درس طراحي و پياده سازي

زبان هاي برنامه نويسي

استاد: سرکار خانم میرشکاری

به نام خدا

جلسه اول.

چرا زبان های برنامه سازی را مطالعه می کنیم ؟

۱- برای گسترش و بهبود توانائی خود در توسعه الگوریتم های کار آمد مثل فهمیدن تکنیک باز گشتی.
۲- استفاده بهینه از زبان برنامه نویسی موجود، با درک این که چگونه ویژگی های یک زبان پیاده سازی می شود توانائی شما در نوشتن برنامه های کار آمد افزایش می یابد.
۳- آشنائی بیشتر با انواع ساختار های مفید برنامه نویسی، این که چه زبانی از چه ابزار ها یا ساختارهای کنترلی پشتیبانی می کند، مثل پشتیبانی از شئی گرائی یا هم روال ها.
۴- انتخاب بهترین زبان برنامه نویسی، برای مثال برای کاربردهای هوش مصنوعی از پرولوگ، محاسباتی فرترن، سیستمی C و C++ ، پردازش رشته(Lisp).
۵- یاد گیری آسان یک زبان جدید
۶- طراحی یک زبان جدید

تاریخچه پیدایش و توسعه زبان های برنامه نویسی

نسل اول(زبان های ماشین یا زبان های صفر و یک):

از اواخر دهه ۱۹۳۰ تا اوایل ۱۹۴۰ بیشتر به عنوان کامپیوتر های محاسبه گر از آنها استفاده می گردد. عمده کاربرد این ها محاسبات عددی بود . از خصوصیات اصلی آنها وابستگی به ماشین بود.

نسل دوم، زبان های نمادی(اسمبلی):

به جای رشته های باینری از یک سری نماد های نیمانیک(nemanic) جهت نمایش دستورات و محل های حافظه استفاده می شود.
علاوه بر وابسته بودن به ماشین، نیاز به یک مترجم ضروری بود.

نسل سوم(زبان های سطح بالا یا زبان های ساخت یافته):

این زبان ها نیاز به یک مترجم جهت ترجمه آن به زبان اسمبلی یا ماشین داشتند و به زبان طبیعی نزدیک بودند، تحول اصلی در جهت ایجاد این زبان ها در سال های ۱۹۵۵ الی ۱۹۵۷ با تولید زبان فرترن که برای محاسبات عددی استفاده می شد صورت گرفت، ساختارهائی مثل حلقه ها، دستورات شرطی و پروسجرها یا رویه ها به وجودآمدند. زبان هائی مثل پاسکال، الگول(ALGOL)،کوبول، c و غیره از این گونه اند.

نسل چهارم(زبان های خیلی سطح بالا):

زبان های تحت ویندوز مثل VB ، Visual C از جمله این زبان ها هستند.

نسل پنجم:

زبان های مدل سازی و تولید کد مثل UML

عوامل موثر در پیدایش و طراحی زبان های برنامه سازی

۱- سخت افزار و سیستم عامل(plat form)

۲- روش های برنامه سازی(شئی گرائی، تابعی، دستوری و ...)

۳- کاربرد ها(صنعتی، تجاری، نظامی و ...)

۴- روش های پیاده سازی(مثلا برای پیاده سازی آرایه از چه تکنیکی استفاده شود،مثلا از Link list ویا فضای ثابت و پشت سر هم حافظه)

۵- مطالعات تئوریک

۶- استاندارد سازی

ویژگی های یک زبان خوب

۱- **وضوح، سادگی و واحدمند بودن مفاهیم:**

☐ هر چه مولفه های یک زبان کمتر باشد سادگی بیشتری خواهیم داشت.

☐ هر چه درک مفاهیم زبان آسان باشد وضوح بیشتری خواهیم داشت.

☐ هر جا ارتباط بین اجزای زبان مشخص تر باشد و میزان انتزاع(Abstration) قدی تر باشد، زبان برنامه سازی واحد مندتر است.

۲- **وضوح در نحو یا Syntax برنامه**: هر چه وضوح بیشتری درSyntax زبان و یا متن برنامه باشد، زبان برنامه سازی بهنر است.

۳- **طبیعی بودن زبان با کاربرد یا تطبیق بیشتر زبان با کاربرد**: مثلا برای رشته ها از زبان Lisp استفاده کنیم

۴- **پشتیبانی تجرید(Support for abstraction)**:هر چه طبقه بندی اجزاء قوی تر انجام شود، زبان برنامه سازی بهتر است

۵- **سادگی تغییر دادن برنامه**

۶- **محیط برنامه سازی:** هر چه امکانات محیط برنامه سازی نظیر کامپایلر،Linker،Debugger بیستر باشد آن زبان برنامه سازی بهتر است.

۷- **انتقال پذیری برنامه(Portability):** هر چه انتقال از سخت افزار(سیستم عامل) به سخت افزار(سیستم عامل) دیگری راحت تر انجام شود زبان برنامه سازی بهتر است.

۸- **پایین بودن هزینه استفاده:**

☐ هزینه اجرای برنامه شامل استفاده از منابع کامپیوترمخصوصا CPU، RAM، Disk

☐ هزینه ترجمه برنامه شامل استفاده از منابع کامپیوترمخصوصا CPU، RAM، Disk

☐ هزینه ایجاد و تست برنامه شامل استفاده از منابع کامپیوتر و نیروی انسانی.

□ هزینه پشتیبانی شامل هزینه تطبیق برنامه با جوانب محیطی، ایجاد امکانات جدید در نرم افزار و بر طرف کردن اشکالات.

۹- **قابلیت تعامد(Orthogonality):** منظور از تعامد این است که بتوان ویژگی های مختلف از یک زبان را با هم ترکیب کرد و ترکیب حاصل نیز با معنا باشد، مثلا عبارت محاسباتی و دستور شرطی زیر را در نظر بگیرید.

… then (a+b>c+d) If □تعامد، یعنی یک دستور محاسباتی را در دستور If بکار بردیم.

انواع زبان ها از نظر کاربرد

۱- زبانها ی پردازش تجاری مثل کوبول در گذشته و C و C++اخیرا

۲- کاربرد های علمی مثل فرترن

۳- کاربرد های سیستمی، برای نوشتن سیستم عامل و پیاده سازی کامپایلرها و ... می باشند. این دسته از زبان ها باید قادر به دستیابی به سخت افزار و ایجاد ارتباط به آن باشند مثل اسمبلی، PL/1

۴- کاربرد های هوش مصنوعی مثل پرولوگ

۵- زبان های فرایندی یا اسکریپتی(کامپایل نمی شوند بلکه تفسیر می شوند و بیشتر تحت وب اجرا می شوند)،خط به خط اجرا می شود، بر عکس برنامه ای که برنامه دیگر را کنترل می کند، مثل برنامه تهیه پشتیبان از فایل ها یا قاصد خودکار به پست الترونیکی، تنظیم ساعت و ... به این نوع زبان ها زبان های فرایندی گویند که فعالیت های(فرایند های) فوق الذکر را شناخته، ترجمه و اجرا می کنند، مثل زباان Perl

۶- کاربرد های جدید، مثل ML در تحقیقات زبان های برنامه سازی برای بررسی تئوری نوع و اسمااتاک(شئی گرای محض)

تکامل بستر توسعه نرم افزار

توسعه نرم افزار در خلاء انجام نمی شود، سخت افزاری که زبان را پشتیبانی می کند تاثیر زیادی در طراحی زبان دارد، محیط خارجی که اجرای برنامه را پشتیبانی می کند(مثل سیستم عامل و سخت افزار)، محیط عملیاتی یا محیط مقصد نامیده می شود.محیطی که برنامه در آن طراحی، کد، تست و اشکال زدائی می شود محیط میزبان نام دارد.محیط مقصد ممکن است با محیط میزبان متفاوت باشد، در زیر به چند نمونه از این محیط ها که توسعه و اجرای نرم افزار در آنها صورت می گیرد اشاره ای می کنیم.

محیط دسته ای:

ساده ترین محیط عملیاتی متشکل از فایل داده هاست. برنامه مجموعه ای از فایل های داده را به عنوان ورودی گرفته، داده های آنها را پردازش و مجموعه ای از فابل های داده خروجی را تولید می کند. این محیط عملیاتی را پردازش دسته ای(bach processing) می گویند. زیرا اطلاعات ورودی در فایل ها دسته بندی می شوند و به صورت دسته ای پردازش می شوند، در این محیط خطائی که اجرای برنامه را خاتمه دهد قابل قبول نبوده، زیرا پس از پردازش خطا، برنامه باید به طور کامل اجرا شود. هیچ کمک فوری برای کاربر وجود ندارد و زبان به کار رفته در این محیط هیچ تاثیری را بر روی سرعت اجرای برنامه ارئه نمی کند.

محیط محاوره ای:

در این محیط یک برنامه مستقیما با کاربر تعامل دارد و خروجی در نمایشگر نشان داده میشود. این گونه محیط ها از سیستم اشتراک زمانی برای انجام کارهای مختلف که در یک زمان واحد به کامپیوتر داده می شود استفاده می کنند، با این ترتیب که به هر برنامه یک برش زمانی(Time Slice) داده می شود، بعد از اتمام این برش زمان، پردازنده به برنامه دیگری که آماده اجراست داده می شود. پردازش خطا در محیط محاوره ای از اهمیت کمتری بر خوردار است و زبان به کار رفته در این محیط نیاز مبرمی به داشتن پردازش خطای قوی ندارد، زیرا کاربر به صورت online پشت کامپیوتر بوده و در صورت بروز خطا می تواند برنامه را دستکاری و تصحیح کند اما خاتمه برنامه در صورت بروز خطا قابل قبول نیست.

محیط سیستم های تعبیه شده(embeded system):

در این محیط ها سیستم کامپیوتری بخشی از یک سیستم بزرگ مثل هواپیما یا ماشین تراش است که این سیستم کامپیوتری وظیفه کنترل کل یا بخشی از سیستم بزرگ را به عهده دارد. در این چنین سیستم هائی خرابی قابل قبول نیست و لطمات جبران ناپذیری به سیستم بزرگ وارد می گردد از مشخصه های اصلی این نوع محیط ها می توان به موارد زیر اشاره کرد.

☐ قابلیت اعتماد و صحت(Reliability):

☐ برنامه های نوشته شده برای این گونه محیط ها معمولا بدون سیستم عامل مربوطه و بدون محیط معمولی فایل و دستگاه های ورودی و خروجی فایل اجرا می شود. در عوض هر کدام از برنامه ها رویه های مخصوص برای ارتباط با دستگاه های ورودی و خروجی سیستم بزرگ دارد و به تعامل به آنها می پردازد.

☐ پردازش خطا در این محیط ها بسیار مهم و حیاتی است هر برنامه باید بتواند خطا ها را در داخل خودش بر طرف سازد. خاتمه برنامه مگر در مواقع فاجعه، غیر قابل قبول است.

☐ این نوع سیستم ها در زمان بلا درنگ کار می کنند یعنی پاسخ سیستم کامپیوتری برای ورودی خاصی که از سیستم بزرگ گرفته است باید در یک بازه زمانی مشخص تضمین شده باشد.

☐ معمولا این نوع سیستم ها یک کامپیوتر واحد نیستند و از چندین قسمت تشکیل شده است، هر قسمت وظیفه خاص خود را دارد و هر کدام بخشی از سیستم بزرگ را کنترل می کند و به طور هم زمان عمل می کند.

محیط های توزیعی(**Disturbuted**):

در سال های اخیر گرایش به سمت برنامه هائی است که قابلیت اجرا به صورت Remote (از راه دور) داشته باشند. محیط های وب و شبکه های محلی از جمله این محیط ها هستند که برنامه ها در آنها به صورت Remote اجرا میگردد.

مدل های زبان

☐ **مدل دستوری(imperative):**

در این زبان ها برنامه شامل دنباله ی از دستورات است و اجرای هر دستور موجب می شود تا مترجم مقدار ثبات های CPU یا محلی از حافظه را تغییر دهد. یعنی ماشین را به حالت جدیدی ببرد. نحو چنین زبان هائی به صورت زیر است.

دستور۱
دستور۲
⋮

این دید گاه در زبان هائی مثل C، C++ و Ada پشتیبانی می شود.

☐ **زبان های تابعی(functional):**

در این روش به جای دنبال کردن تغییر حالت ماشین عملکرد برنامه دنبال می شود. یعنی به جای این که داده های موجود را در نظر بگیریم، نتیجه مطلوب را در نظر خواهیم داشت، یعنی در صورت بر قراری عملیات بر روی داده ورودی نتیجه مطلوب حاصل می شود. توسعه برنامه با ایجاد توابعی از توابع ایجاد شده قبلی به منظور ساختن توابع پیچیده انجام می شود تا این داده اولیه را دستکاری کرده و آخرین تابع پاسخ را از داده های اولیه تولید نماید. نحو این زبان ها به صورت زیر است.

Lisp و ML دو زبان تابعی هستند $f_n(...(f_2(f_1(data)))...)$

☐ **زبان های قانونمند(rule based):**

در این زبان ها شرایطی بررسی می شود و در صورت بر قرار بودن آن شرایط، فعالیتی صورت می گیرد. متداول ترین زبان قانون مند prolog می باشد که زبان برنامه نویسی منطقی نیز نامیده می شود. نحو چنین زبان هائی به صورت زیر است.

انجام عملیات ۱ ⇐ تعدادی شرط ۱
انجام عملیات ۲ ⇐ تعدادی شرط ۲
⋮
انجام عملیات n ⇐ تعدادی شرط n

☐ **مدل های شئی گراء(object oriented):**

اساس این زبان ها کلاس ها و اشیای ساخته شده از روی این کلاس ها می باشد. هر کلاس نشان دهنده یک رده خاص از اشیاء است که خصوصیان و رفتار مشابه هم دارند نمونه ای این مدل عبارتند از java ، .C++

ساختار عملیات یک کامپیوتر

الف-Data :

یک کامپیوتر باید مجموعه ای از داده های اولیه و ساختار یافته را برای انجام عملیات فراهم کند.

ب-primitive operation :

یک کامپیوتر باید مجمو عمه ای از عملیات اولیه برای عملیات روی دا ده ها را در بر داشته باشد(مانند دستورات CPU یا زبان ماشین)

ج-Data Control :

یک کامپیوتر باید مکانیزم هائی جهت کنترل اجرای دستورات فراهم سازد

د-Storage management :

یک کامپیو تر باید مکانیز م هائی جهت تخصیص حافظه برای برنامه و داده و همچنین آزاد سازی حافظه داشته باشد.

و-operating environment :

یک کامپیوتر باید مکانیز هائی برای مبادله اطلاعات با دستگاه های جانبی فراهم سازد

انواع کامپیو تر ها

☐ Harware

کامپیوتر سخت افزاری کامپیوتری است که کاملا از اجزاء سخت افزاری و مدارات الکترنیکی ساخته شده است. دقیقا سخت افزار مربوط به هر دستور زبان ماشین وجود دارد.

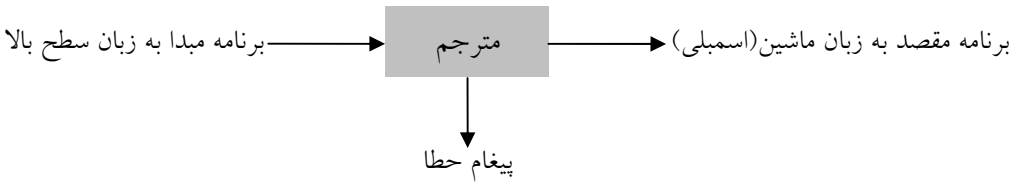
☐ Firmware

یک کامپیوتر به صورت Firmware ساخته می شود که هر دستور زبان ماشین دنباله ای از Micro operation ها است که در حافظه قابل بر نامه ریزی(ROM یا PROM) ذخیره شده است.

برای پیاده سازی و اجرای یک زبان سطح بالا می توا ن از طریق یکی از دو راه زیر عمل کرد

☐ **ترجمه(Translation)** :

در این روش برنامه ای به زبان سطح بالا طی فرایند هائی تبدیل به زبان ماشین می شود که قابل اجرا روی سخت افزار است



انواع مترجم ها به صورت زیر است

۱- **کامپایلر**: ابزاری است برای انتقال برنامه به شکلی که بتواند بر روی سخت افزار اجرا شود.

۲- **اسمبلر**: انتقال برنامه از زبان اسمبلی به زبان ماشین را انجام می دهد.

۳- **لودر(بار کننده)**: برنامه قابل اجرا را در حافظه RAM قرار میدهد.

۴- **Linker** :قسمت های مختلف بر نامه را به یکد یگر متصل کرده و بر نامه را قابل اجرا می سازد.

۵- **پیش پردازنده ها** : منظور از Preprocessore آن است که ورودی برنامه زبان سطح بالا و خروجی به صورت استاندارد هائی به همان زبان است. مثال خاص آن فایل های Include در زبان C است

☐ **شبیه سازی نرم افزاری(software simulation)** :

در این مکانیزم Source برنامه مستقیما به simulator داده می شود(simulator یک نرم افزار است). و simulator دستور زبان سطح بالا را تفسیر و مجموعه دستورات لازم برای انجام دستور زبان سطح بالا را اجرا می کند.

□ تفاوت این دو روش در آن است که در روش اول برنامه کامل به زبان ماشین تبدیل شده و سپس اجرا می شود در حالی که در روش دوم تک تک دستورات زبان سطح بالا ابتدا تفسیر و سپس مجموعه دستورات لازم برای شبیه سازی آن دستور اجرا می گردد.

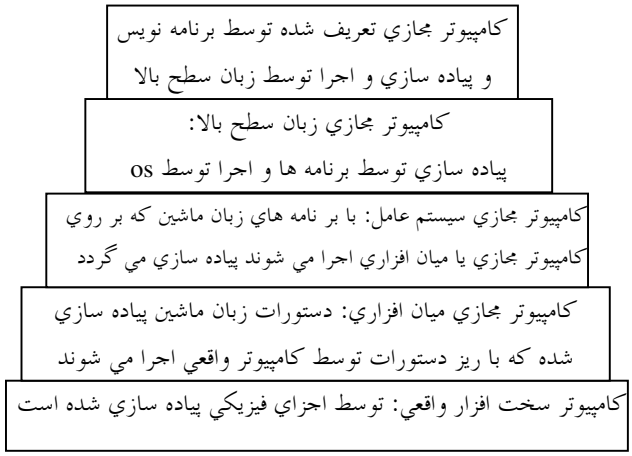
جلسه دوم

سلسله مراتب کامپیوتر ها

نکته:

در عمل یک کامپیوتر مجازی داریم، یعنی کامپیوتری که با زبانی غیر از زبان ماشین کار می کند.

از دید برنامه نویس به زبان سطح بالا و در عمل ساختار یک virtual computer را می توان به صورت سلسله مراتب شکل زیر در نظر گرفت.کامپیوتری مجازی است که توسط طراح زبان برنامه سازی طراحی می شود.



☐ نتیجه ای که از بحث سلسله مراتبی بودن کامپیوتر های مجازی بدست می آید این است که داده و برنامه معادل هم هستند، به عبارتی دیگر یک برنامه در یک محیط به عنوان داده و در محیط دیگر به صورت برنامه عمل می کند. به عنوان مثال فایل HTML توصیف کننده یک صفحه وب برای ماشین مجازی وب به عنوان داده محسوب می شود اما از نظر برنامه نویس وب که از HTML برای پردازش اطلاعات کاربران آن وب سایت استفاده می کند یک برنامه است.

☐ C و فرترن داده و کد برنامه را جدای از هم ذخیره می کنند اما زبان هائی مثل Prolog و Lisp هر دو را یک جا ذخیره می کنند.

تعریف Binding (مقید سازی) و زمان Binding (زمان مقید سازی)

به هنگام پیاده سازی و اجرای یک برنامه به زبان سطح بالا یک عنصر از برنامه می تواند یک خصوصیت از مجموعه خواص ممکن را به خود بگیرد این عمل را Binding و زمان آن را Binding time گویند. مثلا یک متغیر صحیح در حین اجرا می تواند یک ارزش از مجموعه ارزش های ممکن را به خود بگیرد.

انواع مختلف Binding :

انواع Binding برحسب زمان انجام این عمل طبقه بندی شده وشامل موارد زیر است.

☐در زمان اجرا(Run time)

- در زمان اجرا برنامه وارد بلاک یا زیر برنامه خاصی می شود.
- در هر نقطه از برنامه یک نوع Binding انجام می شود و در واقع نتیجه اجرای برنامه اثرات Binding ها است.

☐در زمان ترجمه(Translation or compile time)

- Binding انتخاب شده و یا اعمال شده توسط برنامه نویس، به عنوان مثال تعریف متغیر ها، انواع آنها و ساختار برنامه توسط برنامه نویس مشخص می شود.
- Binding انتخاب شده و یا اعمال شده توسط Translator ، به عنوان مثال در زمان load برنامه، متغیر ها به محل های خاصی از حافظه Bind می شوند.

☐در زمان پیاده سازی زبان(Language implementation time)

به عنوان مثال نمایش اعداد، عملیات، محاسبات ریاضی که از بین کلیه حالت ممکن یکی انتخاب می شود.

☐در زمان تعریف زبان(language definition time)

اکثر ساختار های زبان برنامه سازی در هنگام تعریف زبان مخص می شود.مثل تعین کلمات کلیدی و یا به عنوان مثال دیگر حرف I,J در fortran از نوع int هستند.

زمان Binding می تواند روی انعطاف پذیری و سرعت برنامه موثر باشد.

☐ اگر این عمل در حین اجرا انجام شود انعطاف پذیری زیان ولی سرعت کم به همراه خواهد داشت که به آن انقیاد دیرس گویند.

☐ اگر این عمل در غیر از زمان اجرا باشد سرعت اجرای برنامه زیاد ولی انعطاف پذیری کم را در بر خواهد داشت(انقیاد زودرس).

یک دلیل برای توجیه این واقعیت این است که اگر Type یک برنامه در حین اجرا بتواند تغییر کند انعطاف پذیری زیاد بوجود می‌آید ولی به دلیل استفاده از روتین های تبدیل نوع اطلاعات(Type convesion) کند است.

برای تشریح انقیاد و زمان‌های انقیاد گوناگون دستور انتساب x=x+10 را در نظر می‌گیریم.

با فرض این که این دستور در زبان L نوشته شده است در این صورت در مورد انقیادها و زمان های انقیاد موارد زیر دیده می شود.

☐ **مجموعه‌ای از انواع ممکن برای** x : مجموعه‌ای از انواع قابل قبول برای x در زمان تعریف زبان مشخص می شود یعنی مثلا در پاسکال فقط می توان از انواع real ، integer، character، set ،Bool و ... استفاده کرد.

☐ برخی زبان ها به کاربر اجازه می دهند تا انواع جدیدی را تعریف کند مثل C و پاسکال(مثل نوع شمارشی در پاسکال) در این صورت مجموعه ای از انواع ممکن در زمان ترجمه مشخص خواهد شد.

☐ **نوع متغیر** x : معمولا در زمان ترجمه مشخص می شود. مثلا در پاسکال برای این منظور یک متغیر باید اعلان شود. در برخی زبان ها مثل اسمالتک و پرولوگ نوع داده x ممکن است در زمان اجرا مشخص شود به طوری که انتساب مقداری از یک نوع به x ، نوع x را مشخص می کند. در این نوع زبان ها ممکن است x در قسمتی از برنامه از نوع صحیح و در جای دیگر مقدار کاراکتری داشته باشد.

☐ **مجموعه‌ای از مقادیر ممکن برای** x : در زمان پیاده سازی زبان تعداد بیت هائی، برای قرار دادن یک مقدار از نوع Float تعیین می گردد. لذا مجموعه دقیقی از از مقادیر ممکن برایx به وسیله این تعداد بیت ها تعیین می گردد. مثلا اگر شانزده بیت برای یک متغیر از نوع صحیح در نظر گرفته شود(این عمل در زمان پیاده سازی است) مجموعه مقادیر مکن برایx عبارت است از:

☐ **مقدرا متغیر x** : در هر نقطه از اجرای برنامه مقدار خاصی به x ، Bind می شود.

☐ **مقدار ثابت 10** : انتخاب نمایش دهدهی در متن داخل برنامه یعنی(۱۰ برای عدد **ده**) در زمان تعریف زبان انجام می شود. در حالی که انتخاب رشته ای از بیت ها برای نمایش داخلی،در زمان پیاده سازی انجام می شود.

☐ عملگر+ :

- انتخاب نماد + برای نمایش عمل جمع در زمان تعریف زبان انجام می شود.
- معنای عملگر + برای انجام جمع در زمان ترجمه مشخص می شود(پس از مشخص شده نوع عملوند ها مشخص می شود که علامت + چه جمعی را انجام می دهد، جمع صحیح، جمع اعشاری، جمع موهومی یا ...)

☐ overload (**تعریف مجدد**) :

اینکه امکان تعریف چند باره عملیات برای یک نماد خاص در یک زبان داده شود تا برای انواع ورودی مختلف، عملیات متناسب با آن ورودی را انجام دهد،**overload** گفته می شود.

☐ در بسیاری از زبان ها متداول است که در زمان کامپایل مشخص شود که چه عملی توسط + انجام شود. با استفاده از نوع آرگومان های بکار رفته در عمل جمع مشخص می شود که باید از عمل جمع صحیح یا اعشاری یا ... استفاده شود.

☐ جزئیات هر عمل جمع در زمان پیاده سازی انجام می شود.

☐ در کل برای زبان هائی مثل پاسکال، + به مجموعه‌ای از اعمال جمع در زمان تعریف زبان **Bind** می شودو هر عمل جمع در این مجموعه در زمان پیاده سازی زبان تعریف میشود. هر استفاده خاص از نماد + در برنامه در زمان ترجمه ، به یک عمل خاص جمع **Bind** می شود، مقدار هر عمل جمع برای عملوند های آن فقط در زمان اجرا مشخص می شود.

☐ زمان انقیاد می تواند روی انعطاف پذیری و سرعت برنامه موثر باشد، اگر عمل مقید سازی در حین اجرا مشخص شود، انقیاد دیر رس گفته میشود و اگر این عمل در غیر از زمان اجرا باشد(معمولا در زمان ترجمه) سرعت اجرای برنامه زیاد ولی انعطاف پذیری کم را در بر دارد(انقیاد دیر رس). یک دلیل برای توجیه این امر آن است که اگر Type یک برنامه بتواند تغییر کند با این که انعطاف پذیری زیاد بوجود می آید روتین کند می گردد.

نکته: در زبان های C و Fortran اغلب انقیاد ها در زمان ترجمه صورت می گیرد. و در زبان هائی مثلML و Lisp اغلب انقیاد ها در زمان اجرا صورت می گیرد.

جلسه سوم

اطلاعات از نوع اولیه(Elementary Data Type یا (E.D.T))

یکی از اختلافات اساسی زبان های برنامه سازی انواع اطلاعاتی است که یک زبان برنامه سازی می تواند روی آنها عملیات انجام دهد . طبعا چگونگی و تنوع عملیات بر مبنای قابلیت های یک زبان جهت پوشش اطلاعات می باشد. در این فصل چگونگی پیاده سازی اطلا عات از نوع اولیه(تنها شامل یک ارزش) بحث میشود.

تعاریف اولیه

شئی داده(Data Object):

یک شئی داده به گروهی از یک یا چند قسمت از اطلاعات در کامپیوتر مجازی گفته میشود. در واقع شئی داده ظرفی(Container) برای مقادیری از داده هاست، یعنی محلی است که مقادیر داده در آنجا ذخیره و باز یابی میشود. یک شئی داده توسط مجموعه ای از صفات مشخص میشود که مهمترین آنها نوع داده است.

اشیای داده ای به دو گروه زیر تقسیم میشوند:

الف- تعریف شده توسط برنامه نویس: مانند متغیر ها، مقادیر ثابت، آرایه ها و فایل ها

ب- تعریف شده توسط سیستم: مانند پشته ها در زمان اجرای برنامه

ساختار یک شئی داده

- ✓ یک شئی داده توسط مجموعه ای از صفات مشخص میشود. مثل نوع داده و نام که معمولا در طول عمر آن عوض نمی شوند.
- ✓دارای محلی برای مقدار داده(Data Value) است.
- ✓ یک شئی داده اولیه است اگر تنها شامل یک محل حافظه برای Data Value باشد.
- ✓ یک شئی داده ساخت یافته یا Structuerd است اگر شامل جمعی از اشیائ داده اولیه باشد.
- ✓می تواند Single Number ، Character ، Pointer و غیره باشد.
- ✓ توسط الگوئی از بیت ها مشخص میشود. به شکل زیر دقت کنید.

A: 0000000000010001 **ج. متغیر مقید:** شئی داده به مقدار داده 17 مقید میشود	10001 **ب. مقدار داده:** الگوی بیتی است که هر وقت عدد 17 در برنامه استفاده میشود؛ مترجم از آن استفاده می کند.	A: **الف. شئی داده:** محلی در حافظه کامپیوتر به نام A

انواع مختلف انقیاد یک شئی داده

۱- **انقیاد یک شئی داده به یک نوع:** این انقیاد در زمان ترجمه برنامه انجام می گیرد. یعنی مجموعه ای از مقادیر که شئی داده ای می تواند بپذیرد به آن نسبت داده می شود.

۲- **انقیاد یک شئی داده به محلی از حافظه:** این انقیاد که از دید برنامه نویس پنهان است توسط روال های مدیریت حافظه کامپیوتر مجازی انجام می گیرد.

۳- **انقیاد یک شئی داده به یک یا چند ارزش**: این نوع انقیاد توسط دستورات انتساب میشود.

۴- **انقیاد یک شئی داده به یک یا چند نام** : این انقیاد توسط اعلان ها(declaration) انجام پذیرفته و هنگام فراخوانی زیر برنامه و برگشت از آن اصلاح میشود.

۵- **انقیاد یک شئی داده به یک یا چند شئی داده دیگر:** به هنگام استفاده از متغیر هائی از نوع اشاره گرها استفاده میشود.

- یک متغیر یک شئی داده ای است که توسط برنامه نویس تعریف شده و به صورت صریح استفاده میشود.
- یک ثابت یک شئی داده ای است که نام آن به یک ارزش(مقدار) مقید میشود.
- یک ثابت رشته ای(حرفی)، شئی داده ای است که نام آنها همان ارزش(مقدار) آنها می باشد.
- هر شئی داده چرخه حیات یا Life Time مخصوص به خود را دارد، منظور از Life Time از لحظه به وجود آمدن تا زمان از بین رفتن است.

مثال. متغیر، ثابت و ثابت حرفی را مشخص کنید

```
Const  Max=100;
Var    N: Integer;
N:=27;
N:=N+Max;
```

جواب.

۱- N متغیر ساده است(فقط یک مقدار می تواند بگیرد)

۲- Max یک ثابت تعریف شده توسط کاربر است

۳- 100 و 27 ثابت حرفی هستند.

منظور از مفهوم داده ها ماندگارند چیست؟ یعنی طول عمر متغیر ها با زمان اجرای برنامه یکی است، اجرا که تمام شد متغیر ها هم از بین میروند. اما طول عمر یک داده بیشتر از یک اجرا است لذا گوئیم داده ها ماندگارند.

انواع داده(Data Type): نوع داده کلاسی از اشیاء داده به همراه مجمو عه ای از عملیات برای تولید و دستکاری می باشد. هر زبان مجموعه ای از انواع داده اولیه دارد که هنگام تعریف زبان مشخص شده اند.

یک نوع داده در دو سطح مختلف می تواند برسی شود.

۱- تعریف کردن(تعیین مشخصات یک نوع)(Specification)

۲- پیاده سازی(Implementation)

در سطح تعریف و تعیین مشخصات یک نوع موارد زیر در نظر گرفته میشود

☜ **صفات یک نوع داده**(Attributes) : ویژگی که این نوع را از بقیه نوع ها جدا می کند. مثلا در مورد ارایه ها شامل ابعاد آرایه، اندیس آرایه و نوع عناصر آرایه می باشد.

☜ **مقادیر**(Values) :ارزش های مختلفی که یک نوع داده ای می تواند بگیرد

☜ **عملیات**(operation) : انواع دستکاریهای ممکن برای نوع داده ای خاص

عناصر اساسی جهت پیاده سازی

☜ چگونگی ذخیره اطلاعات و نمایش حافظه ای مربوط اشیاء آن نوع داده

☜ نحوه پیاده سازی عملیات که ممکن است به یکی از سه طریق: عملیات سخت افزاری، به صورت زیر برنامه رویه یا تابع و یا مجموعه ای از دستورات داخل برنامه باشد.

عملیات

مجمو عه ای از عملیاتی که بر روی یک نوع داده تعریف شده است تعیین می کند که آن نوع اشیاء داده چگونه باید دستکاری شوند این عملیات ممکن است عملیا ت اولیه یا عملیاتی باشند که توسط برنامه نویس تعریف شده باشند.

عملیات اولیه به عنوان بخشی از زبان تعریف میشوند، عملیات تعریف شده توسط برنامه نویس به شکل زیر برنامه ها نوشته میشوند (در کلاس ها به صورت متد تعریف می گردند)، مثل زیر برنامه ای که قدر مطلق یک شئی داده از نوع صحیح را محاسبه می کند.

هر عملیات معمولا به صورت یک تابع ریاضی بیان میشود، به طوری که یک یا چند پارامتر(عملوند) را به عنوان ورودی پذیرفته و نتایجی را تولید می کند. مجموعه ای از مقادیر ورودی که عملیات بر روی آن تعریف شده است **دامنه** عملیات و مجمو عه ای از نتایج ممکن **برد** عملیات نام دارد، الگوریتم موجود در بدنه عملیات مشخص می کند بر روی پارامتر های ورودی چه محاسباتی انجام شود تا نتایج مطلوب بدست آید،یعنی الگوریتم عملکرد عملیات را مشخص می کند.

عوامل موثر در پیچیده شده عملیات بر روی اشیائ داده در زبان های برنامه سازی(خیلی مهم)

چهار عامل اساسی تعریف عملیات زبان های برنامه سازی به صورت توابع ریاضی را پیچیده می کند.

۱- **عملیاتی که به ازای ورودی مشخصی(بعضی از مقادیر دامنه) تعریف شده نیستند** مثل عملیات تقسیم بر صفر، یعنی تعیین دقیق دامنه عملیات مشکل می باشد

۲- **آرگومان های ضمنی** : ورودیهای ضمنی و یا ورودیهائی که به صورت صریح تعریف نشده اند مثل متغیر های سراسری باعث میشوند که باز هم تعین دقیق دامنه عملیات بر روی اشیاء داده ممکن نباشد.

۳- **اثرات جانبی** : یک عملیات ممکن است علاوه بر وظیفه اصلی خود اعمال دیگری را نیز انجام دهد، مثلا تابع ممکن است علاوه بر مقدار برگشتی، آرگومان های ورودی خود را نیز تغییر دهد این کار نوعی اثر جانبی است.

۴- **خود اصلاحی(حساسیت به سابقه اجرا)** : مثلا در صورتی که یک زیر برنامه اعداد تصادفی تولید کند برای دفعت بعد اجرا از قبلی اجرا می پذیرد. مثلا متغیر استاتیک در C مثالی از حساسیت به قبل است.

زیر نوع ها

وقتی نوع داده جدیدی را توصیف می کنیم اغلب تمایل داریم بگوئیم که این نوع مشابه نوع دیگری است به عنوان مثال در C انواع int ، long، short و char ، شکل های گوناگونی از صحیح هستند و رفتار آنها یکسان است و علاقه داریم که عملیاتی مانند جمع، ضرب، به طور یکسان تعریف شود. اگر نوعی به عنوان بخشی از نوع بزرگتر باشد، آن را زیر ن.ع و. به نوع بزرگتر ابر نوع یا super Type گویند.به عنوان مثال در پاسکال می توان زیر بازه را ایجاد کرد. به این شکل

type SmallIntege 1…20

نوع زیربازه SmallInteger زیر نوع صحیح است که مقادیرش از ۱ تا ۲۰ است.

در زبان هائی مثل C و پاسکال به خاطر رسیدن به سرعت بالای اجرا و استفاده بهینه از حافظه از زیر نوع استفاده می کنند.

چگونگی پیاده سازی انواع داده اولیه

الف- چگونگی ذخیره اطلاعات و یا باز یابی حافظه: برای ذخیره و بازیابی انواع داده اولیه معمولا از سخت افزار کمک گرفته میشود. به عنوان مثال اعداد Integer با سخت افزار پشیبانی میشوند. در صورت عدم استفاده از سخت افزار از تکنیک های شبیه سازی(software simulation) استفاده میشود.(مثلا شبیه سازی اعداد 24 رقمی در صورتی که سخت افزار حد اکثر اعداد 12 رقمی را پشتیبانی می کند)

ب- چگونگی پیاده سازی عملیات : در این مورد عملیات به چند روش به اجرا در می آیند

۱- مستقیما از طریق سخت افزار مثل جمع دو عدد صحیح

۲- به عنوان یک تابع یا پروسجر از طریق شبیه سازی نرم افزاری مثل دستور SQRT برای گرفتن جذر اعداد

۳- به عنوان دنباله ای از دستورات که مستقیما از طریق سخت افزار پیاده سازی میشود و به صورت کد در برنامه برنامه نویس قرار داه میشود. مثال

ABS(x) = if x<0 then –x else x
a) Fetch value of x from memory
b) If x>0 skip next instruction
c)Set x=-x
d) Store x in memory

دستورات اعلان(Decleratoon):

مجمو عه ای از دستورات زبان برنامه سازی است که به نحوی ارتباط بین برنامه نویس و برنامه مترجم را جهت تولید کد لازم برای اجرا فراهم می کند و اطلا عاتی را در مورد (type) و تعداد و سایر موارد متغیر ها و ثابت ها و ... را فراهم می کند.به طور کلی اعلان بر دو نوع می باشد.

الف) صریح: در این روش نوع و نام شئی داده ای دقیقا توسط برنامه نویس تعیین می گردد. برای مثال در زبان C اعلان float A,B; نشان می دهد که دو شئی داده ای نوع float داریم و این اشیاء در طول عمرشان به نام های A و B مقید شده اند.

ب) ضمنی: خود برنامه مترجم یا کامپیوتر پیش فرض هائی را در مورد خصوصیات اشیاء ارائه می دهد. مثل Fortran که متغیر هائی i,j...n را از نوع صحیح در نظر می گیرد

اطلا عاتی که توسط دستورات اعلان بر آورده میشود.

- نام اشیاء داده ای
- نوع اشیاء داده ای
- مقدار اولیه اشیاء داده ای
- صفات مربوط به اشیائ داده ای

علاوه بر این اعلان ها می توانند اطلا عاتی را در ارتباط با عملیات برای مترجم زبان فراهم سازند، به عنوان مثال در هنگام اعلان یک تابع تعداد، ترتیب، نوع پارامتر و نوع نتیجه مشخص می گردد. Function Sub(int x ,float y): Real

اهداف دستورات اعلان(Decleratoon):

۱- **انتخاب چگونگی نمایش اطلاعات در حافظه:** اگر اعلان اطلا عاتی راجع به نوع و صفات متغیر را در اختیار کامپایلر قرار دهد، بهترین نمایش حافظه برای آن انتخاب می شود.

۲- **چگونگی مدیریت حافظه:** با توجه به این که اعلان ها طول عمر متغیر ها را نیز مشخص می کنند، در حین اجرای برنامه، مدیریت حافظه بهتری صورت می گیرد. به عنوان مثال تمام متغیر هائی که در ابتدای زیر برنامه تعریف می شوند(طول عمر یکسانی دارند) در یک بلوک حافظه قرار می گیرند و پس از خاتمه زیر برنامه ، آن بلوک آزاد می شود. اگر متغیر های پویائی وجود داشته باشند که توسط ابزار های تخصیص حافظه مثل عملگر New در C++ و تابع malloc() در C ایجاد شوند، در بلوک حافظه دیگری قرار میگیرند(زیرا طول عمر انها متفاوت است) .

۳- **امکان عملیات چند ریختی(polymorphism):** بسیاری از زبان ها نماد های خاصی مثل "+" را برای انجام عملیات مختلف استفاده می کنند. برای مثال در A+B اگر A، B از نوع صحیح باشند جمع صحیح و اگر از نوع اعشاری باشند جمع اعشاری صورت می گیرد. به این ترتیب عملیات جمع با توجه به نوع آرگومان ها انجام می گیرد. به این

عمل چند ریختی یا overload کردن توابع گفته می شود.زبان ML این مفهوم را با چند ریختی(polymorphism) به طور کامل بسط داده است، به این ترتیب که تابع بر حسب نوع آرگومان ها می تواند پیاده سازی های مختلفی داشته باشد. اعلا نها موجب می گردند تا مترجم در زمان کامپایل عملیاتی که به وسیله نماد خاص انجام می گیرد را با توجه به آرگومان های آن به طور مناسب انتخاب کند.

۴- **کنرل نوع(Type cheking)**: از دید برنامه نویس مهمترین هدف اعلان ها انجام کنترل نوع ایستا به جای کنترل نوع پویا است.

کنترل نوع(Type cheking):

هدف از کنترل نوع بر رسی صحت عملیات بر روی اشیاء داده مثل متغیر ها بر اساس نوع اعلانی که برای آنها شده است می باشد. کنترل نوع در سطح سخت افزار وجود ندارد، مثلا رشته بیتی 10001001 می تواند مبین اطلاعات از هر نوع باشد. کنترل نوع در سطح نرم افزار انجام می شود. و منظور از آن این است که هر عملیاتی که در برنامه انجام می گیرد، تعداد و نوع آرگومان های آن درست باشد.

دو نوع کنترل نوع وجود دارد.

۱- **کنترل نوع ایستا(Dynamic Type Chekig)**: این کنترل در زمان کامپایل انجام می شود.

۲- **کنترل نوع پویا(Statick Type cheking)**: این کنترل در زمان اجراء صورت می گیرد.

نقاط قوت کنترل نوع ایستا

- سرعت اجرای برنامه معمولا زیاد است
- عدم نیاز به به نگهداری اطلاعات در مورد نوع (استفاده بهینه از حافظه).
- تمام مسیر های اجرائی برنامه از جهت نوع چک می شوند. به عبارت دیگر چون چک کردن نوع به بهترین وجه انجام می گیرد، اشکال زدائی برنامه آسانتر می شود.

نقاط ضعف کنترل نوع ایستا

- انعطاف پذیری کم است
- جهت کامپایلر کردن مجزای برنامه ممکن است مشکلاتی پیش بیاید(نیاز به تعریف اعلان برای تمام اشیاء داده ای، کنترل کردن عملییات چند ریختی، ترجمه مستقل زیر برنامه ها و ساختار های خاص کنرل داده)

نقاط قوت کنترل نوع پویا

- انعطاف پذیری زیاد در برنامه سازی
- تعریف اعلان برای هر نوع می تواند نیاز نباشد
- عمل تبدیل نوع ها می تواند در زمان اجرا انجام شود

نقاط ضعف کنترل نوع پویا

- اشکال زدائی برنامه مشکل است(به علت تغییر نوع ها در زمان اجرا).
- فضای اضافی جهت ذخیره نوع شئی داده ای مورد نیاز است.
- سرعت اجرای برنامه به دلیل استفاده از مکانیزم نرم افزاری پایین است.

نکته: در کنترل نوع ایستا، کنترل در زمان کامپایل انجام می شود. در گذر اول کامپایلر، در جدول سمبل ها برای هر شئی داده، نوع و برای هر عملیات تعداد، ترتیب و نوع آرگومان ها مشخص می شود. در گذر دوم عمل کنترل نـوع صـورت می پذیرد.

نکته: زمانی که نوع داده تعریف شده و نوع پارامتر یکی نیستند(Type Mismach) دو را حل وجود دارد

الف- برنامه خطا بگیرد و عملیات مربوط به کنترل خطا اجرا شود

ب- عمل تبدیل نوع صورت گیرد. Conversion : type1 ⟶ Type2

نکته: به طور کلی به زبان هائی که از کنترل نوع پویا استفاده می کنند زبان های TypeLess یا بدون نوع گفته می شود.

جلسه پنجم

استراتژی های برخورد با Type mismatch (Type تعریف شده و Type تعریف شده یکی نیستند)

الف- برنامه Error بگیرد و عملیات خاص مربوط به Handle کردن Error انجام شود.

ب- عمل تبدیل نوع(Type conversation) انجام گیرد. $conversation\_op : type1 \rightarrow type2$

برای انجام عمل تبدیل نوع اغلب زبان های برنامه سازی امکانات زیر را فراهم می سازند

- عمل تبدیل نوع به مجموعه ای از توابع از پیش ساخته شده(Built in Function) در زبان برنامه سازی تعریف شده تا برنامه نویس به صورت صریح از آن استفاده کند، مانند دستور Round در پاسکال(مقدار اعشاری را به عدد صحیح تبدیل می کند)
- عمل تبدیل نوع به صورت اتوماتیک توسط برنامه مترجم انجام شود.مثلا اگر یک تابع برای اعداد مختلط داشته باشیم و این تابع را با نوع عدد صحیح فراخوانی کنیم، کامپایلر عمل تبدیل نوع را انجام می دهد. یا به عنوان مثال دیگر در پاسکال اگر آرگومان های عملیات محاسبات مثل +، ترکیبی از نوع Float و Int باشند، نوع داده Int قبل از انجام عمل جمع به طور ضمنی به نوع Float تبدیل می شود. به این نوع تبدیل، تبدیل نوع ضمنی نیز گفته میشود.

انواع داده اولیه:

- تک مقداری(اسکالر): مانند انواع صحیح، اعشاری، کاراکتری
- چند مقداری(مرکب): مانند آرایه ها که می توانند دارای چند مقدار باشند.

داده اسکالر: اشیاء داده ای که فقط یک خصوصیت واحد دارند و آن مقدار آنهاست، اشیاء داده اسکالر و یا تک مقداری گفته می شود. به طور کلی اشیاء داده اسکالر از معماری سخت افزار پیروی می کند(سخت افزار آنها را می شناسد و مستقیما توسط سخت افزار اجرا می شوند) اما داده های مرکب یا چند مقداری معمولا ساختار های پیچیده هستند که توسط کامپایلر تولید می شوند و توسط سخت افزار مستقیما پشتیبانی نمی شوند.

انواع اسکالر:

- انواع عددی(صحیح، اعشاری،...)
- انواع بولین
- کاراکتر

انواع مرکب:

- آرایه
- فایل ها
- اشاره گر ها

نکته: رکورد نیز چند مقداری است اما رکورد توسط ما تعریف می شود و از انواع داده اولیه نیست.

اعداد ممیز شناور و نحوه پیاده سازی آنها در زبان های برنامه سازی

برای ذخیره و پیاده سازی اعداد ممیز شناور از استاندارد IEEE 754 استفاده میشود. برای این کارر از فرمتی شبیه به نماد علمی استفاده می کنیم. استاندارد IEEE 754 استاندارد 32 بیتی و 64 بیتی را برای اعداد ممیز شناور مشخص می کند. استاندارد 32 بیتی به صورت زیر است.

S	توان(E)	کسر(M)
1	8	23

اعداد شامل سه فیلد هستند

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

بیت E: توان ظاهری هشت بیتی با افزودنی 127، یعنی در هنگام ذخیره سازی توان در حافظه 32 بیتی مقدار 127 به آن افزوده شده و سپس ذخیره می شود.

بیت M: مانتیس23 بیتی است. معمولا اعداد ممیز شناور را به صورت نرمال شده ذخیره می کنند، در مبنای2 عدد نرمال شده باید با ارزش ترین بیت قسمت اعشار باشد، برای مثال 0/00111 نرمال نیست ولی0/11011 نرمال می باشد

نحوه تبدیل عدد اعشاری در پایه 10 به پایه 2 به صورت زیر است

قسمت صحیح را توسط تقسیمات متوالی بر 2 و نوشتن باقیمانده از سمت چپ و در پایان نوشتن آخرین خارج قسمت در سمت راست محاسبه می کنیم. برای قسمت اعشاری از ضرب متوالی در عدد 2 و نوشتن قسمت صحیح حاصل ضرب از سمت با ارزش دی قسمت اعشار استفاده می شود.

مثال. $(4.75)_{10} = (100.11)_2$

$0.75 * 2 = 1.5$

$0.5 * 2 = 1.0$

- برای ذخیره این عدد دودوئی با فرمت IEEE 754 ابتدا باید عدد را نرمال کنیم یعنی باید قسمت صحیح نداشته باشیم. دقت شود که عدد 127 به خاطر توان های منفی است که موقع باز یابی کم می شود.

$(100.11) = 0.10011 \times 2^{3} \rightarrow 3 + 127 = 130$

پس با این حساب نحوه ذخیره عدد (100.11) در حافظه به این شکل می باشد

S	E	M
0	10000010	1001100…0
1	8	23

به خاطر مثبت بودن عدد

به ازاي باقيمانده بيت ها صفر مي گذاريم

در عمل عکس یعنی استخراج عدد ذخیره شده از حافظه در قالب فرمت ممیز شناور و نمایش آن به صورت دهدهی از فرمول $(-1)^{S} \times (0.M)_{10} \times 2^{E-127}$ استفاده می کنیم، پس خواهیم داشت

$$(-1)^{0} \times (\frac{1}{2} + 0 + 0 + \frac{1}{16} + \frac{1}{32}) \times 2^{3} = 4 + 0.5 + 0.25 = 4.75$$

نکته: معمولا از مقدار افزودنی(Exes) برای بالا بردن توان نمایشی زبان از محدوده اعداد خیلی بزرگ تا محدوده اعداد خیلی کوچک استفاده می کنیم معمولا این مقدار افزودنی از رابطه $(2^{E-1}-1)$ بدست می آید که منظور از E تعداد بیت های آن می باشد

نکته: گاهی اوقات با ارزش ترین مقدار مانتیس را حذف کرده تا میزان دقت نمایش داده شده توسط مانتیس یک بیت افزایش یابد. دلیل این کار این است که همیشه عدد نرمال شده در مبنای دو با ارزش ترین بیتش یک می باشد و می توان آن را در حین ذخیره سازی حذف و در زمان باز یابی دوباره اضافه نمود این کار با عمل $(0.M+1)$ قابل انجام است.

انواع داده مرکب

از موارد مختلف داده های مرکب به بررسی ذخیره و پیاده سازی آرایه ها می پردازیم.

ذخیره سازی آرایه ها در حافظه:

آرایه ها دنباله ای از عناصر هستند که همگی از یک نوع هستند(برای آرایه که عناصر متفاوت دارند از لیست پیوندی استفاده می کنیم و با اشاره گر عناصر را به هم پیوند می دهیم).ذخیره نمودن آرایه های تک بعدی(بردار) در حافظه کامپیوتر به سادگی انجام می گیرد، اما آرایه های دو بعدی(ماتریس) و چند بعدی مسئله بغرنج خواهد داشت یکی از وظایف کامپایلر ها نگاشت آرایه ورودی و عناصر آن به آدرس فیزیکی حافظه است که برای این کار نیازمند محاسبات ویژه ای خواهیم بود.

پیاده سازی

اگر α آدرس شروع اولین خانه آرایه در حافظه باشد و E مقدار بایت های لازم جهت ذخیره هر یک از عناصر آرایه باشد و LB حد پایین اندیس آرایه و UB حد بالای اندیس آرایه باشد، محل ذخیره عنصر **I** ام به شکل زیر خواهد بود.

$loc \quad A[i]=\alpha+E\times(I-LB)$ این فرمول برای آرایه هایی تک بعدی می باشد.

اما برای تولید آدرس عناصر یک ماتریس باید ابتدا آن را به بردار تبدیل کنیم، چون که در حافظه کامپیوتر فضای دو بعدی معنا ندارد، این تبدل آدرس معمولا به دو روش ۱- روش سطری(Row maior) ۲- روش ستونی(column maior) می باشد. بسیاری از زبان ها مانند C، پاسکال ، جاوا از روش سطری و فرترن از روش ستونی استفاده می کند.

روش سطری برای آرایه های دو بعدی

فرض کنیم آرایه $A[LB_1.....UB_1][LB_2.....UB_2)$ را داریم فرمول $A[I_1,I_2]=\alpha+disp\times E$ را محاسبه می کنیم که در آن α مبدا ذخیره سازی و E اندازه هر عنصر آرایه است وdisp تعداد عناصر بین عنصر اول آرایه تا عنصر $A[I_1,I_2]$ می باشد. disp را به صورت جداگانه به روش های سطری و ستونی محاسبه می کنیم

روش سطری:

$$disp = (I_1 - LB_1)\times d_2 + (I_2 - LB_2)$$

- $(I_1 - LB_1)$: تعداد سطر هاي كامل قبل از عنصر موجود را ميدهد
- d_2: تعداد ستون ها را مي دهد
- $(I_2 - LB_2)$: تعداد عناصر موجود در همان سطر ناقص را ميدهد

روش ستونی

$$disp = (I_2 - LB_2) \times d_1 + (I_1 - LB_1)$$

تعداد ستون هاي كامل قبل از عنصر مورد نظر را

تعداد سطر ها را مي دهد

تعداد عناصر موجود در همان ستون ناقص را ميدهد

حال با همین روش بالائی می توانیم آرایه سه بعدی را با فرمول $A[I_1, I_2, I_3] = \alpha + disp \times E$ محاسبه کنیم

روش سطری

$$disp = (I_1 - LB_1)d_2 * d_3 + (I_2 - LB_2)d_3 + (I_3 - LB_3)$$

روش ستونی

$$disp = (I_3 - LB_3)d_2 * d_1 + (I_2 - LB_2)d_1 + (I_1 - LB_1)$$

حال با همین روش بالائی می توانیم آرایه n بعدی را با فرمول $A[I_1, I_2, \ldots, I_n] = \alpha + disp \times E$ محاسبه کنیم

روش سطری

$$disp = (I_1 - LB_1)d_2 d_3 \ldots d_n + (I_2 - LB_2)d_3 d_4 \ldots d_n + \ldots + (I_n - LB_n)$$

روش ستونی

$$disp = (I_n - LB_n)d_{n-1}d_{n-2}\ldots d_1 + (I_{n-1} - LB_{n-1})d_{n-2}d_{n-3}\ldots d_1 + \ldots + (I_1 - LB_1)$$

برای حفظ کردن فرمول های فوق بهتر است جملات زیر را به خاطر بسپارید

ابعاد سمت راست ×(حد پایین آن بعد- اندیس) Σ ⇒آدرس سطری(از چپ به راست)

ابعاد سمت چپ ×(حد پایین آن بعد- اندیس) Σ ⇒آدرس ستونی(از راست به چپ)

مثال: آرایه چهار بعدی $A[1\ldots10][1\ldots20][1\ldots10][1\ldots10]$ مفروض است که به روش سطری ذخیره شده است اگر آدرس شروع حافظه صفر باشد و تعداد بایت های هر عنصر آرایه دو باشد آدرس $A[1,2,2,2]$ را بدست آورید.

$$disp = (1-1)20*10*10 + (2-1)10*10 + (2-1)10 + (2-1) = 111$$

$$a[1,2,2,2] = 0 + disp * 2 = 111 * 2 = 222$$

جلسه ششم

به چهار طریق می توان انواع داده جدید و عملیات بر روی آنها تعریف کرد.

1- ساختمان داده: از نظر مجازی تمام زبان ها خواصی برای ایجاد اشیای داده پیچیده از انواع اولیه را دارند، لیست ها، مجموعه ها و آرایه ها راه هائی برای ایجاد دسته ای از اشیاء همگن هستند. رکورد ها مکانیزمی برای ایجاد اشیاء داده غیر همگن هستند.

2- **زیر برنامه ها:** برنامه نویس با ایجاد زیر برنامه ها در واقع یک سری عملیات جدید تعریف می کند. همچنین در برخی زبان ها عملیات به عنوان یک نوع جدید محسوب می شود.

3- **اعلان نوع:** خود برنامه نویس با استفاده از امکانات زبان یک نوع جدید تعریف می کند، مفهوم نوع داده انتزاعی برای ایجاد انواع جدید بکار برده می شود.

4- **وراثت:** مفهوم برنامه نویسی شئی گراء و وراثت امکان ایجاد انواع جدید و عملیاتی برای آن نوع بوجود آورده است.

نوع داده انتزاعی:Abstract Datea Type(A.D.T)

یک ADT شامل یک نوع داده به همراه مجموعه ای از عملیات بر روی آن می باشد این نوع داده از سوی برنامه نویس و توسط امکاناتی که زبان برنامه سازی در اختیار قرار می دهد تعریف و ایجاد می گردد و خود برنامه نویس یک سری عملیات را بر روی آن نوع داده بوجود می آورد. یک زبان برنامه سازی باید امکاناتی جهت ساختن ADT فراهم سازد.

انتزاع(تجرد سازی): یک زبان برنامه سازی باید ایده انتزاع را پشتیبانی کند بدین معنا که ابتدا چه عملی انجام شود را بیان کنیم سپس در سطوح بعدی چگونگی انجام شدن آن عمل تعیین گردد در واقع یک زبان برنامه سازی باید این امکان را فراهم سازد که طراحی نرم افزار به صورت لایه به لایه انجام شود.

مثال: در اعلان یک کلاس در C++ ابتدا نام کلاس و خصوصیات آن و اسامی متد های آن به همراه پارامتر های مربوطه آورده می شود(چه کار بیان می شود) سپس به دنبال آن بدنه متد ها و کاری که باید انجام دهندآورده می شود(چگونگی انجام کار).

پنهان سازی اطلاعات:

در صورتی که یک زیر برنامه یا برنامه ای از زیر برنامه دیگر استفاده کند و به جزئیات برنامه استفاده شونده دستیابی نداشته باشد ایده پنهان سازی اطلاعات پیاده سازی شده است. مورد فوق در بکار گیری داده نیز معتبر است مثلا در مورد تابع Sqrt() جزئیات الگوریتم جذر گرفتن و نحوه نمایش اطلاعات از دید کاربر پنهان است.

دسته بندی اطلاعات:

به مجمو عه ای از مقادیری که یک متغیر می تواند آنها را بپذیرد نوع داده گفته می شود در کل زبان برنامه سازی امکانی را برای اعلان متغیر های آن نوع و عملیات را بر روی آنها تدارک می بیند برنامه نویس بدون این که از جزئیات نمایش حافظه و چگونگی انجام این عملیات باخبر باشد از اشیاء داده متعلق به آن نوع استفاده می کند در واقع برنامه نویس یک نوع داده با عملیات از پیش تهیه شده برای کار بر روی آن مواجه است که نمایش حافظه داده مورد نظر کاملا دسته بندی شده است.

در صورتی که دسته بندی اطلاعات در انتزاع انجام شودنتایج زیر را خواهیم داشت

۱- لازم نیست بدانیم اطلاعات پنهان شده، جزئیات آن چگونه است

۲- اجازه دستیابی مستقیم به اطلا عات پنهان شده وجود ندارد.

زیر برنامه ها به عنوان عملیات تجرد سازی(Subprograms a Abstract operation)

Subprogram ها ابزاری برای ایده پیاده سازی Abstract ها هستند بدین معنا که می توانند لایه های مختلف نرمافزاری رااز هم جدا کنند.

مشخصات زیر برنامه ها

الف) نام(Name)

ب) تعداد پارامترها، ترتیب و نوع آنها

ج) تعداد نتایج، ترتیب و نوع آنها

د) عملیات انجام شده توسط Subprogram

معمولا Subprogram ها به صورت زیاضی بیان می شوند

مثال.

Function FN(X:Real , Y:Integer):Real

*FN: Real*Integer → Real*

مشکلات زیر برنامه ها:

۱- یک Subprogram ممکن است ورودیهای ضمنی داشته باشد مثل متغیر های سراسری

۲- یک Subprogram ممکن است نتایج ضمنی داشته باشد مثل تغییرات ناخواسته بر روی برخی ورودی ها

۳- یک Subprogram ممکن است برای بعضی از مقادیر ورودیها مبهم باشد مثل تقسیم بر صفر

۴- یک Subprogram ممکن است خو تغییر(Self Modification) یا حساس به ماقبل(History Sensivity) باشد.

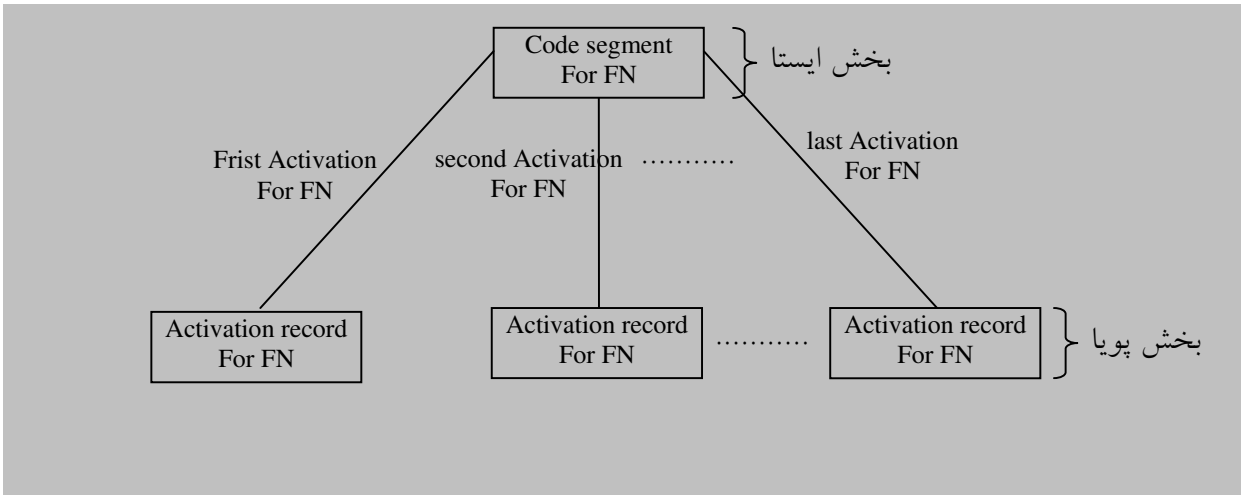
تعریف کردن زیر برنامه و فعال کردن زیر برنامه(Subprogram Definition and Subprogram Activation)

تعریف زیر برنامه اطلا عات و مشخصات لازم را جهت Activation (ایجاد سابقه فعالیت)فـراهم مـی کنـد، تعریـف زیـر برنامه مشابه تعریف یک Type و فراخوانی آن مشابه D.O می باشد، Life Time زیر برنامه از لحظه شروع فراخـوانی تـا لحظه خاتمه یافتن می باشد.

یک سابقه فعالیت ساختمان داده ای با مشخصات زیر جهت نگه داری اطلاعات در هر اجرای زیر برنامه است.

- محلی برای ذخیره پارامترها
- محلی برای ذخیره نتایج زیر برنامه
- محلی برای ذخیره متغیر های محلی
- محلی برای ذخیره مقادیر ثابت و Literal
- محلی برای ذخیره کد تولید شده جهت اجرا

برای فعال سازی یک زیر برنامه دو قسمت ایستا و پویا وجود دارد در قسمت ایستا که سگمنت کد نام دارد کد اجرائـی و مقادیر ثابت وجود دارد، این بخش در حین اجرای زیر برنامه باید ثابت بماند. یک کپی از آن می تواند برای تمـام سـوابق فعالیت زیر برنامه مشترک باشد، بخش پویا که رکورد فعالیت نام دارد به ازای هر بار فراخوانی زیر برنامه ایجاد خواهد شد و هر بار که فراخوانی پایان پذیرفت از بین حواهد رفت(شکل زیر این موضوع را روشن می کند).



زیر برنامه های چند هدفه(Generic Subprogram)

منظور این است که چندین تا زیر برنامه با اسامی یکسان وجود دارد که تعداد پارامترها، نوع و ترتیب آنها متفاوت است

مثال

```
procedure ENTER(st: section; TAB : var Classlist)
begin
end;
```

```
procedure ENTER(student : integer; SECT : var Section)
begin
end;
```

در مثال فوق مورد اول برای ثبت نام یک دانشجو در یک Section می باشد و مورد دوم یک Section به لیست کلاس ها اضافه می کند. این گونه موارد را زیر برنامه ای کلی یا چند هدفه گویند که از جهت پیاده سازی کامپایلر با توجه به تعداد پارامتر ها، نوع و ترتیب انها تشخیص می دهد کدام یک از زیر برنامه ها را فرا خوانی کند و کد لازم برای فراخوانی زیر برنامه مربوطه را تولید کند.

تعریف نوع(Type Difinition):

یک زبان برنامه سازی باید امکاناتی جهت تعریف Type جدید با توجه به Type های موجود در زبان فراهم سازد. هر Type شامل یک Name و شامل یک Description می باشد.

مثال. *type s : Array[1…10] of Real*

مزایای تعریف Type

۱- ساده تر کردن ساختار برنامه

۲- جلوگیری از تکرار Type های موجود در زبان

۳- ساده تر کردن انتقال پارامتر ها

۴- یک شکل جدیدی از بسته بندی و پنهان سازی اطلاعات را بوجود می آورد

مقدار راست یک شئی داده: منظور از آن محتوا یا مقدار شئی داده است.

مقدار چپ یک شئی داده: منظور از آن آدرس شئی داده است.

هم ارزی نوع:

کنترل نوع چه به صورت ایستا و چه به صورت پویا مقایسه بین نوع آرگومان واقعی و نوع دادهای است که عملیات انتظار آن را دارد. اگر انواع یکسان باشند آرگومان پذیرفته می شود و عملیات ادامه می یابد ولی اگر یکسان نباشند یا خطا محسوب می شود یا تبدیل ضمنی صورت می گیرد و کار ادامه می یابد.

منظور از یکسان بودن انواع چیست(دو Type در چه صورت با یکدیگر معادل هستند)؟

معادل بودن Type به دو نوع تقسیم می شود

۱- **معادل بودن نام:**دو تا Type در صورتی یکسان هستند که نام آنها یکسان باشد،

برای مثال اگر قطعه کد زیر را داشته باشیم. در این صورت اگر یک زبان برنامه سازی از هم ارزی نوع یا معادل بودن نام استفاده کند آنگاه SUB(Y) نمی تواند انجام بگیرد.مزیت اصلی این نو هم ارزی سادگی پیاده سازی آن است.

```
Type VECT1:Array[1…10] of Real;
VECT2 :Array[1…10] of Real;
Var X,Y : VECT1;
Y :VECT2;
procedure SUB (A:VECT1);
Begin
⋮
End
Begin
X := Y;
SUB(Y);
END.
```

نقاط ضعف معادل بودن نام

- هر Type باید یک Name داشته باشد و Type های بدون نام نمی توان داشت در این حالت انتقال پارامتر دچار مشکل می شود

مثال. در این جا w را نمی توان به SUB() انتقال داد

Var W : Array [1…10] *of Real*;

- یک تعریف نوع باید در سراسر برنامه یا بخش بزرگی از برنامه قابل استفاده باشد

۲- **معادل بودن ساختار(Structural Equivalence)**

در این حالت دو Type معادل هم هستند اگر ساختار آنها معادل(مساوی) هم باشد و جزئیات تمام مولفه ها یکسان باشد. عمده مزیت این نوع هم ارزی انعطاف پذیری بالای زبان برنامه سازی است.

نقاط ضعف معادل بودن ساختار

- جهت تشخیص معادل بودن Type باید هزینه پرداخت شود یعنی زمان از دست می دهیم، کامپایلر باید زمان صرف کند که آیا دوType معادلند یا خیر؟
- دو متغیر ممکن است به طور تصادفی از نظر ساختاری یکسان باشند حتی اگر برنامه نویس آنها را انواع جداگانه تعریف کرده باشد.

مثال.

Type METERS = Integer;
LITERS = Integer;
Var Len : METERS;
VOL : LITERS;
X := LEN + VOL;

از دید زبان برنامه سازی LEN+VOL صحیح است و هیچ خطائی از طرف کامپایلر گرفته نمی شود چون ساختارشان یکسان است در حالیکه برنامه نویس می خواهد زبان برنامه سازی به او کمکی کند. حتی زمانی که چندین برنامه نویس بر روی یک برنامه کار می کنند این اشتباه ممکن است پیش باید.

جلسه هفتم

کنترل ترتیب اجرای برنامه

کنترل ترتیب اجرای برنامه به چهار دسته تقسیم می شود.

۱- ساختار استفاده شده در عبارات ریاضی مثل قواعد تقدم عملگرها

۲- ساختار استفاده شده بین دستورات یا گروهی از دستورات، مانند دستورات شرطی و حلقه های تکرار

۳- ساختار های استفاده شده در زبان هائی مثل پرولوگ که برنامه نویسی اعلانی نیز نامیده می شود. در این نوع زبان ها پیشرفت برنامه بر اساس یک سری تصمیمات منطقی بر اساس شرایط خاص است.

۴- ساختار های استفاده شده بین زیر برنامه ها. مثل فراخوانی زیر برنامه ها و همروال ها که موجب انتقال کنترل از نقطه ای به نقطه دیگر می شود.

از یک نگرش دیگر می توان کنترل ترتیب اجرای دستورات را به دو دسته تقسیم کرد.

الف. ضمنی(Implicit): منظور از ضمنی، ترتیب اجرای دستورات به صورت پیش فرض توسط زبان برنامه نویسی است (مثل تقدم عملگر* به+)

ب. صریح(Explicite): منظور از صریح، کنترل ترتیب دستورات توسط برنامه نویس می باشد. مشابه وارد کردن پرانتز در عبارات ریاضی، استفاده از دستور goto، استفاده از دستورات شرطی و ...

کنترل ترتیب اجرا در عبارات ریاضی:

دو روش اصلی در این زمینه وجود دارد که قاب تبدیل به یکدیگر هستند.

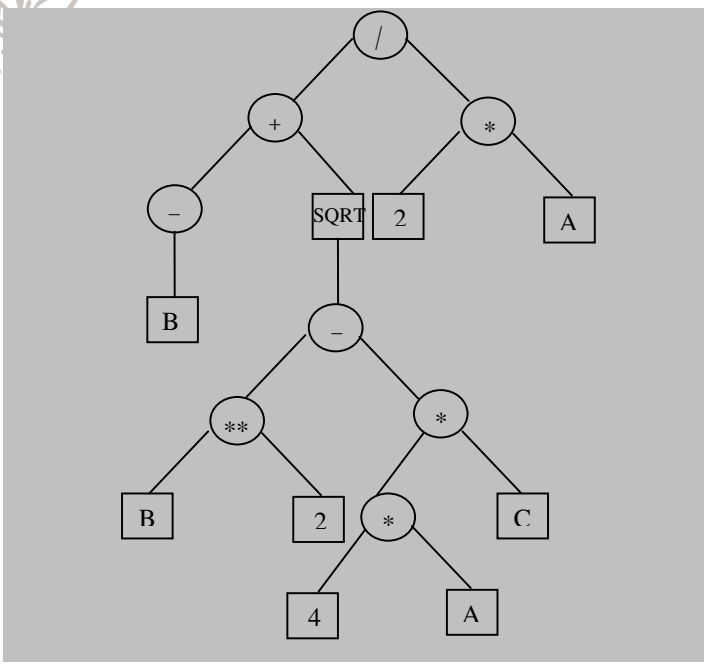
الف. استفاده از ساختار درخت

در این روش ریشه درخت عملیات اصلی، برگ ها داده ها و گره های بین ریشه و برگ ها عملیات میانی را نشان می دهند، برای مثال ساختار درخت فرمول محاسبه ریشه های معادله درجه دوم در شکل صفحه بعد نشان داده شده است.

به هنگام تولید کد توسط کامپایلر، جهت ارزشیابی عبارت، حداقل15 دستور نیاز است، و حال بحث این است که ترتیب اجرای این پانزده دستور چگونه است؟

$$root = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

اما این نحوه ارزیابی ایراداتی دارد همانند نقیض الویت های ارزیابی(این که کدام دستور ابتدا ارزیابی شود) مثلا مشخص نیست که آیا B- قبل و یا بعد از B**2 انجام گیرد.همچنین مشخص نیست که آیا دو ارجاع به B می تواند در یک ارجاع ترکیب شود یا خیر به عبارتی ابهام در کنترل برنامه وجود دارد. برای رفع این ایرادات از نشانه های خاصی استفاده می شود. (مورد بعد)



ساختار درختي براي محاسبه ريشه هاي معادله درجه

الف. استفاده از سه روش پیشوندی، پسوندی، میانوندی

روش میانوندی(infix): همان روشی است که به طور معمول برای نمایش عبارات ریاضی استفاده می گردد. یعنی عملگر بین دو عملوند قرار می گیرد. مثل A+B*C-A

روش پیشوندی(prefix). برای نمایش عبارات ریاضی ابتدا عملگر و سپس دو عملوند ظاهر می شوند.مثلا به این شکل *+AB-CA

روش پسوندی(postfix). در این روش دو عملوند وسپس عملگر در عبارات ظاهر می شوند مثل AB+CA-*

دو نقطه ضعف infix

الف. گرچه infix برایSyntax عبارات مناسب به نظر می رسد ولی یک زبان برنامه سازی تنها نمی تواند شاملinfix باشد، بلکه ترکیبی ازinfix و یکی از دوSyntax دیگر می باشد.

ب. در محاسبه بعضی از عبارات ریاضی ممکن است ابهام به وجود بیاید به عنوان مثال در محاسبه عبارت A*B+C معلوم نیست کدام یک از عمل ها ابتدا انجام شود و ترتیب اجرای هر یک می تواند نتیجه را بسیار متفاوت کند.

ارزیابی مدار کوتاه(اتصال کوتاه در عبارات منطقی):

در ارزیابی مدار کوتاه عبارت، نتیجه عبارت بدون ارزیابی تمامی عملوندها و یا عملگر های آن تعیین می شود. به عنوان مثال. عبارت زیر را در نظر بگیرید $(13*a)*(b/13-1)$

اگر مقدار a صفر باشد، مقدار این عبارت مستقل از $(b/13-1)$ است. یعنی این بخش از عبارت، در مقدار عبارت، تاثیری ندارد. لذا نه تنها نیازی به ارزیابی این بخش نیست، عملگر* دوم نیز لازم نیست انجام شود. اما تشخیص این وضعیت در حین اجر ساده نیست، به همین دلیل اغلب از ارزیابی مدار کوتاه استفاده نمی شود.

به عنوان مثال دیگر توجه کنید $if\ (A=0)\ \ OR\ \ (B/A>C)\ \ Then...$

اگر این دو عبارت را همزمان ارزشیابی کنید در صورت A=0 یک عمل تقسیم بر صفر پیش می آید برای برطرف کردن این مشکل با توجه به این که اگر A مساوی صفر باشد، ارزشیابی عبارت سمت راستی هیچ تاثیری در نتیجه نخواهد داشت پس از مفهوم اتصال کوتاه استفاده می کنیم و اصلا عبارت سمت راستی را ارزشیابی نمی کنیم تا این که خطائی رخ دهد.

به عنوان مثال دیکر توجه کنید $While(I<UB)\ And\ (V[I]>0)\ \ DO$

اگر این دو عبارت را همزمان ارزشیابی کنید وI خارج از حد آرایه باشد هم خطا رخ خواهد داد، برای برطرف کردن این مشکل در صورتی که عبارت اولی False باشد، ارزیابی عبارت دوم تاثیری در نتیجه نخواهد داشت پس از مفهوم اتصال

کوتاه استفاده می کنیم و اصلا عبارت سمت راستی را ارزشیابی نمی کنیم تا این که خطائی رخ دهد،
یک آلترناتیو در زبان ادا برای رفع مشکل مثال دومی(خطا ی تقسیم بر صفر به خاطر صفر بون A)می تواند به شکل زیر باشد

$$if\ (A=0)\ \ OR\ \ else\ \ (B/A>C)\ \ Then...$$

ملاحظه می شود که در صورت صفر بون A (درست بودن عبارت اول)عبارت سمت راستی ارزیابی نمی شود تا خطای تقسیم بر صفر پیش بیاید.

کنتر ترتیب اجرا با استفاده از دستورGoto

دستور goto در زبان های اولیه بیشتر مورد استفاده قرار می گرفت اما با ایجاد مفهوم برنامه نویسی ساخت یافته استفاده از آن محدود شد و توصیه شد که تا حد امکان از آن استفاده نشود.

مفهوم برنامه نویسی ساخت یافته:

منظور این است که هر قسمت از برنامه تنها یک نقطه ورود و تنها یک نقطه خروج داشته باشد. یعنی هر قسمت برنامه تنها یک عمل مشخص را انجام دهد یکی از راههای رسیدن به این ساختار عدم استفاده از دستورgoto می باشد.

مزایای Goto

- استفاده مستقیم از سخت افزار و داشتن کارائی بالا در برنامه
- ساده و راحت در استفاده برای برنامه های کوچک
- آشنا بودن برنامه نویس با آن مخصوصا برنامه نویسان اسمبلی
- یک بلاک از برنامه با استفاده ازgoto می تواند چندین هدف را سرویس دهد.

معایب Goto

- ایجاد نقص و یا ضعیف کردن ساختار برنامه به صورت سلسله مراتبی
- ترتیب دستورات در متن برنامه لزوما با ترتیب اجرای آنها منطبق نیست
- مجموعه ای از دستورات ممکن است چندین هدف را سرویس دهد و این بر خلاف برنامه نویسی ساخت یافته است چرا که در ساخت یافته باید هر قسمت از برنامه تنها یک عمل خاص را انجام دهد.

استفاده از دستور goto در شرایطی اجتناب ناپذیر است که در زیر برخی از آنها آورده می شود

- **از یک حلقه چندین حالت خروج داشته باشیم**

```
for i:=1 to k do
if Vect[i]=0 then goto a (a is outside the loop)
End.
```

- **تکرار قسمتی از دستورات** : یک حلقه تکرار وجود دارد که در همه حالات کل بدنه برای برای همه دفعات به غیر از دفعه آخر که از حلقه خارج می شویم اجرا می گردد در این حالت هم استفاده از دستورgoto اجتناب پذیر است.

```
Loop
    Read (x)
    if x=0 then goto a (outside the loop)
     process (x)
End Loop
```

- **شرایط استثناء:** شرایط استثناء که در حین اجرای برنامه پیش می آید مانند تقسیم برصفر، overflow، underflow و ... با استفاده از دستور goto کنترل برنامه به قسمتی منتقل می شود که به آن راه انداز استثناء گویند، وظیفه این قطعه کد پیگیری اجرای برنامه به همراه پیغام های مناسب است.

کنترل ترتیب زیر برنامه ها(Subprogram Sequence Control)

در کنترل ترتیب زیر برنامه ها فراخوانی یک زیر برنامه از برنامه و یا زیر برنامه دیگر و برگشت از زیر برنامه مطرح است که در اکثر زبان های برنامه سازی به ترتیب با دستور های Call و Return انجام می شود. ساده ترین نوع کنترل زیر برنامه دارای ساختاری به نام ساختار **فراخوانی-بازگشت ساده**(call-Return) می باشد این ساختار کنترلی توسط قاعده کلی بیان می شود، اثر دستور فراخوانی زیر برنامه مثل این است که قبل از اجر یک کپی از زیر برنامه که فراخوانی شده است در نقطه ای که فراخوانی صورت می گیرد قرار داده شود اما در این حالت ساده پنج فرض از سوی طراح زبان در نظر گرفته می شود که عبارتند از

۱- زیر برنامه ها نمی توانند بازگشتی باشند(خودش را فراخوانی کند)

۲- نیاز به دستور فراخوانی صریح زیر برنامه است اما در پردازش استثناء هیچ فراخوانی صریحی وجود ندارد.

۳- زیر برنامه ها در هر فراخوانی باید به طور کامل اجرا شوند اما در همروال ها(coroutine) ممکن است زیر برنامه ای به طور کامل اجرا نشود.

۴- به محض اجرای Call کنترل به زیر برنامه داده می شود و پس از اجرای زیر برنامه کنترل به نقطه فراخوانی بر میگردد.

۵- در هر زمان فقط یک زیر برنامه کنترل را در دست دارد، اگر اجرا در نقطه ای متوقف شد بقیه یا هنوز فراخوانی نشده اند یا اجرای آنها کامل شده است

اما زیر برنامه هائی که به عنون یک Task مورد استفاده قرار می گیرند ممکن است به طور همزمان اجرا شوند به طوریکه چندین زیر برنامه در آن واحد در حال اجرا باشند، یعنی اگر یکی از زیر برنامه ها متوقف شد ممکن است چندین زیر برنامه دیگر در حال اجرا باشند.

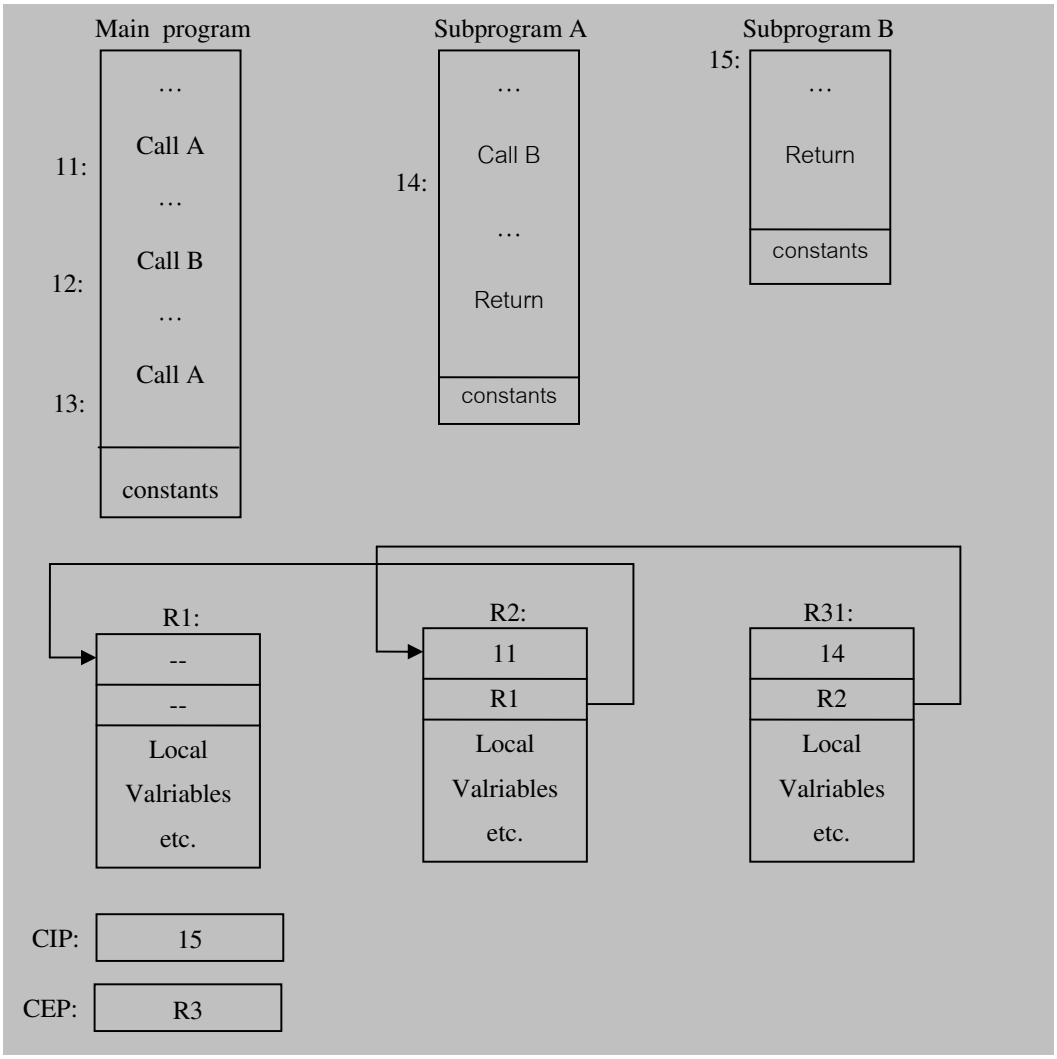
ساختار دیگری هست که مقید به پنج فرض قاعده در این ساختار بین تعریف زیر برنامه(متن زیر برنامه) و سابقه فعالیت آن تفاوت گذاشته می شود، تعریف آن چیزی است که در برنامه می نویسیم و به یک قالب ترجمه می شود اما سابقه فعالیت در هر بار فراخوانی زیر برنامه با استفاده از قالبی که از تعریف ایجاد شد بوجود می آید.

مفاهیم مربوط به کنترل ترتیب زیر برنامه در ساختار فراخوانی- بازگشت

اشاره گر دستور فعلی(CIP): در هر نقطه در حین اجرا، دستوری در سگمنت کد وجود دارد که فعلا توسط کامپایلر یا مفسر در حال اجراست این دستور را دستور فعلی می نامیم و اشاره گری به نام **اشاره گر دستور فعلی** یا CIP به آن اشاره می کند. مترجم، دستوری را کهCIP به آن اشاره می کند بازیابی کرده آن را اجرا می کند.

اشاره گر محیط فعلی(CEP): اشاره گری است که به رکورد فعالیت فعلی(مربوط بـه قطعـه کـد برنامـه ای کـه در خـال اجراست) اشاره می کند، چون تمام سابقه های فعالیت یک زیر برنامه از یک سگمنت کد استفاده می کنند، دانستن دسـتور فعلی کافی نیست، باید اشاره گرCEP هم باشد تا سابقه فعالیت مورد نظر را مشخص نماید به عنوان مثال وقتی دستوری در کد، به متغیرX مراجعه می کند، آن متغیر در رکورد فعالیت وجود دارد، هر رکورد فعالیت آن زیر برنامه، شئی داده ای به نام X دارد که ممکن است محتویاتش با دیگری متفاوت باشد. در زمان شروع اجرای برنامه اصلی بـه رکـورد فعالیـت برنامـه اصلی اشاره می کند وCIP به اولین دستور از سگمنت کد برنامه اصلی اشاره می کند، وقتی کنترل اجرا به دستور فراخوانی زیر برنامه رسید یک رکورد فعالیت از آن زیر برنامه ایجاد می شود وCEP بـه آن اشـاره مـی کنـد وCIP بـه اولـین دسـتور سگمنت کد زیر برنامه اشاره خواهد کرد در زمان فراخوانی باید مقادیر فعلی CIP وCEP در رکـورد فعالیـت برنامـه اصـلی ذخیره شوند تا در زمان بازگشت، اجرای زیر برنامه اصلی از نقطه بعد از فراخوانی زیر برنامه از سر گرفته شود.

شکل زیر این واقعیت را به تصویر گشیده است(حالت اجرا را در شروع زیر برنامه B نشان می دهد)



صفات کنترل داده ها

ویژگی های کنترل داده های یک زبان برنامه سازی آن بخش هائی از زبان است که در نقاط مختلفی از اجـرای برنامـه بـه داده ها دستیابی دارند، وقتی در حین اجرا به عملیاتی رسیدیم داد های مورد نیاز آن عملیات باید آماده باشند، ویژگی های کنترل داده ها در یک زبان تعیین می کند که داده ها چگونه برای عملیات آماده می شوند و نتایج عملیـات بایـد ذخیـره و توسط عملیات بعدی باز یابی شوند برای مثال برای عبارتX:=Y+2*Z وظیفه کنترل داده ها تعیین این نکته اسـت کـه مـثلا در هر اجرا کدامY استفاده شود، زیرا ممکن استY یک متغیر محلی یا غیر محلی باشد.

را ههای استفاده از یک عملوند برای یک عملیات:

۱- **انتقال مستقیم**(Direct Transmission): در هر نقطه از زیر برنامه نتیجه یک عمل به طور مـستقیم بـرای عمـل بعـدی استفاده می شود. به عنوان مثال: نتیجهZ*2 برای عملADD به عنوان یکOperand در محاسبه عبارتX:=Y+2*Z بـه کـار گرفته می شود.

۲- **مراجعه به عملوند(شئی داده) از طریق نـام آن**(Referencing throu a named D.O): هنگـامی کـه شـئی داده ای ایجاد می شود ممکن است نامی به آن اختصاص یابد و آن نام ممکن است به عنوان عملوند یک عملیـات مـورد اسـتفاده قرار گیرد. از طرف دیگر ممکن است یک شئی داده جزئی از شئی داده دیگری باشد که دارای نام اسـت، بـه طـوری کـه شئی داده بزرگتر با یک عملگر انتخاب، آن شئی داده را به عنوان یک عملوند تعیین کند. مثال: فراخوانی یک تابع و یا زیر روال در یک برنامه مثل X:=Y+FN(z) که تابع FN() با شئی داده z فراخوانی شده است

مفهوم وابستگی و محیط های ارجاع(مهم)

وابستگی: انقیاد هر نام به یک شئی داده یا هر نام به یک زیر برنامه خاص را وابستگی گویند. شئی داده یا زیر برنامه → اسم خاص

در آغاز برنامه اصلی در تعریف متغیر ها، هر متغیر را به یک شئی داده مقید می کنیم و با ایـن کـار ایجـاد وابـستگی بـین شناسه ی تعریف شده و شئی داده مربوط به آن ایجاد می کنیم .

نکته. منظور از شناسه همان اسمی است که برنامه نویس به متغیر یا زیر برنامه می دهد.

در حین فراخوانی یک زیر برنامه با استفاده از نام آن هم وابستگی بین نام زیر برنامه و شئی داده ی زیر برنامـه ایجـاد مـی کنیم و هم برای متغیر های تعریف شده در زیر برنامه وابستگی بین اشیای داده با نام متغیر ها را ایجاد می کنیم.

مفهوم محیط ارجاع:

وقتی برنامه اصلی اجرا شود عملیات ارجاعی را فراحوانی می کند تا شئی داده یا زیر برنامه وابسته به یک شناسه را تعیـین کند، وقتی زیر برنامه فراخوانی شد وابستگی های جدیدی در داخل زیر برنامه بین شناسـه های داخـل آن و اشیـاء داده

مربوطه ایجاد می شود. پس از اتمام اجرای زیر برنامه این وابستگی ها از بین رفته و برنامه اصلی با همان وابستگی های ایجاد شده در اول برنامه ادامه می یابد در طی ایجاد این وابستگی ها برای شناسه ها و به کار گیری آنها و نیز دز نهایت از بین رفتن آنها مفهوم اصلی کنترل داده ها دیده می شود، هر برنامه یا زیر برنامه در حین اجرا یک مجموعه شناسه به همراه وابستگی آنها به اشیاء داده شان دارد که برای ارجاع از آنها استفاده می کند این مجموعه از وابستگی شناسه ها را محیط ارجاع برنامه یا زیر برنامه می گویند.

به طور خلاصه هر زیر برنامه ای دارای مجموعه ای شناسه می باشد که در طول اجرا از آنها استفاده می کند به این مجموعه **محیط ارجاع** گویند.

تعریف دیگر. به مجموعه ای از وابستگی هائی گویند که در اعلان برنامه ها یا زیر برنامه ایجاد می شود.

محیط ارجاع زیر برنامه در حین اجرا متغیر نیست این محیط ارجاع در حین ایجاد رکورد فعالیت زیر برنامه ایجاد و تنظیم می گردد. و با از بین رفتن رکورد فعالیت آن(زمان اتمام زیر برنامه) از بین می رود. مقادیر موجود در اشیاء داده ممکن است تغیر کند ولی وابستگی نام ها به اشیاء داده و زیر برنامه تغیر نمی کند.

محیط ارجاع زیر برنامه ممکن است چند جزء داشته باشد(انواع محیط ارجاع)

محلی ارجاع محلی: مجموعه ای از وابستگی ها که به هنگام ورود به زیر برنامه ایجاد می شوند مثل **پارامتر های زیر برنامه، متغیر های محلی**، و **زیر برنامه های داخل آن** که فقط در آن زیر برنامه تعریف شده اند را محیط ارجاع محلی گویند.

محیط ارجاع غیر محلی:

تعریف استاد: مجموعه ای از وابستگی های مربوط به شناسه هائی که در زیر برنامه استفاده می شوند ولی به هنگام ورود به‌آن ایجاد نمی شوند. را محیط ارجاع غیر محلی گویند. برای نمونه در زبان پاسکال اگر تابع F در داخل تابع g تعریف شده باشد، محیط ارجاعی غیز محلی برای F محیط ارجاعی محلی برای g می باشد.

محیط ارجاع سراسری(عمومی)

به مجموعه وابستگی های ایجاد شده در ابتدای برنامه اصلی گفته می شود، محیط عمومی بخشی از محیط ارجاع غیرمحلی است.

مثال. در برنامه زیر انواع محیط های ارجاع را مشخص کنید.(صفحه بعد)

```
program    main ();
var   A , , B , C : real ;
procedure    sub 1( A : real );
var   D : real ;
procedure    sub 2( C : real );
var   D : real ;
begin
⋮
C := C + B;
⋮
end ;
begin
⋮
sub 2( B);
⋮
end ;
begin
⋮
sub 1( A);
⋮
end .
```

محیط های ارجاع برای *sub*2

محیط ارجاع محلی: C , D

محیط ارجاع غیر محلی: A , *sub*2 *in sub*1 , B , *sub*1 *in main*

محیط های ارجاع برای *sub*1

محیط ارجاع محلی: A , D , *sub*2

محیط ارجاع غیر محلی: B , C, *sub*1 *in main*

محیط ارجاع *main* **(سراسری)**

A, B , C , *sub*1

نکته۱: هر زیر برنامه جزء محیط ارجاع غیر محلی خودش است

نکته۲: محیط ارجاع غیر محلی ⊂ محیط ارجاع عمومی

محیط ارجاع از پیش تعریف شده: بعضی از شناسه ها وابستگی هائی از پیش تعریف شده دارند و هر زیر برنامه یا برنامه می تواند بدون ایجاد صریح آنها از آنها استفاده کند مانند Mainint در پاسکال. یا مانند کلمات خاص(رزروی یا کلیدی) هم چنین عملیلت اولیه ای مانند +,- نیز به طریقی محیط ارجاع از پیش تعریف شده را می رسانند.

مفهوم قابلیت مشاهده:

گفته می شود شناسه x در زیر برنامه یا برنامه f قابل مشاهده است اگرx قسمتی از محیط ارجاع f را تشکیل دهد. به عبارت دیگر وابستگی شناسه x به یک شئی داده در حین ورود به زیر برنامه f تعیین شده باشد، در نقطه مقابل مفهوم پنهان بودن است، اغلب وقتی وابستگی مخفی است که زیر برنامه ای شناسه ای را که در جای دیگری در حال استفاده است، دوباره تعریف کند.

مفهوم عملیات ارجاعی:

یک عملیات ارجاع عملیاتی است که یک شناسه و یک محیط ارجاع را گرفته، شناسه را در محیط پیدا کرده و یک شئی داده و یا زیر برنامه را بر می گرداند.

ارجاع محلی ، غیر محلی و سراسری: ارجاع به یک شناسه می تواند محلی، غیر محلی و یا سراسری باشد در صورتی که

محلی: اگر عملیات ارجاع، شناسه را در محیط ارجاع محلی پیدا مند

غیر محلی و یا سراسری: اگر عمیات ارجاع، شناسه را در محیط غیر محلی و یا سراسری بیابد.

مفهوم حوزه پویا و قاعده حوزه پویا

حوزه پویای وابستگی مربوط به یک شناسه مجموعه ای از سابقه های فعالیت زیر برنامه است که وابستگی مربوط به شناسه مورد نظر در حین اجرای زیر برنامه قابل مشاهده است، هر وابستگی دارای یک حوزه پویا است که آن بخشی از اجرای برنامه است که به عنوان قسمتی از محیط ارجاع برنامه است.

قاعده حوزه پویا: حوزه پویای هر وابستگی را بر حسب حالت پویای اجرای برنامه تعریف می کند. به عنوان مثال فرض می کنیم یک وابستگی برای شناسه x در حین ورود به زیر برنامه f ایجاد شده است، قاعده حوزه پویا بیان می کند که از زمان اجرای f جوزه پویای وابستگی x علاوه بر خود سابقه فعالیت f شامل زیر برنامه های دیگری است که توسط f فراخوانی می شود و حتی اگر زیر برنامه دیگری توسط زیر برنامه ای فراخوانی شود که f آن را فراخوانی کرده است شامل آن نیز می شود.

زنجیره پویای سابقه های فعالیت زیر برنامه f، شامل خود f ، زیر برنامه های که f آن را فراخوانی کرده است و نیز زیر برنامه فراخوانی شده توسط زیر برنامه فراخوانی شده توسط f می باشد و ...

حوزه ایستا: وقتی متن برنامه را می بینیم متوجه می شویم که یک ارتباط دیگر بین تعریف یک شناسه و وابستگی آن به صورت ایستا وجود دارد.

در واقع در ابتدای زیر برنامه f تعریف شناسه x یک وابستگی ایستا بین x و نوع آن به وجود می آورد و گفته می شود حوزه ایستای یک اعلان ازx بخشی از متن برنامه است که استفاده ازx ارجاع به اعلان ابتدای f دارد و قاعده حوزه ایستا قاعده تعیین حوزه ایستای یک تعریف از شناسه در ابتدای برنامه یا زیر برنامه است.

به عنوان مثال در پاسکال قاعده حوزه ایستا تعیین می کند که ارجاع متغیر x در برنامهF به اعلان x در آغاز F اشاره می کند یا اگر در آنجا اعلان نشده باشد به اعلانی از x در آغاز زیر برنامه q اشاره دارد که خود تعریف زیر برنامه f داخل برنامه q می باشد و q مثلا داخل N و N داخل ...

به طور کلی قاعده حوزه ایستا ارجاع ها را به اعلان اسامی در متن برنامه مربوط می کند ولی قاعده حوزه پویا ارجاع ها را با وابستگی های اسامی(فراخوانی ها) در حین اجرای برنامه ربط می دهد همواره ایده ال این است که بین دو قاعده ایستا و پویا سازگاری وجود داشته باشد. یعنی در هر دو حالت وابستگی تعیین شده برای شناسه x یک زیر برنامه یکسان باشد که برقرای این حالت مشکل است.

جلسه نهم

مثال برای قواعد حوزه پویا و ایستا

الف. فرض می کنیم در این حالت زبان از قواعد حوزه پویا برای یافتن وابستگی های متغیر استفاده می کند دراین حالت هنگام اجرای دستور $x := x+1$ ابتدا دنبال وابستگی x در داخل خود پروسجر R می گردیم، مشاهده می شود که هیچ وابستگی برای متغیر x داخل R تعریف نشده است در این حالت قاعده پویا بیان می دارد که برای یافتن وابستگی x سراغ پروسجری خواهیم رفت که R توسط آن فراخوانی شده است(Q). و مشاهده میشود که داخلQ، x از نوع $Real$ تعریف شده است.

ب. در قاعده حوزه ایستا بعد از عدم موفقیت در یافتن وابستگی x در در داخل R سراغ پروسجری خواهیم رفت که R داخل آن پروسجر تعریف شده است($main$).

```
program main();
var x, y : integer;
procedure R;
var y : real;
begin
⋮
x := x+1;
⋮
end R;
procedure Q;
var x : real;
begin
⋮
R; // call R
⋮
end Q;
```

```
procedure P;
var x : Boolean;
begin
⋮
Q // call Q
end p
begin main
⋮
p;
⋮
end.
```

قواعد حوزه برای داده های محلی و غیر محلی

برای محیط های محلی قواعد حوزه ایستا و پویا با هم سازگارند، قاعده حوزه ایستا مشخص می کند که ارجاع به شناسه x در بدنه زیر برنامه Q (در برنامه بالائی) به اعلان محلی x در عنوان زیر برنامه Q ربط پیدا می کند، قاعده حوزه پویا مشخص می کند که در حین اجرای Q ارجاع به x به وابستگی x در سابقه فعالیت فعلی Q اشاره می کند حتی اگر چند سابقه فعالیت از Q وجود داشته باشد در حال حاضر فقط یکی در حال اجراست.

دو رویکرد برای ارجاع محلی وجود دارد(رویکرد در مورد پیاده سازی)

روش نگهداری(Retention): در این روش، در اولین ورود به زیر برنامه متغیر تعریف و ایجاد می شود. در موقع بازگشت از زیر برنامه متغیر از بین نمی رود و در فراخوانی های بعدی از آخرین مقدار متغیر استفاده می گردد. مانند زبان های fortran و Cobol.

روش حذف(Deletion): در این روش متغیر در اولین ورود به زیر برنامه ایجاد می شود و به هنگام بازگشت از بین می رود مانند $Ada, Lisp, pascal$ و C

نکته: $Algol$ و $pl/1$ از هر دو روش استفاده می کنند.

امتیاز روش نگهداری: این روش به برنامه نویس اجازه می دهد تا زیر برنامه هائی ایجاد کند که نسبت به گذشته حساس باشند به طوریکه بخشی از نتایج آنها در هر فراخوانی توسط ورودی و بخشی دیگر توسط داده های محلی تعیین شود که

در حین سابقه فعالیت قبلی ایجاد شده اند، این در حالی است که در روش حذف برای انتقال داده هائی از یک فراخوانی به فراخوانی دیگر(از همان زیر برنامه) باید یک متغیر به صورت غیر محلی ایجاد شود.

امتیاز روش حذف: برای زیر برنامه های بازگشتی روش حذف روش متداول تری است. روش حذف موجب صرفه جوئی در حافظه می شود.

محیط ها و اطلاعات مشترک

همیشه مواردی پیش می آید که در حین اجرای زیر برنامه ها یک سری اطلاعات مشترک وجود داشته باشد که بین زیر برنامه ها به اشتراک گذاشته می شود. معمولا به چهار طریق این امکان را فراهم می کنند تا زیر برنامه ها به اطلاعات مشترک (محیط مشترک) دسترسی داشته باشند.

الف. محیط اشتراکی صریح(Explicit): هر زیر برنامه باید اعلان صریح از داده اشتراکی داشته باشد مانند کلاس ها در++C و بلوک های Common در فرترن

ب. محیط اشتراکی ضمنی(Implicit): مثل برخی ثوابت مثلا Maxint در پاسکال

ج. قلمرو پویا(Dynamic scope)

د. قلمرو ایستا و ساختار بلاکی

تعریف محیط های مشترک: یک راه ارتباط زیر برنامه ها با یکدیگر است، روش متداول تر جهت انتقال اشیاء داده بین زیر برنامه ها استفاده از تکنیک انتقال پارامتر هاست در این تکنیک اطلاعات بین زیر برنامه ها از طریق قرار دادن آنها در یک سری پارامتر ها و فرستادن آنها به زیر برنامه ی فراخوان می باشد.

برای بررسی این تکنیک نیازمند بررسی یک سری تعاریف اولیه هستیم که در زیر به بررسی آنها می پردازیم.

آرگومان: یک شئی داده است که به یک تابع و یا زیر برنامه فرستاده می شود.

نتیجه: مقداری که توسط زیر برنامه یا تابع فراخوانی شده برگشت داده می شود.

پارامتر های مجازی(صوری):اشیای داده تعریف شده در زیر برنامه مانند x و y در مثال زیر

$$procedure\ \ sub(x:\text{int}eger\ ,\ y:boolean)$$

نکته. پارامتر مجازی یک شئی داده(متغیر) **محلی**، برای زیر برنامه ای است که در آن تعریف شده است.

پارامتر واقعی(اصلی، رسمی) : شئی داده ای که در فراخوانی زیر برنامه می آید مثل i و j در sub(i,j)

اما کاری که در تکنیک انتقال پارامتر ها انجام می گیرد نسبت دادن پارامتر های واقعی به پارمتر های مجازی است که معمولا به دو روش زیر انجام می گیرد.

تناظر موقعیت: در تناظر موقعیت اولین پارامتر واقعی به اولین پارامتر مجازی، دومین پارامتر واقعی به دومین پارامتر مجازی و ... نگاشت می شوند.

تناظر نام: در این روش در زمان فراخوانی زیر برنامه صریحا اسامی پارامتر های اصلی به پارامتر های مجازی نسبت داده می شود مثل زبان Ada $sub(y \Rightarrow B\ ,\ x \Rightarrow 27)$

مکانیزم های ارسال یا انتقال پارامتر ها

بعد از این که پارامتر های واقعی به پارامترهای مجازی نسبت داده شده اند به طرق مختلفی این نسبت دادن ها تفسیر و استفاده می شود. در کل چهار روش که به روش های انتقال پارامتر ها(parametr passing) معروف هستند وجود دارد.

- **فراخوانی با مقدار(call by value):** در این روش یک کپی از مقدار راست پارامتر واقعی(محتوای پارامتر واقعی) در پارامتر مجازی قرار داده می شود در این روش مقدار پارامتر واقعی تغییر پیدا نمی کند و هر تغییر در پارامتر مجازی هیچ تاثیری بر روی پارامتر واقعی ندارد این فراخوانی درC مرسوم است.
- **فراخوانی با نتیجه(call by result):** در این روش زیر برنامه یک سری عملیات انجام داده و نتیجه آن در پارامتر های مجازی قرار داده می شود. در هنگام خروج از زیر برنامه مقدار پارامتر های مجازی در پارامتر های واقعی کپی می شود به این ترتیب مقدار نهائی پارامتر های مجازی مقدار جدید پارامتر های واقعی محسوب می شود. در پاسکال برای قرار دادن نتیجه در داخل یک پارامتر مجازی از کلمه کلیدی var قبل از اسم متغیر در تعریف آرگومان زیر برنامه آورده می شود.
- **فراخوانی باارجاع(call by Reference):** این روش متداول ترین روش انتقال پارامتر هاست دراین روش مقدار چپ پارامتر واقعی(آدرس پارامتر واقعی) در پارامتر مجازی کپی می شود. در داخل زیر برنامه هر ارجاع به پارامتر مجازی ارجاع به پارامتر واقعی محسوب می شود و در هر لحظه هر تغییر در پارامتر مجازی در محل حافظه به پارامتر واقعی(مقدار راست پارامتر واقعی) اعمال می گردد در نهایت در زمان خروج از زیر برنامه و بازگشت به ذیر برنامه فراخوان تغییرات اعمال شده بر روی پارامتر واقعی قابل مشاهده و استفاده است زبان C این نوع فراخوانی را با استفاده از اشاره گر ها پیاده سازی می کند.
- **فراخوانی با نام(call by name):** این روش کمتر مورد استفاده زبان ها می باشد(در Algol اهمیت زیادی دارد) زیر سربار اجرائی بالائی دارد، روش کار چنین است که پارامتر های واقعی تا زمان مراجعه زیر برنامه به پارامتر های مجازی مورد ارزشیابی قرار نخواهند گرفت در هر زمان که پارامتر های مجازی مورد ارجاع قرار گرفت مقدرا پارامتر واقعی متناظر با آن مورد ارزیابی قرار خواهد گرفت.از دید برنامه نویس این مکانیزم معادل call by Reference می باشد.

مثال. در پروسجر procedure sub(x:intege) اگر بخواهیم فراخوانی sub(y) را در متن برنامه اصلی داشته باشیم می توان این گونه فرض کرد که در بدنه روال sub به جای همه x ها y قرار داده می شود حال در هر مراجعه به y یک ارزیابی مجدد از y صورت می گیرد از نظر کاربر نتایج فراخوانی با نام و فراخوانی با مقدار یکی است ولی نتایج میانی متفاوتی از هر دو دیده می شود.

مثال. در برنامه زیر تمامی آرگومان ها به صورت call by name تعریف شده اند. با توجه به قطعه کد زیر خروجی برنامه چه چیزی می باشد.(مهندسی کامپیوتر۷۲-۸۲)

```
program example;
var A: Array[1...10] of integer;
I :integer;
procedure Exchange(x, y: integer);
var temp :integer;
begin
temp ← x;
x ← y;
y ← temp;
end;
Begin(main)
I ← 4;
A[1] ← 8; A[2] ← 6; A[3] ← 4; A[4] ← 2;
Exchang (I , A[I]);
output(I , A[1] , A[2] , A[3] , A[4]);
End.
```

حل.روش فراخوانی با نام مانند روش فراخوانی با ارجاع تغییراتی را که بر روی متغیر ها در زیر روال ها انجام میشود به متغیر های برنامه اصلی اعمال می کند و بنابراین مقادیر I و A[I] یعنی به ترتیب 4 و2 باهم تعویض میشوند. اما در بازگشت از فراخوانی چون 2 در I قرار می گیرد انتساب $A[2] \leftarrow 4$ انجام می شود.

	I	A[I]
قبل از فراخوانی	4	A[4]=2
در هنگام از فراخوانی	$x \leftarrow 4$	$y \leftarrow 2$
قبل از بازگشت	$x = 2$	$y = 4$
بعد از بازگشت	$I = 2$	$A[2] = 4$

پس خروجی برابر است با 2, 8, 4, 4, 2 (از چپ به راست بخوانید)

مثال. برنامه زیر را در نظر بگیرید، مقدار خروجی برنامه فوق با استفاده از روش فراخوانی توسط نام(call bye name) چیست(مهندسی کامپیوتر -۷۳)؟

```
var s: array[1...3] of char;
var i j: integer;
procedure P(x: integer ; y: char)
var j: integer;
begin
j := 2;
x := x+1
output(y);
output(i);
end;
S[1]:='A' ; S[2]:='B' ; S[3]:='C';
i := 0 ; j := 1;
P(i , S(1+ j) );
output(i);
```

	i	$S[1+j]$
قبل از فراخوانی	0	$S[2] = 'B'$
در هنگام از فراخوانی	$x \leftarrow 0$	$y \leftarrow 'B'$
قبل از بازگشت	$x = 1$	$y \leftarrow 'B'$
بعد از بازگشت	$i = 1$	$S[2] = 'B'$

نکته. توجه شود که j در عبارت $S[1+j]$ به متغیر سراسری j اشاره دارد. و در برنامه اصلی مقدار $i = 1$ چاپ می شود. با این توصیف خروجی $'B' , 1 , 1$ می باشد.

مثال. برنامه زیر را در نظر بگیرید، اگر روش انتقال پارامتر به روال با نام باشد مقـدار List[1] ، List[2] و Global بعـد از اجرای برنامه به ترتیب از راست به چپ چند است.

جواب. از راست به چپ 3 ، 5 ، 2

```
procedure bigsub;
Integer Global;
Integer array List[1..2];
procedure SuB(param);
integer param;
begin
param := 3;
Global := Global + 1;
param = 5;
end;
begin
List[1] := 2;
List[2] := 2;
Global := 1;
suB(List[Global];
end.
```

مثال. خروجی برنامه زیر در صورتی که مکانیزم تبادل پارامتر

call by name , *call by refrence* , *call by result* , *call by valu* باشد کدام است(مهندسی کامپیوتر- ۸۵).

پاسخ.

Call by name	Call by reference	Call by Result	Call by Value
900	900	100	100

```
program main;
var k : integer;
procedure XYZ(i, j : integer);
var k : integer;
begin
i = 300 ; k = 2;
if i = j then j := i * k + j;
end;
begin
k = 100;
XYZ(k, k);
write(k);
end.
```

جلسه دهم

مدیریت حافظه :

فازهای مدیریِت حافظه.

منظور از مدیریت حافظه ،مدیریت حافظه اصلی است.

۱-تخصیص اولیه :

یعنی در نخستین بار که برنامه یازیر برنامه ای اجراشد.حافظه موردنیاز آن ازحافظه زمان اجرا تخصیص یابد.

۲-بازیابی فضا های زباله یا garba :

بعد از آزاد سازی بخشی از فضای حافظه که مورد استفاده برنامه ای بود ، باید این قسمت به لیست فضاهای آزاد حافظه اضافه شود.

۳-فشرده سازی و استفاده مجدد.

انواع مدیریت حافظه

۱-مدیریت یصورت استاتیک :

در این روش که ساده ترین روش است در شروع یک برنامه فضائی از حافظه (RAM) به آن اختصاص می یابد.

ایراد: در طول اجرا این فضا ثابت باقی میماند ودر پایان اجرافضا آزاد می گردد.این روش نیاز به نرم افزار مدیریت حافظه ندارد. مثل Fortran

۲-مدیریت بر اساس پشته :

ساده ترین روش مدیریت حافظه در زمان اجرا این روش می باشد . این روش برای فراخوانی تو در تو زیر برنامه ها مفید است.

۳-مدیریت بر اساس Heap یا هرم :

Heap قسمتی از حافظه است میتوانیم توسط برنامه بصورت پویا بلاک هائی از آن را اشغال و بلاک های دیگر را آزاد کنیم . پس از اجرای چندین برنامه بلاک هایی آزاد و بلاک هایی اشغال می شوند .مسئله مهم در این روش مدیریت فضاهای آزاد می باشد . در واقع بعد از اینکه فضاهایی از حافظه آزاد شدچگونه می توان آن را به لیست فضاهای آزاد برگردانیم .

لازم به ذکراست که فضاهای آزاد در روش Heap بصورت یک لیست پیوندی که یک اساره گر ثابت بنام Head به اول آن اشاره میکند نگهداری میشود.

سه تکنیک برای بازیابی فضای اشغال شده وجود دارد :

۱-بازگشت سریع

۲- شمارش تعداد مراجعات

۳-جمع آوری فضاهای از دست رفته.

در کل دو مشکل اساسی در مدیریت حافظه زمان اجرا وجود دارد :

۱-Garbage یا زباله

```
Alloc  Elem  Set(p);
p = q
Int  * p,*q;
⋮
q = malloc(size  Of (int));
p = q;
```

با اجرای این دستور محلی که p به آن اشاره میکرد دیگر آدرسی از آن دردسترس نیست و یک زباله تولید شد.

۲-ارجاعات معلق :

```
int  * p,*q;
p = malloc(size  Of (int))
q = p;
free(p);;
```

با free کردن p در واقع قسمتی از حافظه که p,q به آن اشاره می کردند جزء فضاهای آزاد حساب میشوند و سایر برنامه ها می توانند از آن استفاده کنند در واقع q یک آدرس معلق است .

تکنیک اول:

ساده ترین روش بازیابی استفاده از دستورات صریح است . جهت باز گرداندن حافظه آزاد شده. مثل دستور Dispose در پاسکال وfree در زبان c.حتی سیستم اگر فضایی را آزاد کند باید به صورت صریح اعلام کند.اما این روش همیشه امکان پذیر نیست که یکی از علل ان همان علل قبلی است .

تکنیک دوم :

روش شمارش تعداد مراجعات :

برای هربلوک ازheap یک فضای اضافی بنامRefrence counter درنظر گرفته میشود که این شمارنده تعداد اشاره گرهایی که به یک بلوک اشاره میکنند را درخود نگه میدارد در اثر تخصیص یک بلوک به اشاره گر شماره گر مربوطه یک واحد افزایش یافته و با آزادی آن شمارنده یک واحد کم میشود.

در صورت صفر شدن این شمارنده این فضا جزء فضاهای آزاد محسوب شده و به لیست فضاهای آزاد اضافه میشود .

درست است که دو مشکل قبلی رفع شد ولی هزینه نگهداری و مدیریت این شمارنده سنگین است .

تکنیک سوم :

روش جمع آوری فضاهای از دست رفته :

در این روش اجازه میدهیم تا زباله ها تولید شوند ولی هیچ وقت ارجاع معلق نداشته باشیم ، در این روش به هر بلوک حافظه یک بیت (تگ) اختصاص میدهیم در زمان کمبود حافظه به روش زیر عمل میکنیم : در ابتدا همه این بیت ها on هستند(یک)

۱- عناصری Active هستند که قسمت هایی از برنامه به آن اشاره دارند .بیت تگ(G.C.B) آنها off میشوند (صفرمیشود)

۲-پس از پایان یافتن تنظیمات تگ ها،هر بلوک از حافظه که بیت آن یک است زباله است و باید به لیست فضاهای آزاد اضافه شود

تنها نکته این روش ان است که این روند جمع اوری چه زمانی صورت پذیرد:

دیدگاه اول این است که هر زمان که فضا نیاز شد اجرا شود.

دیدگاه دوم یک دوره زمانی (تناوب زمانی) را پیشنهاد دهد مثلا هر ۵ دقیقه یک بار...