدرس دهی می شود.

$$Lvalue(R.I) = \alpha + \sum_{J=1}^{I-1}(size \quad of \quad R.J)$$

α آدرس پایه بلک حافظه است که Rرا نشان می دهد و R.J عنصر J ام است. چون اندازه هر عنصر فـرق مـی کنـد نیـاز بـه جمع بندی دارد.

مثال :

```
struct student {
    int id;
   int age;
   float ave;
   char name[20];
}
studentd st, sp;
```

نمایش حافظه رکورد student

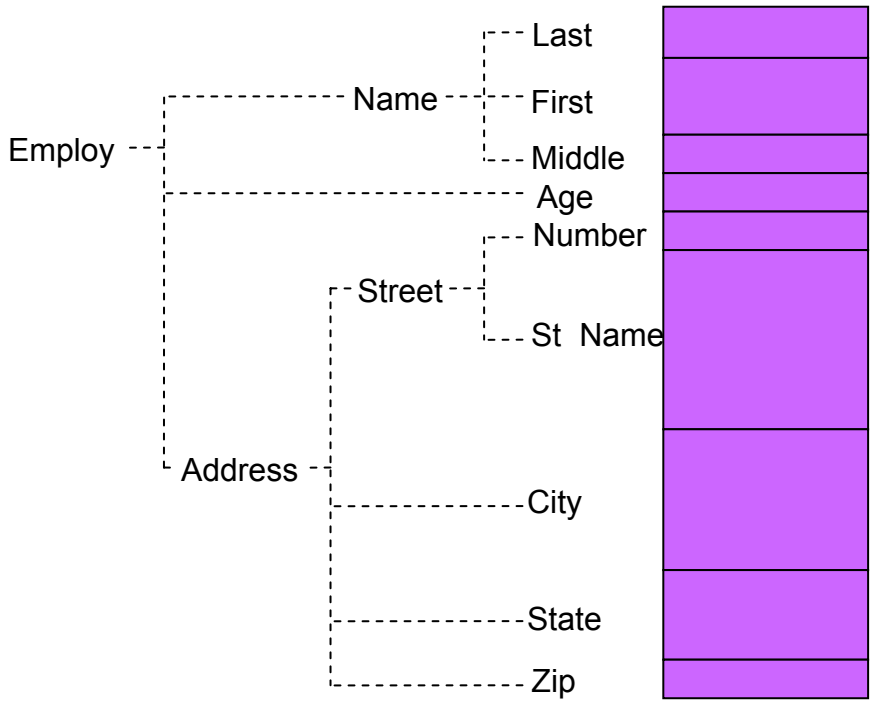
id	age	ave	name
100	18	15.5	Ali

۴-۱-۶-۱- رکوردها و آرایه های با عناصر ساختاری

در زبان هایی که رکوردها و آرایه ها به عنوان انواع داده ای هستند این امکان فراهم است که عناصری که از دو نوع مختلف با عناصری از انواع داده ترکیب شوند. عناصر رکورد می تواند آرایه یا رکورد دیگری باشد و این روند می تواند تا چند سطح ارائه داشته باشد و ساختار سلسله مراتبی را ایجاد کند. انتخاب عناصر مستلزم دنباله ای از انتخاب ها با شروع از آدرس پایه ساختمان اصلی و محاسبه یک آفست برای یافتن محل عنصر اولین سطح و سپس محاسبه یک آفست از این آدرس پایه برای یافتن عناصر دومین سطح و غیره است.

مثال زیر نمونه ای از این حالت را در زبان PL/I نشان می دهد.

```
1 Employ,
    2 Name,
        3 Last CHARACTER(10),
        3 First CHARACTER(15),
        3 Middle CHARACTER(1),
    2 Age FIXED(2),
    2 Address
        3 Street,
            4 Number FIXED(5),
            4 St_Name CHARACTER(20),
        3 City CHARACTER(15),
        3 State CHARACTER(10),
        3 Zip FIXED(5);
```



نمایش حافظه برای رکورد چند سطحی در PL/I

۴-۱-۶-۲- رکوردهای طول متغیر

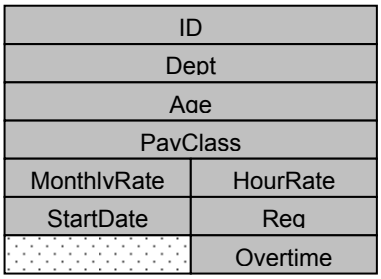
نوعی است که در زمان های مختلف اجرای برنامه ، ساختارهای متفاوتی می تواند داشته باشد. در اینگونه رکوردها یک عنصر ممکن است در یک زمان وجود داشته باشد و در زمان دیگر وجود نداشته باشد یعنی منظور این است که در زمان اجرا قبل از دستیابی به یک عنصر باید کنترل کرد تا مشخص شود که عنصر وجود دارد یا خیر. این مشکل را می توان به صورتهای زیر حل کرد:

۱. **کنترل پویا**: عنصر برچسب را می توان در زمان اجرا،قبل از دستیابی به یک عنصر کنترل کرد تا مشخص شود که آن عنصر وجود دارد یا خیر. اگر بر چسب مقدار مناسبی داشته باشد آن عنصر قابل دستیابی است وگرنه خطای زمان اجرا اتفاق می افتد و پردازش استثنا فراخوانی می شود.

۲. **کنترل انجام نشود**: طراحی زبان ممکن است طوری باشد که تعریف رکورد طول متغیر بدون عنصر بر چسب صورت گیرد بطوریکه در زمان اجرا نتوان کنترل را انجام داد. در این زبانها فرض می شود که انتخاب عنصری از رکورد طول متغیر همواره معتبر است. اما هنگام دسترسی ممکن است ناخواسته مقدار دیگری دستیابی شود. در زبان C این نوع با نام union شناخته می شود.

پیاده سازی: پیاده سازی رکورد طول متغیر ساده تر از کاربرد صحیح آن است. در حین ترجمه با توجه به اعلان رکورد طول متغیر، اندازه حافظه مربوط به ساختارهای مختلف رکورد را محاسبه کرده ، حافظه ای که به اندازه بزرگترین ساختار ممکن تخصیص می یابد. بطوریکه آدرس تمام فیلدهای متغیر از یک نقطه شروع می شود (یعنی حافظه مشترک) توصیف کاملی از هر ساختار رکورد طول متغیر باید در زمان ترجمه وجود داشته باشد. مثال زیر نمونه ای از این ساختارها را در زبان pascal همراه با نمایش حافظه آن نشان می دهد.

```
Type PayType = (Salaried, Hourly);
Var Employee: record
        ID: integer;
        Dept: array [1..3] of char;
        Age: integer;
        case PayClass: PayType of
          Salaried: (MnthlyRate: real;
                StartDate: integer);
          Hourly: (HourRate: real;
                Reg: integer;
                Overtime: integer)
        end
```



نمایش حافظه برای رکورد طول متغیر

۴-۱-۷- لیست

لیست ساختمان داده مرکبی از دنباله مرتب ساختمان داده ها می باشد.

۴-۱-۷-۱- مشخصات و نحو

لیست مانند بردارها حاوی دنباله مرتبی از اشیاء می باشد یعنی می توان به اولین عنصر، دومین عنصر و ... مراجعه کرد اولین عنصر لیست را معمولا راس می نامند .

لیستها از ۲ جهت با بردارها متفاوتند:

- طول لیست به ندرت ثابت است.
- لیستها به ندرت ممکن هستند.

زبانهایی که از لیستها استفاده می کنند نوعاً چنین داده ای را به صورت ضمنی و بدون صفات صریع برای اعضای لیست تعریف می کنند .

۴-۱-۷-۲- پیاده سازی

مدیریت حافظه ی منظمی که برای بردارها و آرایه ها مفید است برای لیست مفید نیست. و معمولاً برای آن از سازمان مدیریت حافظه پیوندی استفاده می شود. در لیست به سه فیلد از اطلاعات نیاز داریم یک فیلد نوع و دو فیلد اشاره گر لیست . اگر فیلد نوع atom باشد آنگاه فیلدهای باقی مانده، توصیف گرهایی می باشند که این اتم را ، توصیف می کنند اگر فیلد از نوع یک لیست باشد ، اشاره گر اول راس لیست و اشاره گر دوم ، انتهای لیست است.

شکل زیر ، ساختار گره های لیست را نشان می دهد.

Type	Head Field	Toil Field

شکل های گوناگون لیستها :

۱- پشته ها و صف ها

۲-درختها

۳- گرافهای جهت دار

۴- لیستهای خاصیت

۴-۱-۸- مجموعه ها

مجموعه ، شی داده ای است که شامل مقادیر نا مرتب و مجزا است عملیات اصلی روی مجموعه ها ، عبارتند از ، عضویت ، درج و حذف یک مقدار و اجتماع، اشتراک و تفاضل مجموعه ها .

۴-۱-۸-۱- پیاده سازی

مجموعه ها ، ساختمان داده ای می باشند که عناصر مرتب را نشان می دهند. مجموعه مرتب ، لیستی است که مقادیر تکراری از آن حذف شده اند. برای پیاده سازی مجموعه ها، دو نمایش حافظه ای وجود دارد

۱. نمایش بیتی مجموعه ها
۲. نمایش درهم سازی مجموعه ها

در نمایش مجموعه ها ، مجموعه جهانی به طور دلخواه مرتب می شود می توان هر مجموعه ای را با یک رشته بیتی به طول (N) نشان د اد. اگر e_i در مجموعه وجود داشته باشد i امین بیت این رشته برابر یک و گرنه برابر صفر قرار خواهد گرفت. برای درج یا حذف یک عنصربه مجموعه کافی است بیت مربوطه را از صفر به یک یا از یک به صفر تغییر دهیم .برای تعیین عضویت می توان بیت مربوطه را بررسی کرد.

برای اجتماع از عملیاتOR ، اشتراک از عملیات AND و برای تفاضل رشته اول با مکمل رشته دوم AND خواهد شد.

نمایش درهم سازی مجموعه ها :

از این روش زمانی استفاده می شود که مجموعه جهانی بزرگ باشد. در این روش تست عضویت به سادگی و با کارآیی خوبی انجام می شود. اما اجتماع ، اشتراک و تفاضل به صورت دنباله ای از تستهای عضویت، درج و حذف تک تک عناصر انجام می شود.

مثال :

$$R(\underset{1}{A},\underset{2}{B},\underset{3}{C},\underset{4}{D},\underset{5}{E},\underset{6}{F},\underset{7}{G},\underset{8}{H})$$

$$R_1(A,C,F,H) \rightarrow R_1(1,2,6,8)$$

$$R_2(D,E,F,H) \rightarrow R_2(4,5,6,8)$$

$$R_1 \cup R_2 = (1,3,6,8,4,5)$$

$$R_1 \cap R_2 = (6,8)$$

$$R_1 - R_2 = (1,3)$$

۴-۱-۹- تکامل مفهوم نوع داده

مفهوم اولیه نوع داده ، نوع را به صورت مجموعه ای از مقادیر تعریف می کند که یک متغیر می تواند آنها را بپذیرد متغیر را می توان از نوع مورد نظر اعلان کرد. برای انواع اولیه ای مثل حقوقی و صحیح زبان برنامه سازی امکانی را برای اعلان متغیرهای

آن نوع و عملیاتی را برای آنها تدارک می بینند. نمایش حافظه ای مربوط به مقادیر حقیقی و صحیح کاملاً بسته بندی شده است. یعنی از دید برنامه نویس پنهان می باشد.

برنامه نویسی بدون اینکه از جزئیات نمایش حافظه ای آن اطلاع داشته باشد از اشیاء داده ای آنها استفاده می کند برنامه نویس فقط نام نوع و عملیاتی را برای دستکاری آن نوع فراهم می بیند.

۴-۱-۹-۱- انتزاع داده ها

برای بسط مفهوم بسته بندی به داده هایی که توسط برنامه نویس تعریف می شود نوع داده انتزاعی را به صورت زیر تعریف می کنیم:

۱. **مجموعه ای از اشیای داده** ، معمولا با استفاده از یک یا چند تعریف نوع.
۲. **مجموعه ای از عملیات** انتزاعی بر روی آن انواع داده .
۳. **بسته بندی** تمام آنها به طوری که کاربر نوع جدید نتواند اشیای داده ای از آن نوع را به جز از طریق عملیاتی که برای آن تعریف شده است دستکاری کند.

۴-۱-۱۰- پنهان سازی اطلاعات

معمولا نوشتن برنامه های بزرگ ،خارج از درک یک نفر می باشد به همین دلیل برای نوشتن برنامه بزرگ باید از استراتژی تقسیم و حل استفاده کرد برنامه به مجموعه ای از قطعات بنام ماژول تقسیم می شود هر ماژول مجموعه محدودی از عملیات را بر روی مقدار محدودی از داده ها انجام می دهد و به این ترتیب کنترل منطقی برای فرآیند طراحی برنامه وجود خواهد داشت. طراحی ماژول معمولا به دو روش انجام می شود.

۱. ماژولها ، تجزیه تابعی برنامه را نشان می دهند .
۲. ماژولها ، تجزیه داده ای برنامه را نشان می دهند

پنهان سازی اطلاعات اصطلاح مهمی در طراحی انتزاع های برنامه نویسی است هر قطعه از برنامه باید تا آنجائی که ممکن است اطلاعات را از کاربران آن قطعه پنهان کنندزبان برنامه سازی، انتزاع را به دو روش پشتیبانی می کند.

۱. با تدارک کامپیوتر مجازی که کاربرد آن ساده تر و قدرت آن بیش از کامپیوتر سخت افزاری است.
۲. زبان امکاناتی را فراهم می کند که برنامه نویسی می تواند انتزاعها را به وجود آورد.

زیر برنامه ها ، کتابخانه های زیر برنامه ، تعاریف نوع ، کلاسها، و پکیج ها بعضی از امکانات می باشند که در زبانهای مختلف برای پشتیبانی از انتزاعهای برنامه نویسی وجود دارد. وقتی اطلاعات در یک انتزاع بشته بندی شده اند معنایش این است که :

۱- لازم نیست کاربر از اطلاعات مخفی ، اطلاع داشته باشد که بتواند از آن انتزاع استفاده کند .
۲- در صورت لزوم نمی تواند مستقیماً اطلاعات مخفی را دستکاری کند، بسته بندی اصلاح برنامه را آسان می کند.

زیر برنامه ها، مکانیزم بسته بندی را شکل می دهند که تقریباً در هر زبانی وجود دارد. بسته بندی به طراحی زبان مربوط می شود ، یک انتزاع وقتی به خوبی بسته بندی می شود که زبانها ، دستیابی به اطلاعات مخفی شده در آن انتزاع را مجاز ندانند.

۴-۲- بسته بندی با زیر برنامه ها

۴-۲-۱- زیر برنامه ها و عملیات انتزاعی

همانند عملیات اولیه تعریف زیر برنامه دو بخش دارد : مشخصات و پیاده سازی .

مشخصات زیر برنامه: مشخصات ریز برنامه مثل عملیات اولیه است این مشخصات شامل موارد:

۱. **نام ریز برنامه**
۲. **امضای زیز برنامه** ، که تعداد آرگومانها ، ترتیب آنها ، نوع هر یک از آنها و تعداد نتایج ، ترتیب و نوع آنها را مشخص می کند
۳. **فعالیتی که توسط ریز برنامه انجام می شود** ریز برنامه یک تابع ریاضی را نشان می دهد که هر مجموعه خاصی از آرگومانها را به مجموعه خاصی از نتایج نگاشت می کند اگر زیربرنامه یک مقدار را بر گرداند زیربرنامه تابع نامیده می شود .

پیاده سازی زیر برنامه: زیر برنامه با استفاده از ساختمان داده ها و عملیاتی پیاده سازی می شود که توسط خود زبان برنامه سازی ارائه می شود. پیاده سازی توسط بدنه زیر برنامه تعریف می شود که متشکل از اعلان داده های محلی است که ساختمان داده مورد استفاده زیز برنامه را تعریف می کند و شامل دستوراتی است که عملکرد زیربرنامه را مشخص می کند .

تابع فاکتوریل خودش را فراخوانی می کند داده های تابع Fact ، یکی n و دیگری result یا نتیجه است در هر بار اجرا تابع Fact دارای سابقه فعالیت می شود و هر بار داده ای که می دهیم نتیجه را به ما بر می گرداند.

```
int fact (int   n)
{
   if (n==1) return 1;
   else  return fact (n-1)*n;
}
```

۴-۲-۲- تعریف و فراخوانی زیربرنامه

تعریف زیر برنامه ، خاصیت ایستای یک برنامه می باشد در حین اجرای برنامه اگر زیر برنامه اجرا شود ***سابقه فعالیتی*** از آن زیر برنامه ایجاد خواهد شد و وقتی زیر برنامه خاتمه یافت سابقه فعالیت آن نیز از بین می رود و اگر فراخوانی دیگری صورت بگیرد سابقه فعالیت جدیدی ایجاد خواهد شد .

تعریف زیربرنامه ***قالبی*** برای ایجاد سابقه فعالیت در حین اجرا می باشد سابقه فعالیت زیر برنامه دارای طول عمر می باشد که با فراخوانی زیر برنامه شروع شده و تا از بین رفتن آن ادامه دارد. اصطلاح شی داده را برای سابقه فعالیت زیربرنامه بکار نمی بریم.

اشیای داده ای در حین اجرای برنامه ایجاد می شوند در حین ورود به زیربرنامه یا توسط عملیاتی مثل mallocبرای ساختن سابقه فعالیتی خاص از الگوی زیربرنامه کل الگو باید در ناحیه جدیدی از حافظه کپی شود . اما به جای کپی کامل بهتر است الگو به دو بخش تقسیم شود .

۱. بخش ***ایستا*** که سگمنت کد[1] نام دارد و حاوی سوابق و کد اجرایی است .

۲. بخش ***پویا*** که رکورد فعالیت[2] نام دارد و شامل پارامترها، نتایج تابع و داده های محلی و ناحیه حافظه موقت ، نقاط برگشت و پیوندهایی برای مراجعه به متغیرهای غیر محلی است ساختار این بخش برای تمام سوابق فعالیت زیر برنامه یکسان است . اما مقادیر متفاوتی در آنها وجود دارد.

اندازه و ساختار رکورد فعالیت مورد نیاز برای یک زیر برنامه می تواند در زمان ترجمه تعیین شود . دستیابی به عناصر با استفاده از آدرس ***پایه*** و ***آفست*** امکان پذیر است .

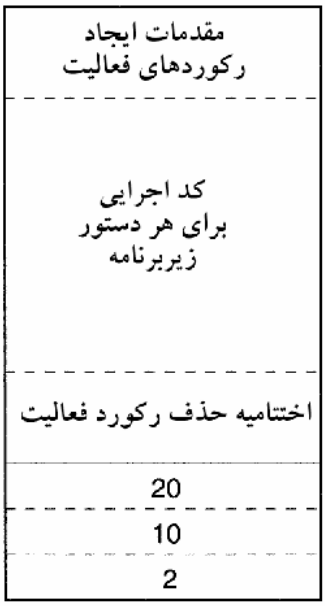
بخش ایستا بخشی است که تغییر پیدا نمی کند شامل کدها و بخش پویا همیشه تغییر می کند تعریف زیر برنامه ها تقریباً شبیه structure هاست.

تعریف زیربرنامه به عنوان اشیای داده ها: برنامه منبع توسط کامپایلر به شکل اجرایی در می آید با تعریف زیر برنامه ها می توان مثل اشیای داده زمان برخورد کرد. ترجمه، عملیاتی می باشد که تعریف زیربرنامه را به شکل رشته کاراکتری گرفته، شی داده زمانی اجرا را تولید می کند. که این تعریف را نمایش می دهد.

اجرا عملیاتی است که تعریفی به شکل زمان اجرا را گرفته ، سابقه فعالیتی را از آن ایجاد می کند و آن سابقه فعالیتی را اجرا می نماید. به عنوان مثال تابع زیر تعریف شده است.

سگمنت که زیربرنامه و همچنین رکورد فعالیت آن از طریق تعریف ریزبرنامه به صورت زیر نتیجه خواهد شد .

```
float f(float x ,int y)
{
   const int value = 2;
   float m[20];
   int n;
   .
   .
   .
   return(10 * x + m[n])
}
```



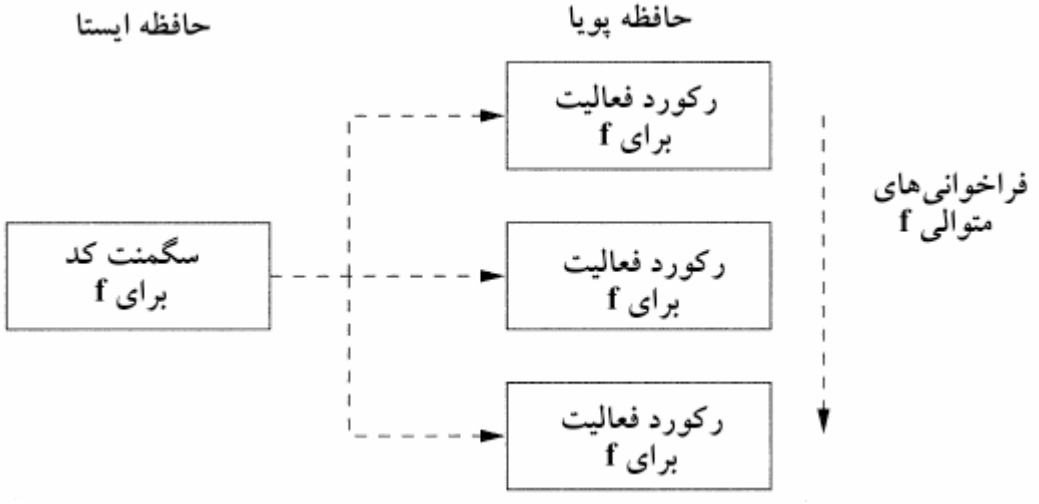
سگمنت کد زیربرنامه FN

رکورد فعالیت FN (الگو)

سابقه فعالیت زیر برنامه.

[1] Code segment
[2] Activation record

در هر بار فراخوانی زیربنامه بخش پویای زیر برنامه ایجاد خواهد شد یعنی در هر بار فراخوانی یک رکورد فعالیتی جدید ایجاد خواهد شد اما بخش ایستای زیربرنامه برای فراخوانی های متعدد ثابت خواهد بود شکل زیر چند رکورد فعالیت با سگمنت که مشترک را برای ریز برنامه F نشان می دهد .



چندرکورد فعالیت با سگمنت کد مشترک.

۴-۳- تعریف نوع

در تعریف نوع ، نام تعیین می شود و اعلانی وجود دارد که ساختار دسته ای از اشیای کلاس را توصیف می کند به این ترتیب نام نوع به عنوان نام دسته ای از اشیای داده ای محسوب می شود هر وقت به اشیای داده ای از آن ساختار نیاز باشد بجای تکرار توصیف ساختمان داده کافی است نام نوع ارائه شود. در اغلب زبانهای جدید امکان تعریف نوع گنجانده شده است به عنوان مثال تعریف زیر در زبان c:

```
strict rational {
    int num1;
    int num2;
};
```

استفاده از نوع جدید در c مستلزم استفاده از عبارت Struct در اعلان اشیای داده ای است . به عنوان مثال

```
struct   rational A,B,C ;
```

اما اینکار یکی از اصول انتزاع را نقض می کند تمایل داریم نوع داده جدید به عنوان توسعه ای از زبان برنامه سازی وجود داشته باشد. و از نظر نحوی و معنایی مشابه انواع داده اولیه باشد. اما استفاده از کلمه کلیدی struct برای اعلان متغیرهای این ویژگی را نقض می کند تعریف نوع ساختار داخلی آن نوع از اشیای داده ای را پنهان می کند.

پیاده سازی : اطلاعات موجود در اعلان متغیرها برای تعیین نمایش حافظه ای اشیاء و اهداف مدیریت حافظه و کنترل نوع بکار می رود . اعلان ها در زمان اجراء وجود ندارند بلکه برای ایجاد مناسب اشیاء در زمان اجراء بکار می روند. مترجم زبان اطلاعات حاصل از تعریف نوع را در جدولی قرار می دهد و با استفاده از این اطلاعات که اجرایی مناسب را برای تنظیم و دستکاری اشیای مطلوب تولید می کند .

زبان c از هم ارزی نام استفاده می کند از هم ارزی ساختار استفاده می کند و در زبان c ساختار بدون نام می توانیم تعریف کنیم در زبان c++ از ساختار و هم ارزی نام استفاده می شود.

۴-۳-۱- هم ارزی نوع

کنترل نوع چه به صورت ایستا و چه به صورت پویا مقایسه ای بین نوع آرگون واقعی و نوع داده ای است که عملیات انتظار آنرا دارد اگر انواع یکسان باشند آرگون پذیرفته می شود و عملیات ادامه می یابد و اگر یکسان نباشد یا خطا محسوب می شود و یا تبدیل ضمنی صورت می گیرد دو راه حل برای مشخص کردن تساوی نوع وجود دارد.

۱. هم ارزی نام
۲. هم ارزی ساختاری

هم ارزی نام : دو نوع داده وقتی با هم هم ارز می باشند که نام یکسان داشته باشند .

معایب هم ارزی نام :

۱. هرشی که در انتساب بکار می رود باید دارای نام باشد یعنی انواع داده بی نام وجود ندارد.
۲. یک تعریف نوع باید در سراسر برنامه قابل دسترسی باشد زیرا نوع شی داده ای که از طریق زنجیره ای از زیر برنامه ها بصورت آرگومان انتقال داده شد نمی تواند دوباره در هر زیر برنامه تعریف شود.

هم ارزی ساختاری : دو نوع داده وقتی هم ارز هستند که اشیای داده آنها عناصر داخلی یکسانی داشته باشند معمولا معنای عناصر داخلی یکسان این است که تمام اشیای داده از یک نمایش حافظه استفاده کنند .

معایب هم ارزی ساختاری عبارت است از :

۱. وقتی دو نوع از نظر ساختاری هم ارز باشند چند پرسش مطرح می شود به عنوان مثال در مورد رکوردها آیا اسامی فیلدها باید یکسان باشند یا اگر تعداد و نوع فیلد ها و ترتیب آنها یکسان باشد کافی است بازه اندیس آرایه ها باید یکسان باشد و یا ....
۲. دو متغیر ممکن است به طور تصادفی از نظر ساختار یکسان باشند حتی اگر برنامه نویسی آنها را به عنوان انواع جداگانه ای تعریف کرده باشد هم ارزی تصادفی انواع ممکن است آنچه را که کنترل نوع ایستا بدست آمده است خراب کند زیرا بسیاری از خطاهای نوع تشخیص داده نشود .
۳. تعیین هم ارزی ساختار در مورد انواع پیچیده هزینه ترجمه دارد . در زبان c از هم ارزی ساختاری استفاده می شود ولی در زبان c++ از هم ارزی نام استفاده می شود .

```
struct  test1
{
  int A;
  int B;
};
```

```
struct  test2
{
  int K;
  int M  ;
};
```

```
struct  test1 T1;
struct  test2 T2;
```

T1 و T2 که از نظر ساختاری با هم برابر هستند یعنی هر دو، دو عنصر از نوع int دارند ولی از نظر هم ارزی نام با هم یکسان نیستند T1 از نوع test1 و T2 از نوع test2 می باشد.

فصل پنجم

۵- وراثت

۵-۱- نگاهی دوباره به انواع داده انتزاعی

با توجه به مطالبی که در فصل قبل گفته شد ***نوع داده انتزاعی*** ، نوع داده جدیدی می باشد که توسط برنامه نویس تعریف می شود و شامل موارد زیر می باشد:

۱. نوع داده ای که توسط برنامه نویسی تعریف شد .
۲. مجموعه ای از عملیات انتزاعی بر روی اشیایی از آن نوع
۳. بسته بندی اشیای آن نوع به طوری که کاربر آن نوع آن اشیاء را بدون استفاده از این عملیات آن را دستکاری کند.

در زبان c ساختار داخلی اشیای داده نوع بسته بندی شده نیست هر زیر برنامه ای که می تواند متغیری را از نوع جدید اعلان کند اجازه دارد به هر عنصر از نمایش آن نوع دستیابی داشته باشد.

مفهوم بسته بندی تعریف نوع داده انتزاعی این دستکاری را غیر ممکن می کند بطوری که فقط زیربرنامه هایی که به عنوان بخشی از خود نوع هستند می دانند اشیای داده آن نوع چگونه نمایش داده شوند.

```
class student
{
  private:
    int  ID;
    char name[20];
    char adderess[50];
  public:
    void setID(int T)
    int getID;()
}
```

زبانهایی مثل ADA ، java , c++ , small tak و غیره از بسته بندی نوع داده انتزاعی پشتیبانی می کنند. داخل این تعاریف بخش private ، عناصری را تعریف می کند که فقط توسط توابع داخل کلاس (پکیج) قابل دسترس می باشند. اما عناصر بخش public هم از طریق توابع عنصر کلاس و هم ارز خارج از کلاس قابل تغییر می باشد.

پیاده سازی: پکیج بسته بندی را برای تعریف نوع و زیربرنامه فراهم می کند اولین اثرش محدود کردن قابلیت مشاهده اسامی اعلان شده در پکیج است. به طوری که کاربران این نوع داده انتزاعی نمی توانند به عناصر داخلی این تعریف دستیابی داشته باشند.

هر پکیج قابل دو بخش شامل دو بخش می باشد:

۱. مشخصات

۲. پیاده سازی

بخش مشخصات پکیج تمام داده ها، انواع و زیربرنامه ها را تعریف می کند که توسط رویه هایی کـه در پکـیج هـای دیگـری اعلان شده اند قابل مشاهده اند. پیاده سازی رویه های اعلان شده در بخش مشخصات پکیج قرار داده می شود .

۵-۱-۱- کلاس در C++

C++ علاوه بر ویژگی زبان دستوری C،یک زبان برنامه سازی شیء گرا نیز هست.نوع داده انتزاعـی در C++ بـا اسـتفاده از کلاس ها ایجاد می شوند . شکل کلی کلاس در C++ به صورت زیر است:

```
class نام کلاس {
  Private:
      داده ها و توابع اختصاصی
  Public:
      داده ها و توابه عمومی
  Protected:
      داده های محافظت شده
};
```

نمونه ای از کلاس در C++ به صورت زیر است:

```
class circle {
    int radius;
    float area;
    float perime;
public:
    circle();
    ~circle();
    void get_radius();
    void calculator();
    void display();
};
```

داخل بخش public تابع همنام با نام کلاس تابع سازنده و تابع همنام به نام کلاس که در ابتدای آن کاراکتر ~ قرار داشته باشد تابع مخرب خوانده می شود تعریف هر کلاس از دو بخش تشکیل می شود بخش مشخصات که مشخصات کـلاس را نـشان می دهد و بخش پیاده سازی که در حقیقیت عملکرد توابع کلاس را نشان می دهد.

```
class Stack
{
  int *x;
  int myTop;
  int max;
public:
  stack(int);
  int empty();
  int full();
  void push(int a);
  void pop();
  int top();
}; // end of class
Stack :: stack (int n)
{
  myTop = -1;
  max = n;
  x = new int{n};
}
void stack :: empty()
{
  return myTop == -1;
}
void stack :: full()
{
  return myTop >= max;
}
```

```
void stack :: push (int a)
{
   if(full())
       return;
   x{++myTop} = a;
}
void stack :: pop()
{
  if(!empty())
      myTop--;
}
int stack :: top()
{
  if(empty())
      return -1;
  return x[myTop];
}
int main()
{
  stack s1(10);
  stack s2(20);
  s1 .push(5);
  s2 .push(10);
  ...
}
```

در زبان C++ کامپایلر با دیدن تعریف یک کلاس آن را به صورت زیر به تعریف ساختمان تبدیل می کند:

```
struct نام کلاس {
    فیلد های داده ای
};
```

بنابر این، کلاس stack به صورت زیر به ساختمان تبدیل می گردد:

```
struct stack {
    int *x;
    int myTop;
    int max;
};
```

دراین صورت، وقتی s1 و s2 از نوع stack اعلان می شوند، درواقع به صورت دو ساختمان اعلان می شوند که هر کدام دارای فیلد هایی به نام های x ، myTop و max هستند.

توابع عضو کلاس (متدها) به تابعی در C تبدیل می شوند. به عنوان مثال، فرض کنید متد fعضو کلاس myClass باشد و به صورت زیر اعلان شده باشد:

```
type1 f(type2 p);
```

یعنی تابع f از نوع type1 است و پارامتری به نام p دارد که از نوع type2 است. در این صورت، این متد به صورت زیر در می آید:

```
type1 myClassf (struct myClass *this, type2 p);
```

یعنی نام کلاس به ابتدای نام تابع الحاق می شود و پارامتری به نام thisبه پارامترهای تابع اضافه می گردد. این پارامتر موجب می شود اشاره گر به ساختمانی که از کلاس به دست آمده است، به این تابع ارسال شود و این تابع از طریق اشاره گر، بتواند در آن تغییراتی ایجاد کند.

بار دیگر دو اعلان زیر را در برنامه در نظر بگیرید:

```
stack s1(10);
stack s2(20);
```

این اعلان ها به کد زیر تبدیل می شوند:

```
struct stack s1;
stackstack(&s1, 10);
struct stack s2;
stackstack(&s2, 20);
```

معنایش این است که با اعلان stack s1(n) تابع stack (سازنده پشته) به صورت زیر در می آید:

```
void stackstack(struct stack *this, int n)
{
  this -> myTop = -1;
  this -> max = n;
  this -> x = malloc(n *sizeof(int));
}
```

۵-۱-۲- انواع داده انتزاعی کلی

برنامه نویسی می تواند با استفاده از انواع داده اولیه ی که در زبان وجود دارد نوع پایه ای را برای دسته جدیدی از اشیای داده اعلان کند و سپس صفاتی را برای آن اشیای داده مشخص نماید. به عنوان مثال پشته ای را برای نوع داده صحیح تعریف می کنیم برای پشته ای از نوع داده حقیقی و یا غیره نیاز به تعریف پکیج دیگری می باشد. تعریف نوع داده انتزاعی کلی این امکان را فراهم می کند که صفتی از یک نوع به طور جداگانه مشخص شود به طوری که یک نوع پایه با صفاتی به عنوان پارامتر تعریف می شود و سپس چندین نوع از یک نوع پایه مشتق می گردد.

نوع عناصر کلاس انتزاعی کلی در هنگام ایجاد اشیایی از آن کلاس مشخص می شود بر این اساس نوعی که هنگام ایجاد شی تعریف می شود که مناسب توسط کامپایلر تولید می گردد امتیاز استفاده از کلاس های کلی کاهش حجم برنامه و قابلیت

انعطاف می باشد مثال زیر یک کلاس کلی با یک فیلد داده ای می باشد که نوع آن در هنگام ایجاد اشیای کلاس مشخص می شود .

```
template <class T >
class simple {
   private:
      T data;
   public:
      simple (T n);
      void show();
};
```

نوع واقعی بعداً تعیین می شود

هرنوع داده اولیه را می توان تعریف کرد

برای اعلان شیءای از این نوع، به صورت زیر عمل می شود:

```
نام کلاس <نوع > object;
```

به عنوان مثال، دستور زیر شیء s را از نوع کلاس simple اعلان می کند که نوع فیلد داده ای آن int است:

```
Simple <int> s;
```

اکنون با استفاده ار کلاس کلی، یک پشته کلی را ایجاد می کنیم:

```
template <class T>
class stack {
  private:
     T *x;
     int myTop;
     int max;
  public:
     stack();
     void push(T a);
     T top();
     void pop();
     int full();
     int empty();
};
```

توجه داشته باشید که توابع مربوط به کلاس های کلی باید از نوع کلی باشند. به این ترتیب، تابع سازنده stack و توابع push و top به صورت زیر تعریف می شوند (بقیه توابع نیز به همین صورت تعریف خواهند شد):

```
template <class T>
stack<T> :: stack
{
...
```

```
}
template <class T>
void stack<T>:: push(T a)
{
...
}
template <class T>
T stack<T> :: top()
{
...
}
```

اکنون با استفاده از اعلا ن های زیر می توان یک پشته از نوع int ، یک پشته از نوع float و یک پشته از نوع char ایجاد کرد:

```
stack <int> s1(10);
stack <float> s2(20);
stack <char> s3(30);
```

پیاده سازی: کلاس کلی به عنوان الگویی در زمان ترجمه عمل می کند. وقتی کامپایلر اعلان شیء ای از کلاس کلی را ببیند، ابتدا نوع را جایگزین می کند تا شیء کاملی به دست آید. از آن جا به بعد، مثل اشیای معمولی با آن ها برخورد می شود که شرح آن گذشت.

۵-۲- وراثت

اطلاعات موجود در یک بخش از برنامه در بخشهای دیگر مورد استفاده قرار می گیرند اغلب اطلاعات بطور ضمنی بین قطعات برنامه تبادل می شود ارسال این اطلاعات را وراثت گوییم .

وراثت یعنی اخذ خواص و ویژگی های یک قطعه از برنامه توسط قطعه دیگر بر اساس رابطه ای که ما بین این قطعات وجـود دارد شکل اولیه وراثت در قواعد حوزه مربوط به داده های ساخت یافته بلوکی وجود داشت اسامی مورد استفاده در بلوک داخلـی ممکن است از بلوک خارجی به ارث برده شوند به عنوان مثال i داخل بلوک دوم از بلوک اول با ارث برده شده است.

```
{ int i, j;
    { int j, k;
      k = i + j; }
}
```

مشکل استفاده مجدد از نوع داده انتزاعی این است که، خواص و قابلیت های نوع موجود، کاملاً برای استفاده مجـدد مناسـب نیستند، بلکه معمولاً باید تغییراتی در آن ها ایجاد شود. این کار مستلزم وجود فردی است که با کد (نوع) موجود آشنایی داشـته باشد. گاهی این تغییرات مستلزم اعمال تغییرات در برنامه های کاربردی است.

مشکل دیگر برنامه نویسی با انواع داده انتزاعی این است که مستقل از یکد یگر تعریف می شوند و در یک سطح قرار دارنـد (سلسله مراتبی نیستند) . این کار، سازمان دهی برنامه را جهت تطبیق با مسئله مورد نظر دشوار می سازد. در بسیاری از مـوارد، مسئله ها دارای اشیایی هستند که با هم ارتباط دارند (خیلی شبیه به هم هستند)، به عنوان مثال، ممکن است مـسئله بـه طـور سلسله مراتبی با هم ارتباط داشته باشد.

مفهوم ***وراثت*** هر دو مشکل مربوط به استفاده مجدد از نوع داده انتزاعی را رفع می کند. وراثت موجب می شود تا یک نوع داده انتزاعی، داده ها و عملکرد نوع داده انتزاعی موجود را به ارث ببرد، به طوری که نیاز به انجام تغییرات درنوع موجود نباشد. با استفاده از مفهوم وراثت، می توان یک نوع داده انتزاعی موجود را برای رفع نیازهای مسئله جدید به کاربرد. علاوه بر این، وراثت چارچوبی را برای تعریف سلسله مراتب انواع داده انتزاعی فراهم می آورد.

وراثت به طور کلی بر دو نوع است:

وراثت ***یگانه*** در این رابطه وراثت، کلاسی مثل x فقط وارث یک کلاس مثل y است.

وراثت ***چند گانه*** در این رابطه وراثت، کلاسی مثل x وارث چند کلاس است.

اگر داشته باشیم A=>B (B از A ارث بری کرده است) A کلاس پدر یا ابر کلاس است و B کلاس وابسته ، فرزند یا زیر کلاس است اگر کلاس فقط یک پدر داشته باشد می گوییم وراثت یگانه و اگر کلاس بتواند چند پدر داشته باشد می گوییم وراثت چندگانه .

وراثت چندگانه در c++ امکان دارد و در java امکان ندارد (وراثت چندگانه)

۵-۲-۱- وراثت در C++

برای این که کلاسی از کلاس دیگر مشتق شود، به صورت زیر عمل می شود:

```
class کلاس مشتق :کلاس پایه نوع دستیابی
{
...
}
```

نوع دستیابی در این اعلان، یکی از واژه های public یا private است.

```
class base {
   int x;
   int y;
    public:
       void get();
       void diaplay();
    protected:
       calculate();
};
class derived : public base
{
   int m, n;
   public:
      void color();
};
```

توجه داشته باشید که در C++ هم وراثت یگانه و هم وراثت چند گانه وجود دارد. در دستورات زیر، کلاس c3 از کلاس هـای c1 و c2 مشتق می شود:

```
class c1 {
...
}
class c2 {
...
}
class c3 : public c1, public c2
{
...
}
```

۵-۲-۲- کلاس های مشتق

هر انتزاع شامل داده ها و توابعی است که بر روی اشیایی از آن نوع عمل می کند که این توابع را متد می نامند. داخل تعریف کلاس اسامی که در بخش public قرار دارند در خارج از کلاس قابل استفاده اند و می توانند توسط کلاسهای دیگر به ارث بـرده شوند تابع هم اسم با کلاس سازنده نام دارد و هنگام ایجاد شی فراخوانی می شود . تابع هم نام با کلاس که با ~ شروع می شـود مخرب کلاس نام دارد این تابع هنگام ار بین رفتن شیئی از آن کلاس فراخوانی می شود .

وراثت در c++ و java از طریق کلاسهای مشتق انجام می شود به عنوان مثال :

```
class base {
   int x;
   int y;
    public:
       void get();
       void diaplay();
    protected:
       calculate();
```

```
};
class derived : public base
{
   int m, n;
   public:
      void color();
};
```

کلمات کلیدی private و public قابلیت مشاهده اشیای ارثی را کنترل می کند کلمه کلیدی protected نیز استفاده می شود که اسامی موجود در آن فقط توسط کلاسهای مشتق قابل مشاهده اند . اما در خارج از تعریف کلاس قابل مشاهده نیستند.

پیاده سازی: در کلاس مشتق فقط اسامی ارثی از کلاس پایه به فضای نام محلی کلاس مشتق اضافه می شوند و اسامی عمومی برای کاربران آن کلاس قابل مشاهده اند .

اگر شی از کلاس پایه مشتق شود این وراثت توسط مترجم اداره می شود و حافظه واقعی آن شی حاوی تمام جزئیات پیاده سازی آن می باشد این کار را روش کپی در وراثت می گوییم و ساده ترین مدل پیاده سازی است . مدل دیگر پیاده سازی روش اشتراک حافظه نام دارد در این روش هر شی از کلاس مشتق از حافظه کلاس پایه استفاده می کند . خواص ارثی در کلاس مشتق تکرار نمی شوند . تغییرات در کلاس پایه منجر به تغییر در شی مشتق می شود .

۵-۲-۳- پیاده سازی وراثت در C++

وراثت در C++ قبلا توضیح داده شده است به عنوان مثال وراثت زیر را در نظر بگیرید

```
class base
{
   int width;
   int length;
   public:
      void calculate();
   protected:
      int area;
};
class derived : public base
{
  int diameter;
  float perime;
public:
  void display();
};
```

همچنان که قبلا توضیح داده شده است هنگام ترجمه به صورت struct زیر تبدیل می شود

```
struct base
{
   int width;
   int high;
   int area;
};
```

اما کلاس derived یک کلاس مشتقی می باشد از کلاس base هنگام ترجمه کامپایلر علاوه بر عناصر struct کلاس پایه عناصر مربوط به derived را به struct مربوط به آن اضافه خواهد کرد با این وجود کلاس derived به یک ساختمان داده به صورت زیر تبدیل می شود

```
struct derived {
   int width;
   int high;
   int area;
   int diameter;
   float perime;
};
```

۵-۲-۴- وراثت چند گانه

در C++ هم وراثت یگانه هم وراثت چند گانه وجود دارد به عنوام مثال در کد زیر کلاس c3 از کلاس c1 و c2 مشتق می شود .

```
class c1 {
…
}
class c2 {
…
}
class c3 : public c1, public c2
{
…
}
```

در مثال قبل تا زمانی که مجموعه ای از اشیای تعریف شده توسط کلاسهای c1 و c2 هم پوشانی نکنند (نام توابع یکی نباشد) ادغام آنها برای ایجاد کلاس c3 مشکلی به وجود نمی آورد .

۵-۳- کلاسهای انتزاعی

گاهی تعریف کلاس می تواند به صورت یک قالب باشد به طوری که کلاسهای دیگر از آن ساخته شوند دو روش برای این کار وجود دارد

۱. ابر کلاس انتزاعی
۲. وراثت mixin

۵-۳-۱-۱- ابر کلاس انتزاعی

در ابر کلاس انتزاعی از توابع virtual استفاده می شود کلاسهایی که توابع مجازی تهی داشته باشند برای ایجاد شی استفاده نمی شوند کلاسهایی که از کلاس انتزاعی ارث بری می کنند باید تعاریف مجزایی را برای خود از توابع مجازی داشته باشند . به عنوان مثال قطعه که زیر :

```
class shape
{
  public:
     virtual void draw()
    {
      …
    }
}
class circle : public shape
{
  public:
     void draw()
     {
      …
     }
}
class square : public rectangle
{
  public:
     void draw()
     {
      …
     }
}
```

کلاس circle از کلاس shop ارث بری کرده و کلاس square نیز همین طور

۵-۳-۱-۲- تابع مجازی vratual

گاهی اوقات کلاسها می توانند به صورت یک قالب باشند بطوری که کلاسهای دیگر از آن ساخته شوند در اینگونه کلاسها معمولا از توابع مجازی (virtual) استفاده می شود اگر داخل کلاس تابع virtual وجود داشته باشد باید تمام کلاسهایی که کلاس مورد نظر را به ارث می برند تابع virtual را دوباره برای خود تعریف کنند.

به عنوان مثال در شکل زیر کلاس shape یک تابع مجازی با نام drow وارد کلاس circle , square از کلاس shope ارث بری کرده اند. پس حتما باید تابع draw را برای خود به صورت مجزا تعریف کنند . کلاسی که تابع virtual تهی داشته باشد فقط برای تعریف استفاده می شود و از آن هیچ شی ایجاد نمی شود .

۵-۳-۲- اشیاء و پیام ها

اسمالتاک روشی دیگر برای توسعه اشیا و متدها ارائه می کند که به آنچه مورد ADA و c++ گفته شد متفاوت می باشد برنامه اسمالتاک مرکب از مجموعه ای از تعاریف کلاس است که حاوی اشیاء داده و متدها است . تمامی داده ها بسته بندی شده اند. زیرا تمام مدهای آن کلاس می توانند به آنها دسترسی داشته باشند پنهان سازی اطلاعات و بسته بندی ویژگی این زبان می باشد در حالی که در زبان c++ با تعریف نوع اعمال می شوند هر برنامه اسمالتاک متشکل از سه ویژگی زبان است .

- **تعریف کلاس**: دستورات اجرایی می باشند که ساختار داخلی . متدهایی را تعریف می کنند که می تواند با ایجاد و دستکاری اشیای کلاس مورد استفاده قرار گیرند.
- **نمونه سازی اشیاء**: با فراخوانی متدهای ایجاد کننده در تعریف کلاس اشیای خاصی برای هر تعریف کلاس ایجاد می شود . می توان متدهایی را برای نمونه کلاس تعریف کرد.
- **ارسال پیام**: متدها به عنوان پیامهایی به اشیاء ارسال می شوند تا عملی را انجام دهند. بجای اختصاص مجموعه ای از پارامترها به تابع که در اغلب زبانها اینگونه می باشد در اسمالتاک هر تابع به یک شی داده وابسته می شود وابستگی یک متد با یک شی پیام نام دارد.

در اسمالتاک سه نوع پیام وجود دارد.

- **پیام یگانی** : متدی است که فاقد پارامتر می باشد.
- **پیام دورویی** : متدی می باشد که برای عملکردهای محاسباتی بکار می رود. ۶+۳ موجب می شود تا متد دورویی به اضافه با آرگومان ۶ به شی ۳ فرستاده شود. متد به اضافه شی ۹ را برای این پیام بر می گرداند.
- **پیام کلمه کلیدی** : رفتاری مشابه با توابع همنام در زبانهایی مثل ADA و c++ دارد.

۵-۳-۲-۱- وراثت کلاس

داده های اسمالتاک بر اساس سلسله مراتب کلاس مشخص می شوند اگر هر متدی که به شی ارسال می شود در آن کلاس تعریف نشده باشد به کلاس پدر ارسال می شود و این روند ادامه می یابد کلاس abject ابر کلاس ، یا پدر و همه کلاسها است وراثت متدها ویژگی ذاتی زبان می باشد.

۵-۳-۳- مفاهیم انتزاع

وراثت مکانیزمی برای انتقال اطلاعات بین اشیای کلاسهای مربوط ارائه می کند اگر A→B به معنای این باشد که B به A ربط دارد رابطه مابین اشیای A و اشیای B چیست ؟

۴ نوع رابطه وجود دارد:

۱- **اختصاصی شدن**[1] : متداول ترین شکل وراثت است که اجازه می دهد مشتق B خواص دقیقی نسبت به آنچه در A وجود دارد کسب کند.

۲- **تجزیه**[2] : اصل تفکیک انتزاع به عناصر آن است نوعی مکانیزم بسته بندی در زبانهایی مثل ADA بدون وراثت متدها است .

[1] specialization
[2] decomposition

۳- **نمونه سازی**[1] : فرایند ایجاد نمونه ای از کلاس است . در واقع عملیات کپی است . مفهوم متضاد آن دسته بندی است .

۴- **انفرادی سازی**[2] : در این حالت اشیای مشابه برای اهداف مشترکی در یک گروه قرار می گیرند به عنوان مثال ممکن است بین پشته ها در یک مترجم رکورد فعالیت یا جدول نما وجود داشته باشد c و پاسکال رکوردهای فعالیت را به صورت پشته پیاده سازی می کنند اما پشته هایی با ویژگی های متفاوت . تمام اینها نمونه هایی از اختصاصی شدن است . تمام آنها push و pop را انجام می دهند.

۵-۴- چند ریختی

استفاده از پارامتر ها در زیر برنامه ها قدیمی ترین ویژگی زبان های برنامه سازی است چند ریختی به این معناست که نام یک زیر برنامه با عملگر بر حسب انواع آرگومان ها و نتایج به هر تعداد از تعریف توابع اشاره می کند چند ریختی به توابعی اعمال می شود که یک نوع به عنوان آرگومان آنها است زبان هایی مانند ام ال و اسمالتاک از چند ریختی به بهترین شکل استفاده می کنند .

پیاده سازی : برای زبان هایی مانند ام ال و C++ چند ریختی پیچیدگی جدیدی را ایجاد نمی کند اما زبان هایی که چند ریختی پویا را اجاره می کنند منجر به مشکل می شود زیرا آرگومان تابع چند ریختی باید در حین اجرای برنامه تعیین شود آرگومان به دو شکل می تواند به تابع چند ریختی ارسال شود

۱- **توصیفگر صریح** : حالتی است که مقدار ارسالی به تابع کوچکتر از اندازه ثابت فیلد باشد

۲- **توصیفگر فشرده** : حالت توصیفگر فشرده حالتی غیر حالت توصیفگر صریح است

[1] instalation
[2] individualization