طراحی و پیاده سازی زبانهای برنامه سازی

بر اساس کتاب اصول طراحی و پیاده سازی زبانهای برنامه سازی

ترجمه جعفرنژاد قمی

فصل اول

اصول طراحی زبانها

چرا زبانهای برنامه سازی را مطالعه می کنیم ؟

- برای بهبود توانایی خود در توسعه الگوریتمهای کارآمد
- استفاده بهینه از زبان برنامه نویسی موجود
- می توانید با اصلاحات مفید ساختارهای برنامه نویسی آشنا شوید.
- انتخاب بهترین زبان برنامه سازی
- آموزش زبان جدید ساده می شود.
- طراحی زبان جدید ساده می شود.

تاریخچه مختصری از زبانهای برنامه سازی

توسعه زبانهای اولیه

- زبانهای مبتنی بر اعداد (اواخر دهه 1930 تا اوایل دهه 1940)
- اهداف الگول عبارت بودند از:
  - نشانه های الگول باید به ریاضیات استاندارد نزدیک باشد.
  - الگول باید برای توصیف الگوریتمها مفید باشد.
  - برنامه ها در الگول باید به زبان ماشین ترجمه شوند.
  - الگول نباید به معماری یک ماشین مقید باشد.

تاریخچه مختصری از زبانهای برنامه سازی

توسعه زبانهای اولیه (ادامه)

- زبانهای تجاری (1955)
- زبان هوش مصنوعی (دهه 1950)
- زبانهای سیستم

تاریخچه مختصری از زبانهای برنامه سازی (ادامه)

تکامل معماری نرم افزار

دوران کامپیوترهای بزرگ

- محیط دسته ای
- محیط محاوره ای
- تاثیر بر طراحی زبان

دوران کامپیوتر شخصی

- کامپیوترهای شخصی
- محیطهای سیستم تعبیه شده
- تاثیر بر طراحی زبان

تاریخچه مختصری از زبانهای برنامه سازی (ادامه)

تکامل معماری نرم افزار(ادامه)

دوران شبکه بندی

- محاسبات توزیعی
- اینترنت
- تاثیر بر زبان برنامه سازی

تاریخچه مختصری از زبانهای برنامه سازی (ادامه)

دامنه های کاربرد

کاربردها در دهه 1960

- پردازش تجاری
- محاسبات علمی
- برنامه نویسی سیستم
- کاربردهای هوش مصنوعی

تاریخچه مختصری از زبانهای برنامه سازی (ادامه)

دامنه های کاربرد(ادامه)

کاربردهای قرن 21

- پردازش تجاری
- محاسبات علمی
- برنامه نویسی سیستم
- کاربردهای هوش مصنوعی
- انتشارات
- فرآیند
- کاربردهای جدید (مانند شی گراهاو...)

نقش زبانهای برنامه سازی

اثرات

- قابلیتهای کامپیوتر: تبدیل کامپیوترهای بزرگ ، کند و گرانقیمت که از لامپ خلا استفاده می کردند به ریز کامپیوترها و سوپر کامپیوترها تبدیل شدند.
- موارد کاربرد: زمینه های کاربرد جدید ، طراحی زبانهای جدید ، ارتقاء و بازبینی زبانهای قدیمی
- متدهای برنامه نویسی: یافتن متدهای خوب برای نوشتن برنامه های بزرگ و پیچیده و تغییر در محیط برنامه نویسی
- متدهای پیاده اسزی : انتخاب ویژگیهای نو
- مطالعات تئوری: استفاده از متدهای رسمی ریاضیات
- استانداردسازی: اجازه انتقال برنامه از کامپیوتری به کامپیوتر دیگر

نقش زبانهای برنامه سازی

زبان خوب چگونه است ؟

صفات یک زبان خوب

- وضوح ، سادگی و یکپارچگی
- قابلیت تعامد
- طبیعی بودن برای کاربردها
- پشتیبانی از انتزاع

نقش زبانهای برنامه سازی(ادامه)

زبان خوب چگونه است ؟(ادامه)

صفات یک زبان خوب(ادامه)

- سهولت در بازرسی برنامه
- محیط برنامه نویسی
- قابلیت حمل برنامه
- هزینه استفاده
  - هزینه اجرای برنامه
  - هزینه ترجمه برنامه
  - هزینه نگهداری برنامه

نقش زبانهای برنامه سازی(ادامه)

زبان خوب چگونه است ؟(ادامه)

نحو و معنای زبان

- نحو زبان برنامه سازی ظاهر آن زبان است.
- مشخص شود دستورات ، اعلانها و سایر ساختارهای زبان چگونه نوشته می شوند
- معنای زبان همان مفهومی است که به ساختارهای نحوی زبان داده می شود.

نقش زبانهای برنامه سازی(ادامه)

مدلهای زبان

- زبانهای دستوری: زبانهای مبتنی بر فرمان یا دستورگرا
- زبانهای تابعی : به جای مشاهده تغییر حالت عملکرد برنامه دنبال می شود.
- زبانهای قانونمند: شرایطی را بررسی می کنند و درصورت برقرار بودن آنها فعالیتی را انجام می دهند.
- برنامه نویسی شی گرا: اشیای پیچیده به عنوان بسطی از اشیای ساده ساخته می شوندو خواصی را از اشیای ساده به ارث می برند.

- **زبانهای دستوری یا رویه ای (Imperative)**

برنامه یک سری از دستورات پشت سرهم است که از بالا به پایین اجرا می‌شوند.

Statement 1;
Statement 2;

مانند : C, C++, Fortran, Algol, Cobol, PL/I, Ada, Smalltalk, Pascal

- **زبانهای کاربردی یا تابعی (Functional)**

توابع در کنار هم برنامه را تشکیل می‌دهند و قابلیت فراخوانی تودرتو دارند.

$Function_n(\ldots Function_1\ (data)\ldots)$

مانند : Schema, ML, Lisp

- **مدل مبتنی برقانون (Rule-Base)**

برنامه‌ها به صورت مجموعه ای از قوانین پایه ای تجزیه مساله ، می‌باشند.

ساختمان این زبان مشابه ساختار if است که در صورت رخداد شرایطی ،قانونی اجرا می‌شود.

Enabling condition1 $\rightarrow$ action1 = Action1 if enabling condition1

مانند : Prolog

- **زبانهای شیء گرا (Object Oriented)**

برنامه از تعدادی ماژول تشکیل شده است که هر ماژول مشابه یک برنامه در زبان C می‌باشد. مبحث اصلی در این نوع زبان پشتیبانی از کلاس است.

مانند : C++, Java, Visual

نقش زبانهای برنامه سازی(ادامه)

استاندارد سازی زبان

روش:

- برای پی بردن به معنای دستورات به مستندات زبان مراجعه شود.
- برنامه را در کامپیوتر تایپ . اجرا کنید
- به استاندارد زبان مراجعه شود.

استاندارد خصوصی(توسط مالک زبان)

استاندارد عمومی (توسط شرکتهای استانداردISO)

مسائل مهم استفاده موثر از استاندارد:

- زمان شناسی(تسریع در استاندارد شدن به منظور عدم وجود نسخه های ناسازگار مانند فرترن دیر استاندارد شد و آدا زود)
- اطاعت و پیروی(اطاعت زبان از استاندارد)
- کهنگی(عدم تغییر استاندارد)

نقش زبانهای برنامه سازی(ادامه)

بین المللی شدن برنامه نویسی

- ترتیب تلفیق: کاراکترها به چه ترتیبی باید ظاهر شوند ؟
- ترتیب: موقعیت کاراکترهای غیر رومی
- حالت کاراکترها: حروف کوچک و بزرگ در زبانهایی مثل ژاپنی ، عربی و یهودی
- جهت پیمایش: اغلب زبانها از چپ به راست خوانده می شوند.
- فرمت تاریخ در یک کشور خاص
- فرمت زمان در یک کشور خاص
- مناطق زمانی
- سیستمهای حروفی
- علامت پول

محیط های برنامه نویسی

تاثیر بر طراحی زبان

ویژگیها

- کامپایل کردن مجزا مانند مشخه ، اعلان نوع داده ، تعریف نوع داده
- تست و اشکال زدایی مانند : ویژگیهای ردیابی اجرا ، نقاط کنترلی ، ادعا

محیط های برنامه نویسی(ادامه)

محیط های کاری

- خدماتی مثل ذخیره داده ها ، رابط گرافیکی کاربر ، امنیت و خدمات ارتباطی را فراهم می کند.

محیط های برنامه نویسی(ادامه)

زبانهای کنترل کار و فرآیند

- مفهوم کنترل کار به چارچوبهای محیط برمی گردد.
- کاربر کنترل مستقیم بر روی مراحل مختلف برنامه دارد.

فصل دوم

اثرات معماری ماشین

عملکرد کامپیوتر

کامپیوتر مجموعه ای از الگوریتمها و ساختمان داده ها است که قابلیت ذخیره و اجرای برنامه ها را دارد.

- هر کامپیوتر از 6 جزء تشکیل شده است:
  - داده ها
  - اعمال اولیه
  - کنترل ترتیب
  - دستیابی به داده ها
  - مدیریت حافظه
  - محیط عملیاتی

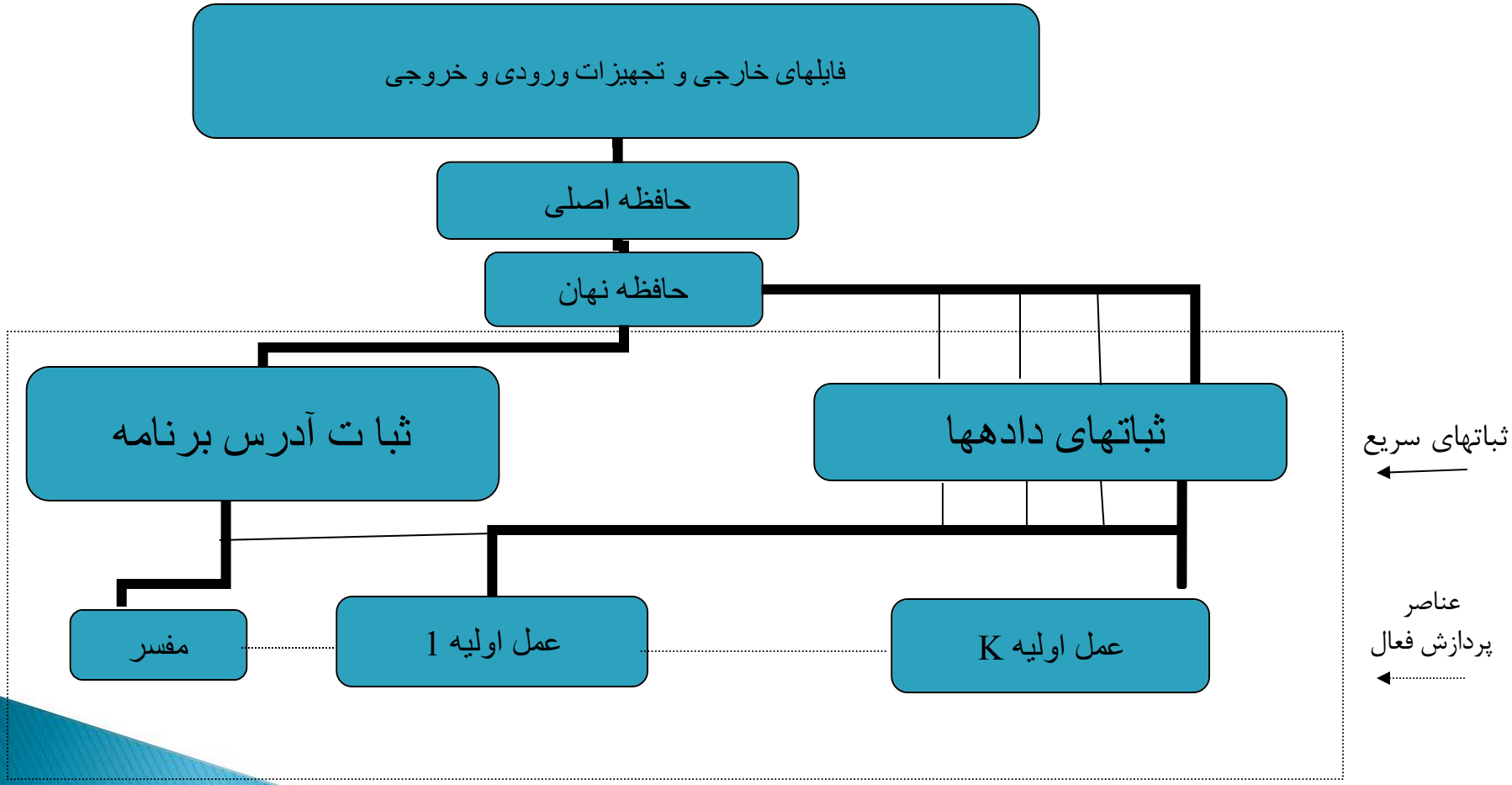
عملکرد کامپیوتر(ادامه)

سخت افزار کامپیوتر

داده ها : حافظه اصلی ، ثباتهای سریع و فایلهای خارجی

- حافظه اصلی به صورت دنباله ای از بیتهای خطی سازمان دهی می شود که از کلمات با طول ثابت تشکیل می گردد.
- طول ثباتهای سریع به اندازه طول کلمات است و طوری تقسیم بندی می شود که هر قسمت آن قابل دستیابی باشد.
- حافظه سریع نهان معمولاً بین حافظه اصلی و ثبات ها قرار می گیرد و مکانیزمی برای دسترسی سریع به داده های موجود در حافظه است
- فایلهای خارجی که بر روی دیسک مغناطیسی ، نوار مغناطیسی یا CD ذخیره می شوند.

سازمان یک کامپیوتر معمولی



عملکرد کامپیوتر(ادامه)

سخت افزار کامپیوتر (ادامه)

- اعمال : کامپیوتر باید مجموعه ای از اعمال اولیه توکار داشته باشد که متناظر با کدهای عملیاتی که هستند به صورت دستورات زبان ماشین می باشند.
- کنترل ترتیب: در حین اجرای برنامه دستور بعدی که باید اجرا شود توسط محتویات ثبات آدرس برنامه مشخص می گردد. این ثبات حاوی آدرس دستور بعدی است.

عملکرد کامپیوتر(ادامه)

سخت افزار کامپیوتر (ادامه)

- دستیابی به داده ها : علاوه بر کد عملیاتی هر دستور ماشین باید عملوندهایی را مشخص کند که آن عمل از آن استفاده می کند. عملوند ممکن است در حافظه اصلی یا در ثبا ت باشد.
- مدیریت حافظه: تمام منابع کامپیوتر (مثل حافظه ، پردازنده مرکزی ، دستگاههای حافظه خارجی) تا آنجایی که ممکن است فعال باشند.
- محیط عملیاتی : متشکل از مجموعه ای از حافظه جانبی و دستگاههای ورودی و خروجی است. مثل حافظه های سریع ، حافظه هایی با سرعت متوسط ، حافظه های کند و دستگاههای ورودی و خروجی

عملکرد کامپیوتر (ادامه)

کامپیوترهای میان افزار

- کامپیوتر میان افزار توسط ریز برنامه ای 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.

عملکرد کامپیوتر (ادامه)

مفسرها و معماریهای مجازی

- ترجمه (کامپایل کردن) : مفسر می تواند طوری طراحی شود که برنامه ای به یک زبان سطح بالا را به برنامه ای در زبان ماشین ترجمه کند.
  - مفسر هر پردازنده زبانی است که برنامه ای را به یک زبان منبع (که ممکن است سطح بالا یا پایین باشد) به عنوان ورودی گرفته به برنامه ای در زبان مقصد تبدیل می کند که از نظر کارایی با هم یکسان هستند.
    - اسمبلر
    - کامپایلر
    - بارکننده یا ویراستار پیوند
    - پیش پردازنده یا پردازنده ماکرو

عملکرد کامپیوتر (ادامه)

مفسرها و معماریهای مجازی (ادامه)

- شبیه سازی نرم افزاری (تفسیر نرم افزاری): به جای ترجمه برنامه های سطح بالا به برنامه های زبان ماشین معادل می توانیم از شبیه سازی استفاده کنیم که از طریق آن برنامه بر روی کامپیوتر میزبان اجرا می شود.

عملکرد کامپیوتر (ادامه)

مفسرها و معماریهای مجازی (ادامه)

- زبانها به دو دسته هستند:
- زبان های کامپایلری : C,C++ ، فرترن ، پساکال و ادا . برنامه های آن قبل از شروع اجرای برنامه به زبان ماشین کامپیوتر واقعی ترجمه می شوند به طوریکه شبیه سازی به مجموعه ای از روالهای پشتیبانی زمان اجرا محدود می شود که اعمال اولیه موجود در زبان منبع را شبیه سازی می کند که شباهت زیادی به زبان ماشین ندارد.
- زبان های مفسری: لیسپ ، ام ال ، پرل ، پست اسکریپت ، پرولوپ و اسمالتاک معمولاً با مفسر نرم افزاری پیاده سازی می شود.

کامپیوترهای مجازی و زمانهای انقیاد (Binding)

روشهای ساخت کامپیوتر:

- از طریق سخت افزار
- از طریق نرم افزار
- از طریق ماشین مجازی
- از طریق ترکیبی

کامپیوترهای مجازی و زمانهای انقیاد (ادامه)

کامپیوترهای مجازی و پیاده سازی های زبان

- بعنوان مثال اگر کامپیوتر مجازی ، حاوی عمل جمع صحیح و عمل جذرگیری باشد ، پیاده ساز ممکن است جمع صحیح را به وسیله سخت افزار و جذرگیری را به وسیله نرم افزار انجام دهد.

کامپیوترهای مجازی و زمانهای انقیاد (ادامه)

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 .

کامپیوترهای مجازی و زمانهای انقیاد (ادامه)

سلسله مراتب ماشینهای مجازی

کامپیوتر برنامه کاربردی وب
کامپیوتر مجازی وب
کامپیوتر مجازی C
کامپیوتر مجازی سیستم عامل
کامپیوتر مجازی میان افزار
کامپیوتر سخت افزاری واقعی

کامپیوترهای مجازی و زمانهای انقیاد

انقیاد و زمان انقیاد

- زمان اجرا
  - در ورود به زیر برنامه یا بلوک
  - در نقطه خاصی از اجرای برنامه
- زمان ترجمه (زمان کامپایل)
  - انقیاد توسط برنامه نویس انتخاب می شود
  - انقیاد توسط مترجم انجام می شود
  - انقیادهایی که توسط بارکننده صورت می گیرد.
- زمان پیاده سازی زبان
- زمان تعریف زبان

کامپیوترهای مجازی و زمانهای انقیاد (ادامه)

انقیاد و زمان انقیاد (ادامه)

- در برنامه ای که به زبان L نوشته می شود انقیادها و زمانهای انقیاد عناصر زیر بحث می شود:
  - مجموعه ای از انواع ممکن برای متغیر X
  - نوع متغیر
  - مجموعه ای از مقادیر ممکن برای X
  - مقدار متغیر X
  - نمایش مقدار ثابت 10
  - خواص عملگر +

کامپیوترهای مجازی و زمانهای انقیاد (ادامه)

انقیاد و زمان انقیاد (ادامه)

- زبانی مثل فرترن که در آن انقیاد در زمان ترجمه انجام می شود زبانهایی با ***انقیاد زودرس*** و زبانی با ***انقیاد دیررس*** مثل ام ال اغلب انقیادها را در زمان اجرا انجام می دهد.

فصل سوم

انواع داده اولیه

خواص انواع و اشیاء

- هر برنامه صرفنظر از نوع زبان مجموعه ای از عملیات است که باید به ترتیب خاصی بر روی داده ها اجرا شوند.
- تفاوتهای بین زبانها ناشی از انواع دادهها ، عملیات موجود و مکانیزم کنترل ترتیب اجرای عملیات بر روی داده ها است.

خواص انواع و اشیاء (ادامه)

اشیای داده ، متغیرها و ثوابت

- از اصطلاح شی داده برای گروهبندی زمان اجرای یک یا چند قطعه از داده ها در کامپیوتر مجازی استفاده می کنیم.
- بعضی اشیا در حین اجرای برنامه توسط برنامه نویس تعریف شده اند.
- بعضی از اشیای داده توسط سیستم تعریف می شوند.
- اجزای تعریف شده توسط سیستم در حین اجرای برنامه در صورت نیاز به طور خودکار ایجاد می شوند.

خواص انواع و اشیاء (ادامه)

اشیای داده ، متغیرها و ثوابت (ادامه)

- شی داده ظرفی برای مقادیر داده است.
- مقدار داده ممکن است یک عدد ، کاراکتر یا اشاره گری به شی داده دیگر باشد.
- اگر شی داده حاوی مقداری باشد که همیشه به عنوان یک واحد دستکاری شود آن را طی داده اولیه گویند.
- اگر شی داده مجموعه ای از سایر اشیای داده باشد ساختمان نامیده می شود.

خواص انواع و اشیاء (ادامه)

اشیای داده ، متغیرها و ثوابت (ادامه)

- شی داده ظرفی برای مقادیر داده است.
  - نوع
  - محل
  - مقدار
  - نام
  - اجزاء

خواص انواع و اشیاء (ادامه)

اشیای داده ، متغیرها و ثوابت (ادامه)

متغیرها و ثوابت

- شی داده ای که توسط برنامه نویس تعریف و نامگذاری می شود متغیر نام دارد.
- ثابت یک شی داده با نام است که مقداری به آن نسبت داده می شود.
- یک لیترال ثابتی است که نامش همان نمایش مقدارش است
- ثابت تعریف شده توسط برنامه نویس ثابتی است که نامش در تعریف شی داده توسط برنامه نویس انتخاب می شود.

خواص انواع و اشیاء (ادامه)

اشیای داده ، متغیرها و ثوابت (ادامه)

ماندگاری

- اغلب برنامه های امروزی هنوز براساس مدل پردازش دسته ای نوشته می شوند.

یعنی برنامه نویس دنباله از رویدادهای زیر را فرض می کند:

- برنامه به حافظه بار می شود.
- داده خارجی مناسب برای برنامه مهیایند.
- داده ورودی موردنظر خوانده شده در متغیرهایی در حافظه قرار می گیرند.
- برنامه خاتمه می یابد.

خواص انواع و اشیاء (ادامه)

انواع داده

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

خواص انواع و اشیاء (ادامه)

انواع داده (ادامه)

- عناصر اصلی مشخصات یک نوع داده:
  - صفاتی
  - مقادیری
  - عملیاتی

خواص انواع و اشیاء (ادامه)

انواع داده (ادامه)

مشخصات انواع داده اولیه:

- صفات
- مقادیر
- عملیات

خواص انواع و اشیاء (ادامه)

انواع داده (ادامه)

چهارعامل موجب می شوند تا تعریف عملیات زبان برنامه سازی دشوار شود:

- عملیاتی که برای ورودیهای خاصی تعریف نشده اند.
- آرگومانهای ضمنی
- اثرات جانبی (نتایج ضمنی)
- خود اصلاحی

اگر نوعی به عنوان بخشی از نوع بزرگتر باشد آن را **زیر نوع** و نوع بزرگتر را **ابرنوع** می گویند.

خواص انواع و اشیاء (ادامه)

انواع داده (ادامه)

پیاده سازی انواع داده اولیه

- 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:

- برای کارایی بعضی از زبانها طوری طراحی شدند که صفات داده ها توسط کامپایلر تعیین شوند.
- صفات شی داده ممکن است در زمان اجرا دریک توصیف گر و به عنوان بخشی از شی داده ذخیره شود.

خواص انواع و اشیاء (ادامه)

انواع داده (ادامه)

پیاده سازی انواع داده اولیه (ادامه)

- پیاده سازی عملیات: هر عملیاتی که برای نوعی از اشیای داده تعریف شد ممکن است به یکی از سه روش زیر پیاده سازی شود:
  - به صورت عملیات سخت افزاری
  - به صورت زیر برنامه رویه یا تابع
  - به صورت دستوراتی در داخل برنامه نوشته شوند.

خواص انواع و اشیاء (ادامه)

اعلانها

- دستوری از برنامه است که نام و نوع اشیای داده را که در حین اجرای برنامه مورد نیاز هستند مشخص می کند.
- اشیایی که در طول عمرشان به اسامی مانند A,B مقید می شوند اعلان صریح گویند.
- در بعضی از زبانها اعلان ضمنی یا اعلان پیش فرض وجود دارد.

خواص انواع و اشیاء (ادامه)

اعلانها

اعلان عملیات

اعلانها می توانند اطلاعاتی راجع به عملیات را برای مترجم زبان فراهم کنند.

- اهداف اعلان:
  - انتخاب نمایش حافظه
  - مدریت حافظه
  - عملیات چندریختی
  - کنترل نوع

خواص انواع و اشیاء (ادامه)

کنترل نوع و تبدیل نوع

- منظور از کنترل نوع این است که هر عملیاتی که در برنامه انجام می گیرد تعداد و نوع آرگومانهای آن درست باشد.
- کنترل نوع ممکن است در زمان اجرا صورت گیرد(کنترل نوع پویا)
- کنترل نوع ممکن است در زمان ترجمه صورت گیرد(کنترل نوع ایستا)

خواص انواع و اشیاء (ادامه)

کنترل نوع و تبدیل نوع (ادامه)

معایب کنترل نوع پویا:

- اشکالزدایی برنامه و حذف تمام خطاهای نوع آرگومان مشکل است.
- در کنترل نوع پویا لازم است اطلاعات مربوط به نوع در زمان اجرای برنامه نگهداری شوند.
- کنترل نوع پویا باید به صورت نرم افزاری پیاده سازی شود.

خواص انواع و اشیاء (ادامه)

کنترل نوع و تبدیل نوع (ادامه)

برای برطرف کردن معایب کنترل نوع ایستا دو روش:

- کنترل نوع پویا
- عملیات کنترل نشوند.

خواص انواع و اشیاء (ادامه)

کنترل نوع و تبدیل نوع (ادامه)

تبدیل نوع و تبدیل نوع ضمنی

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

خواص انواع و اشیاء (ادامه)

کنترل نوع و تبدیل نوع (ادامه)

تبدیل نوع و تبدیل نوع ضمنی (ادامه)

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

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

خواص انواع و اشیاء (ادامه)

انتساب و مقدار دهی اولیه

- انتساب عملیات اصلی برای تغییر انقیاد یک مقدار به یک شی داده است.
- این تغییر اثیر جنبی عملیات است.

خواص انواع و اشیاء (ادامه)

انتساب و مقدار دهی اولیه (ادامه)

تعریف عملیات انتساب به صورت زیر :

- مقدار چپ اولین عبارت عملوند را محاسبه کن
- مقدار راست دومین عبارت عملوند را محاسبه کن
- مقدار راست محاسبه شده را به شی داده مقدار چپ محاسبه شده نسبت بده
- مقدار راست محاسبه شده را به عنوان نتیجه عملیات برگردان

خواص انواع و اشیاء (ادامه)

انتساب و مقدار دهی اولیه (ادامه)

- تساوی و هم ارزی: انتساب دادن مقدار یا عبارت به متغیر
- مقدار دهی اولیه: یک شی داده است که ایجاد شده ولی هنوز مقداری به آن داده نشده است.

انواع داده اسکالر

- داده مرکب را در نظر می گیریم که درآن یک شی می تواند چندین صفت داده باشد.
- اشیای اسکالر از معماری سخت افزار کامپیوتر پیروی می کنند.

انواع داده اسکالر(ادامه)

انواع داده عددی

انواع صحیح :

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

عملیات بر روی اشیای داده صحیح شامل موارد زیر است:

- عملیات محاسباتی
- عملیات رابطه ای
- انتساب
- عملیات بیتی
- پیاده سازی

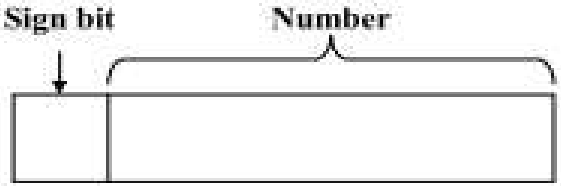
اعداد صحیح (ادامه)

➢ پیاده سازی

نوع داده صحیح توسط نمایش حافظه سخت افزار و مجموعه‌ای از عملیات محاسباتی و رابطه ای بر روی مقادیر صحیح پیاده سازی می‌شوند. ۳ نوع نمایش حافظه ای برای نوع داده صحیح وجود دارد:

الف- بدون توصیفگر:

این نمایش حافظه، توصیفگر زمان اجرا ندارد و فقط مقدار در آن ذخیره می‌شود. این نمایش حافظه در زبان‌هایی استفاده می‌شود که زبان اعلان‌ها و کنترل نوع ایستا را برای مقادیر صحیح فراهم می‌کند مانند C و فرترن

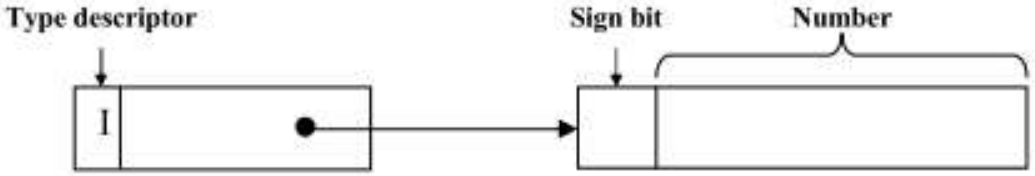


اعداد صحیح (ادامه)

- پیاده سازی

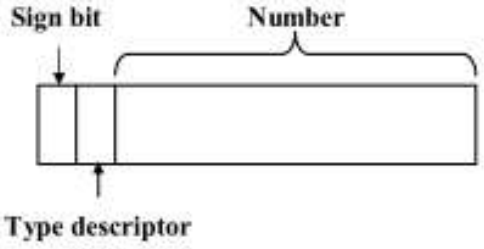
ب- توصیفگر و مقدار در دو کلمه مجزا:

این نمایش حافظه، توصیفگر زمان اجرا دارد و توصیفگر در محل دیگری از حافظه ذخیره شده است که اشاره‌گری به آن اشاره می‌کند. این نمایش حافظه در لیسپ استفاده می‌شود. عیب آن این است که حافظه لازم برای شی داده صحیح دو برابر می‌شود و مزیت آن این است که عملیات روی آن به صورت سخت افزاری قابل پیاده سازی است. استفاده از نمایش سخت افزاری باعث افزایش سرعت عملیات می شود.



ج- توصیفگرو مقدار در یک کلمه:

توصیفگر نوع و مقدار در یک کلمه ذخیره می‌شود لذا در حافظه صرفه جویی می‌شود ولی برای استفاده عملیات سخت افزار باید مقدار را از توصیفگر توسط دستورات شیفت از یکدیگر جدا کرد لذا سرعت عمل کمتر است.(از روش ب بیشتر استفاده می‌شود.)



انواع داده اسکالر(ادامه)

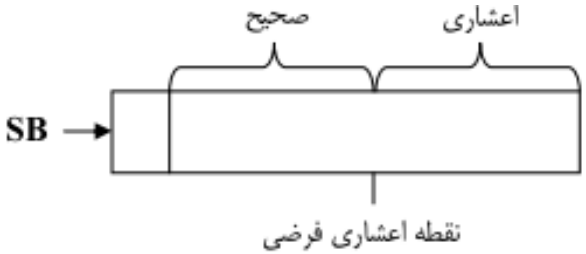
انواع داده عددی (ادامه)

اعداد حقیقی ممیز شناور

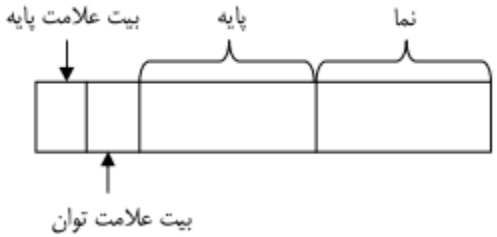
- مشخصات : معمولاً با صفت نوع داده مثل real در فرترن یا float در C مشخص می شود.
- پیاده سازی: نمایشهای حافظه برای انواع آن معمولاً به سخت افزار بستگی دارد که در آن ممیز حافظه به دو بخش مانتیس (پایه) و توان تقسیم می شود.

پیاده سازی

1- ممیز ثابت (fixed Point)



2-ممیز شناور (Floating Point)

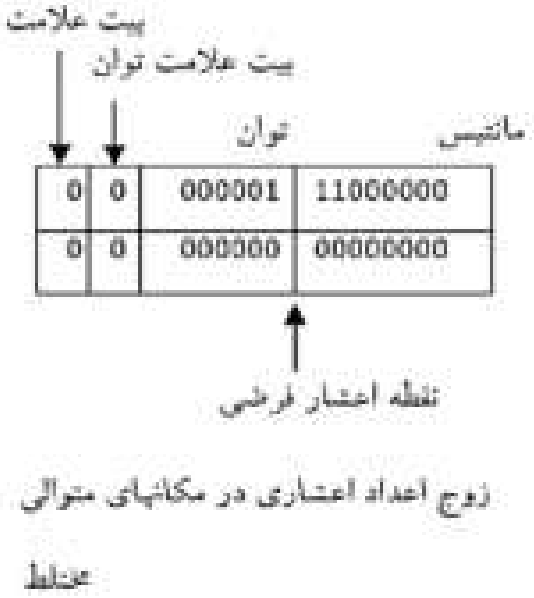


انواع داده اسکالر(ادامه)

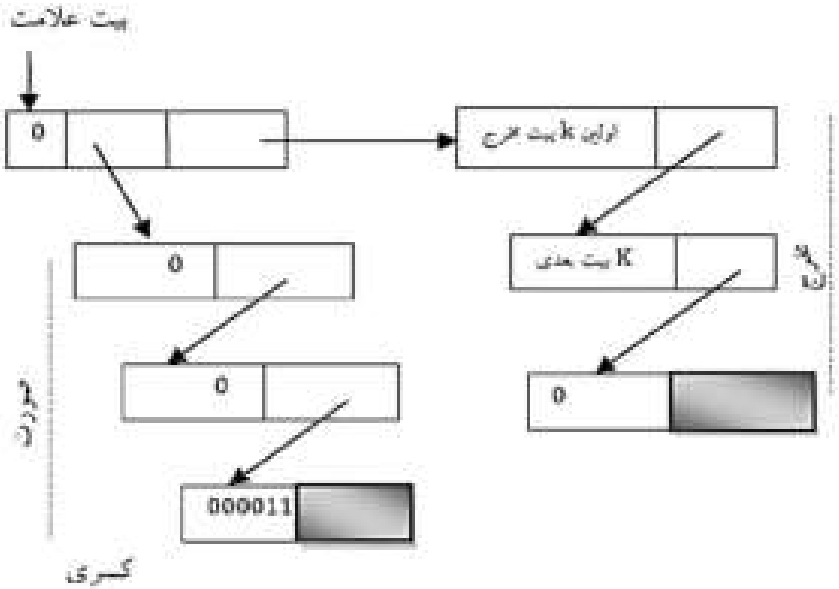
انواع داده عددی (ادامه)

سایر انواع داده عددی

- اعداد مختلط: 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.



انواع داده اسکالر(ادامه)

نوع شمارشی

- مشخصات: لیست مرتبی از مقادیر مجزا است. برنامه نویس اسامی لیترالهایی را که باید برای مقادیر مورد استفده قرار گیرند و همچنین ترتیب آنها را با استفاده از اعلانی مانند زیر در C مشخص می کند.
- پیاده سازی: نمایش حافظه برای شی داده ای از نوع شمارشی ساده است.

انواع داده اسکالر(ادامه)

نوع بولی

- مشخصات: متشکل از اشیای داده ای است که یکی از دو مقدار TRUE یا FALSE را می پذیرد.
- پیاده سازی: نمایش حافظه برای شی داده بولی یک بیت از حافظه است .مقادیر true و false به دو روش در این واحد حافظه نمایش داده می شوند:
  - بیت خاصی برای این مقادیر استفاده می شود.
  - مقدار صفر در کل واحد حافظه نشاندهنده false و مقدار غیرصفر نشاندهنده true است.

انواع داده اسکالر(ادامه)

کاراکترها

- مشخصات : نوع داده کاراکتری اشیای داده را به وجود می آورد که مقدار آنها یک کاراکتر است.
- پیاده سازی: مقادیر داده های کاراکتری همیشه توسط سخت افزار و سیستم عامل پشتیبانی می شوند.

انواع داده مرکب

رشته های کاراکتری

مشخصات و نحو

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

- طول ثابت
- طول متغیر با حد بالا
- طول نامحدود

انواع داده مرکب(ادامه)

رشته های کاراکتری (ادامه)

مشخصات و نحو (ادامه)

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

- الحاق
- عملیات رابطه ای در رشته ها
- انتخاب زیر رشته با استفاده از اندیس
- فرمت بندی ورودی – خروجی
- انتخاب زیر رشته با تطابق الگو
- رشته های پویا

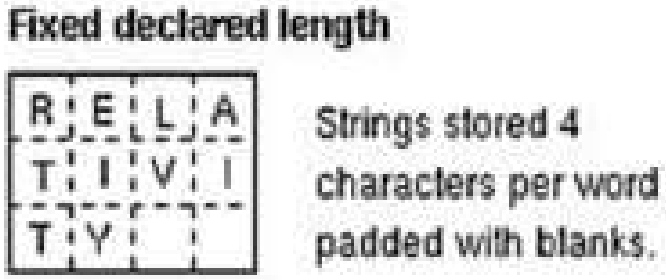
.

انواع داده مرکب(ادامه)

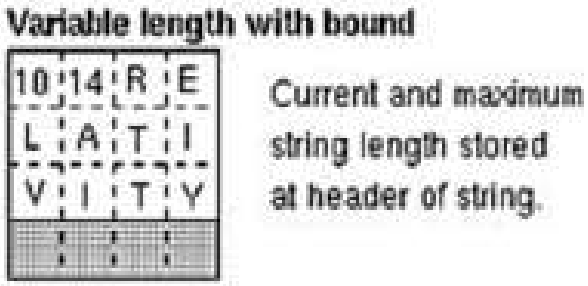
رشته های کاراکتری (ادامه)

پیاده سازی

- رشته با طول ثابت : هر 4 کاراکتر در یک کلمه ذخیره می شود و بقیه طول رشته با فضای خالی پر می شود.



- 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.

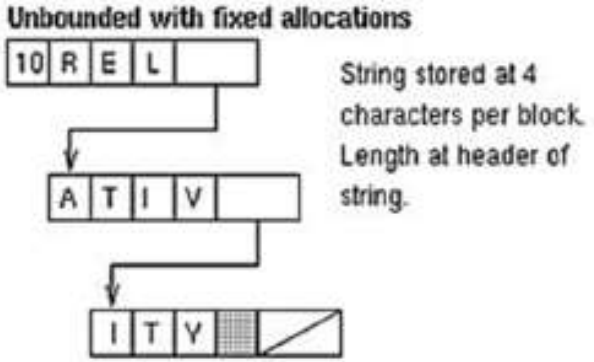


انواع داده مرکب(ادامه)

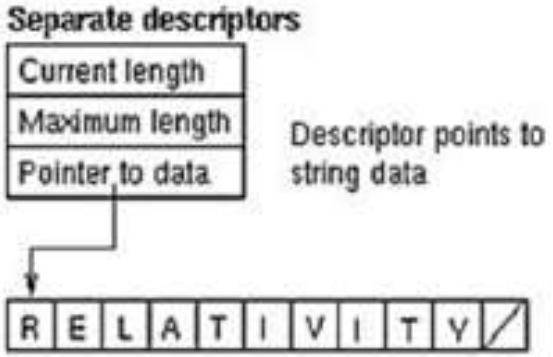
رشته های کاراکتری (ادامه)

پیاده سازی

- رشته های با طول نامحدود با تخصیص ثابت : 4 کاراکتر در هر بلوک ذخیره می شود و طول در ابتدای رشته قرار می گیرد.



- رشته های با طول نامحدود با تخصیص های متغیر: رشته به صورت ارایه پیوسته کاراکترها ذخیره می شود و انتهای هر رشته با null یا تهی مشخص می شود.



انواع داده مرکب(ادامه)

اشاره گرها و اشیای داده برنامه نویس

زبان باید ویژگیهای زیر را داشته باشد:

- نوع داده اولیه اشاره گر
- عمل ایجاد کردن
- عملیات دستیابی به محتویات

انواع داده مرکب(ادامه)

اشاره گرها و اشیای داده برنامه نویس (ادامه)

مشخصات: نوع داده اشاره گر دسته از اشیای داده را تعریف می کند که مقادیر آنها آدرسهای اشیای دیگری اند

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

انواع داده مرکب(ادامه)

اشاره گرها و اشیای داده برنامه نویس (ادامه)

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: مقدار اشاره گر ممکن است آدرس واقعی بلوک حافظه مربوط به شی داده باشد.
- آدرس نسبی: مقدار اشاره گر ممکن است آفستی از آدرس پایه بلوک حافظه هرم باشد که شی داده در آن ایجاد شده است.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی

- فایل ساختمان داده ای با دو ویژگی است:
- بر روی حافظه ثانویه مثل دیسک یا نوار تشکیل می شود و ممکن است بسیار بزرگتر از سایر ساختمان داده ها باشد.
- طول عمر آن می تواند بسیار زیاد باشد.

انواع داده مرکب(ادامه)

فایلها و ورودی – خروجی (ادامه)

- متداول ترین فایلها ، فایلهای ترتیبی اند.
- فایلهای دستیابی مستقیم
- فایلهای ترتیبی شاخص دار
- ورودی – خروجی محاوره ای
- فایلهای متنی

انواع داده مرکب(ادامه)

فایلها و ورودی – خروجی (ادامه)

فایلهای ترتیبی

- ساختمان داده ای مرکب از دنباله خطی از عناصر همنوع است .
- طول آن متغیر است و حد بالایی ندارد.
- برای ورودی – خروجی داده ها معمولاً به صورت کاراکتری اند.
- فایل می تواند در حالت خواندن یا در حالت نوشتن دستیابی شود.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

فایلهای ترتیبی (ادامه)

- مشخصات: عملیات اصلی بر روی فایلهای ترتیبی :
  - بازکردن
  - خواندن
  - نوشتن
  - تست انتنهای فایل
  - بستن

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

فایلهای ترتیبی (ادامه)

- پیاده سازی: سیستم عامل مسئول پیاده سازی فایلها است.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

فایلهای متنی

- فایلی از کاراکترها است .
- شکل اولیه فایل مربوط به ورودی – خروجی در اغلب زبانها است.
- فایلهای متنی را مستقیماً از طریق صفحه کلید می توان ایجاد و چاپ کرد.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

ورودی – خروجی محاوره ای

- اصلاح چندین جنبه از دیدگاه معمولی فایلهای ترتیبی که در بالا مطرح شدند :
- فایل همزمان باید در حالت خوان و نوشتن باشد.
- بافر کردن داده در ورودی و خروجی محدود می شود.
- اشاره گر موقعیت فایل و تست انتهای فایل ارزش چندانی ندارند.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

فایلهای دستیابی مستقیم

- در فایل ترتیبی عناصر به ترتیبی که در فایل قرار دارند بازیابی می شوند .
- دستیابی تصادفی به عناصر غیر ممکن است.
- می توان به هر عنصر به طور تصادفی دست یافت.
- به صورت مجموعه ای از عناصر نامرتب سازماندهی می شود.

انواع داده مرکب(ادامه)

فایلها و ورودی - خروجی (ادامه)

فایل ترتیبی شاخص دار

- این سازمان فایل مصالحه ای را بین سازمانهای ترتیبی محض و دستیابی مستقیم محض به وجود می آورد.
- نیازمند شاخصی از مقادیر کلید است
- اما ورودیهای شاخص باید بر حسب مقادیر کلید مرتب باشند.

فصل چهارم

بسته بندی

مقدمه

تمام فعالیتهای طراحی را می توان به عنوان طراحی مشخصات نوع داده انتزاعی در نظر گرفت:

- طراحی صفات
- عملیات موردنیاز

چهار روش انواع داده جدید و عملیاتی بر روی آنها:

- ساختمان داده
- زیربرنامه ها
- اعلان نوع
- وراثت

ساختمان داده ها

اشیای داده ساختاری و انواع داده

- شی داده ای که مرکب از اشیای داده دیگری است ساختمان داده نام دارد.
- بسیاری از مفاهیم و اصول مربوط به ساختمان داده ها در زبانهای برنامه سازی مشابه اشیای داده اولیه است

ساختمان داده ها (ادامه)

مشخصات انواع ساختمان داده

صفات اصلی مشخص کننده ساختمان داده:

- تعداد عناصر
- نوع هر عنصر
- اسامی برای انتخاب عناصر
- حداکثر تعداد عناصر
- سازمان عناصر

ساختمان داده ها (ادامه)

مشخصات انواع ساختمان داده (ادامه)

عملیات در ساختمان داده ها

دسته های دیگری از عملیات از اهمیت ویژه ای برخوردارند:

- عملیات انتخاب عناصر
- عملیات بر روی کل ساختمان
- درج و حذف عناصر
- ایجاد و حذف ساختمان داده ها

ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها

دو موضوع دیگر که انتخاب نمایش حافظه را تحت تاثیر قرار می دهد:

- انتخاب کارآمد عنصر از ساختمان
- مدیرحافظه کارآمد برای پیاده سازی زبان

ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها (ادامه)

نمایش های حافظه

شامل:

- حافظه ای برای عناصر ساختمان داده
- توصیفگر اختیاری آنها

دو نمایش اصلی:

- نمایش ترتیبی
- نمایش پیوندی

ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها (ادامه)

پیاده سازی عملیات ساختمان داده ها

- انتخاب عناصر ساختمان داده مهمترین مسئله در پیاده سازی آن است
- کارآمد بودن عملیات انتخاب تصادفی و انتخاب ترتیبی ضروری است.

ساختمان داده ها (ادامه)

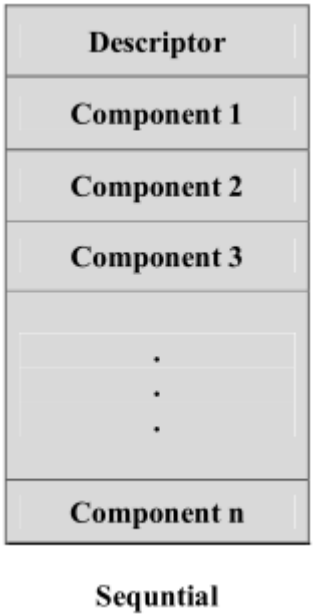
پیاده سازی انواع ساختمان داده ها (ادامه)

پیاده سازی عملیات ساختمان داده ها (ادامه)

- نمایش ترتیبی: در انتخاب تصادفی یک آدرس پایه – آفست باید با استفاده از فرمول دستیابی محاسبه شود.

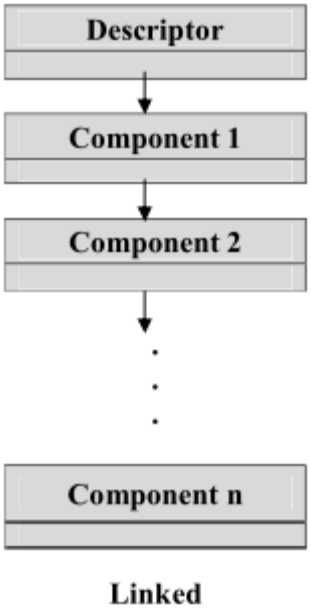
در ساختار همگن انتخاب دنباله ای از عناصر می تواند به صورت زیر انجام شود:

- برای دستیابی به اولین عنصر دنباله از محاسبه آدرس پایه – آفست استفاده کنید.
- برای دستیابی به عنصر بعدی دنباله اندازه عنصر فعلی را به موقعیت عنصر فعلی اضافه کنید.



Sequntial

ساختمان داده ها (ادامه)



پیاده سازی انواع ساختمان داده ها (ادامه)

پیاده سازی عملیات ساختمان داده ها (ادامه)

- نمایش پیوندی: برای انتخاب باید زنجیره ای از اشاره گرها را از اولین بلوک موجود در ساختار تا عنصر موردنظر دنبال کرد.
- برای انتخاب دنباله ای از مولفه ها باید اولین عنصر را انتخاب و سپس اشاره گر پیوندی را تا عنصر مورد نظر دنبال کرد.

ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها (ادامه)

مدیریت حافظه و ساختمان داده ها

- طول عمر هر شی داده با انقیاد شی به محلی از حافظه شروع می شود.

به علت تاثیر متقابل بین طول عمر شی داده و مسیرهای دستیابی دو مسئله مهم در مدیریت حافظه به وجود می آید:

- زباله Garbage data
- ارجاعهای معلقDangling reference

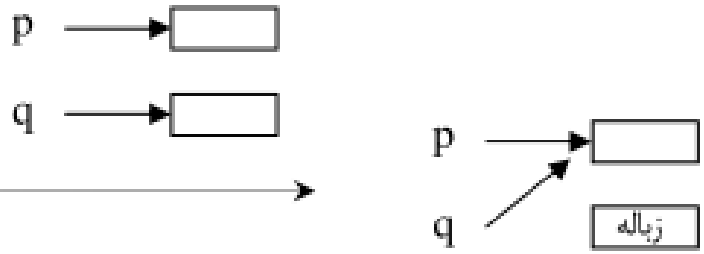
ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها (ادامه)

مدیریت حافظه و ساختمان داده ها (ادامه)

- زباله Garbage data : هنگامی که یک مسیر دستیابی به یک شی داده از بین برود ولی خود شی داده وجود داشته باشد ، شی داده را زباله گوییم زیرا دیگر قابل استفاده نیست ولی انقیاد آن به محل حافظه اش از بین نرفته است.
- مثال:

```
Int *p,*q;
p = malloc(10);
q = malloc(10);
q = p;
```



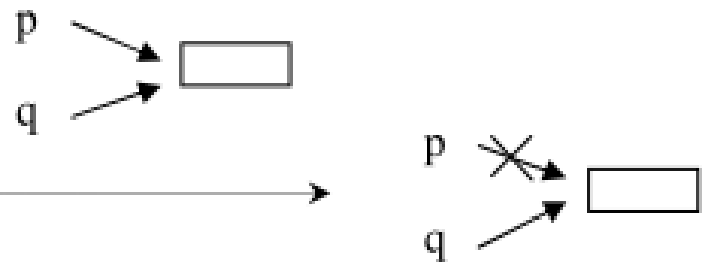
ساختمان داده ها (ادامه)

پیاده سازی انواع ساختمان داده ها (ادامه)

مدیریت حافظه و ساختمان داده ها (ادامه)

- ارجاعهای معلق Dangling reference: اگر طول عمر شی داده تمام شده باشد ولی هنوز یک مسیر دستیابی به آن را در اختیار داشته باشیم ارجاع معلق رخ داده است. در حقیقت ارجاع معلق یا سرگردان ، یک مسیر دستیابی است که پس از اینکه طول عمر شی داده خاتمه یافت ، هنوز وجود داشته باشد.

```
Int *p,*q;
p = malloc(10);
q = p;
free(p);
```



ساختمان داده ها (ادامه)

اعلانها و کنترل نوع برای ساختمان داده ها

- مثل انواع داده اولیه است
- ساختمان داده ها پیچیده ترند زیرا صفات بیشتری باید مشخص شوند.
- دو مسئله در این مورد وجود دارد:
  - وجود مولفه انتخابی

Int A[1..10] A[13] غیر قانونی

  - نوع عنصر انتخابی

Int A[1..3][1..3]
A[1][2]=3.2

ساختمان داده ها (ادامه)

بردارها و آرایه ها

- متداولترین ساختمان داده ها در زبانهای برنامه سازی اند.
- بردار ساختمان مرکب از تعداد ثابتی از عناصر همنوع است که به صورت یک دنباله خطی سازمان دهی شده است.
- برای دستیابی به عناصر بردار از اندیس استفاده می شود.

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

بردارها:

- تعداد عناصر: معمولا توسط بازه که برای اندیس مشخص شده ، تعیین می گردد.
- نوع هر عنصر: تمام عناصر بردار ار یک نوع هستند.
- اندیس برای انتخاب هر عنصر

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

عملیات بر روی بردارها:

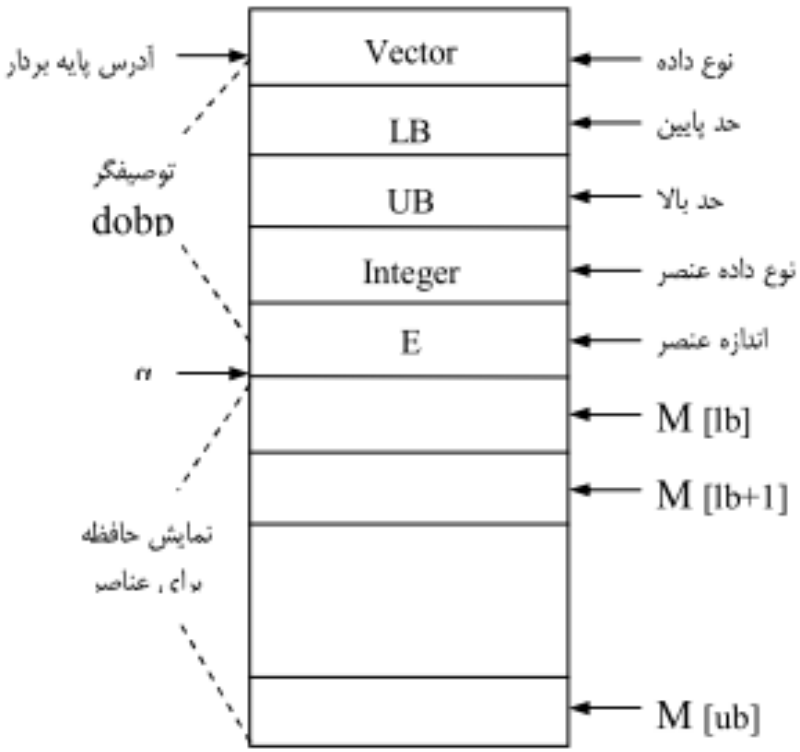
- عملیاتی که عنصری را از برداری انتخاب می کند اندیس گذاری نام دارد.
- برای ذخیره صفات بردار می توان از توصیفگر استفاده کرد.
- توصیفگرهای مربوط به پارامترهای آرایه می تواند به زیربرنامه ها ارسال شود ولی آرایه واقعی در جای دیگری ذخیره شده باشد.
  - نمایشهای حافظه به صورت فشرده و غیرفشرده
  - عملیات بر روی کل بردار

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

نمایش حافظه

- نمایش حافظه آرایه یک بعدی به صورت ترتیبی است. هر بردار ، توصیفگری در زمان ترجمه دارد که برای کنترل نوع ، نمایش حافظه و محاسبه فرمول دستیابی (تابع دستیابی) به کار می رود.



ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

پیاده سازی عملیات بردار

اگر α آدرس شروع اولین عنصر آرایه در حافظه و n اندازه هر قلم عنصر آرایه باشد و همچنین L و U کران های بالا و پایین آرایه یک بعدی A به صورت زیر باشند آنگاه آدرس عنصر A[i] به صورت زیر محاسبه می شود.

$$A: Array[L\ ..U\ ]\ of\ items; \qquad sizeof\ (items) = n;$$

$$\text{تعداد عناصر} = (U - L + 1)$$

$$\text{میزان حافظه مصرفی} = (U - L + 1) \times n$$

$$\text{آدرس عنصر } A[i] = (i - L) \times n + \alpha$$

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

آرایه های چند بعدی

- مشخصات و نحو: تفاوت آرایه چند بعدی و بردار در بازه اندیس هر بعد است.
- پیاده سازی: می توان آن را برداری از بردارها در نظر گرفت .

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

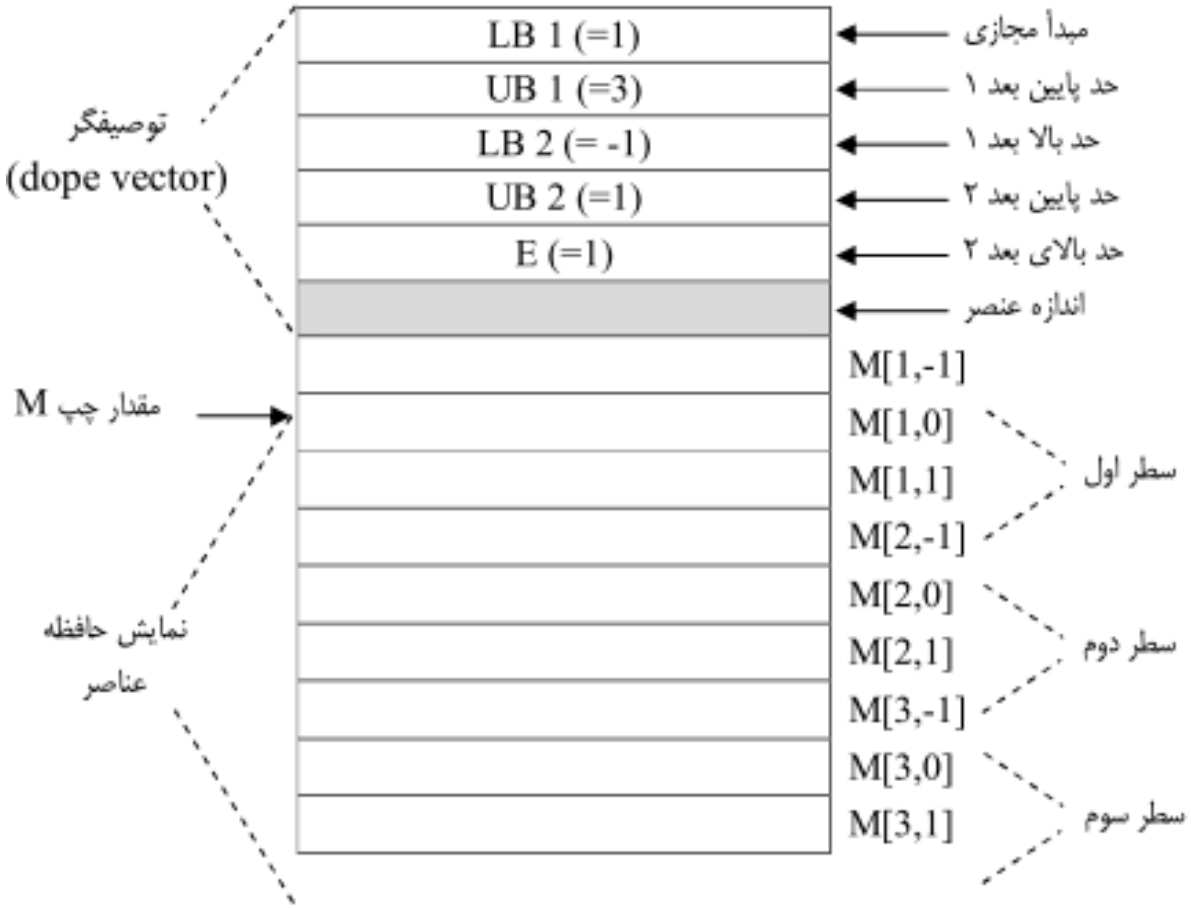
$$\underbrace{3\ ,\ 4\ ,\ 7}_{\text{سطراول}}\ ,\ \underbrace{6\ ,\ 2\ ,\ 5}_{\text{سطردوم}}\ ,\ \underbrace{1\ ,\ 3\ ,\ 8}_{\text{سطرسوم}}$$

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

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

پیاده سازی آرایه های چند بعد



ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

پیاده سازی عملیات آرایه دو بعدی

اگر α آدرس شروع اولین خانه آرایه در حافظه و n اندازه هر قلم عنصر آرایه باشد و همچنین L_1, L_2 کران های پایین و U_1, U_2 کران های بالا آرایه دوبعدی زیر باشند آنگاه آدرس عنصر A[i,j] به صورت زیر محاسبه می شود.

$$A : Array[L_1..U_1, L_2..U_2] \; of \; items; \qquad sizeof(items) = n;$$

آدرس عنصر به روش سطری $A[i,j] = [(i - L_1)(U_2 - L_2 + 1) + (j - L_2)] \times n + \alpha$

آدرس عنصر به روش ستونی $A[i,j] = [(j - L_1)(U_1 - L_1 + 1) + (i - L_1)] \times n + \alpha$

ساختمان داده ها (ادامه)

بردارها و آرایه ها (ادامه)

پیاده سازی عملیات آرایه سه بعدی

$$A: Array[L_1..U_1, L_2..U_2, L_3..U_3]\ of\ items; \qquad sizeof(items) = n;$$

آدرس عنصر به روش سطری $A[i,j,k] = [(i-L_1)(U_2-L_2+1)(U_3-L_3+1)+(j-L_2)(U_3-L_3+1)+(k-L_3)]\times n+\alpha$

آدرس عنصر به روش ستونی $A[i,j,k] = [(k-L_3)(U_2-L_2+1)(U_1-L_1+1)+(j-L_2)(U_1-L_1+1)+(i-L_1)]\times n+\alpha$

ساختمان داده ها (ادامه)

رکوردها

- مشخصات و نحو: ساختمان داده های خطی با طول ثابت هستند اما رکوردها از دو جهت متفاوتند:
  - عناصر رکورد ممکن است ناهمگن و از انواع مختلفی باشند.
  - عناصر رکورد دارای نام هستند.
- پیاده سازی: نمایش حافظه برای رکورد شامل یک بلوک از حافظه است که عناصر درآن به ترتیب ذخیره می شود.

```
Struct employee{
        int ID;
        int Age;
        float Salary;
  }A;
```

ساختمان داده ها (ادامه)

رکوردها

رکوردها و آرایه هایی با عناصر ساختاری

- عناصری از دو نوع مختلف با عناصری از انواع داده ترکیب شوند.
- انتخاب عناصر مستلزم دنباله ای از انتخابها 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.

A.ID=12

ساختمان داده ها (ادامه)

رکوردها

پیاده سازی رکورد

برای پیاده سازی رکورد از یک بلوک حافظه که در آن عناصر به ترتیب ذخیره می شوند استفاده می گردد. ممکن است برای هر عنصر از توصیفگری استفاده می شود ولی اغلب برای کل رکورد نیاز به توصیفگر نیست. فرمول دستیابی برای محاسبه آدرس محل i امین عنصر به صورت زیر است.

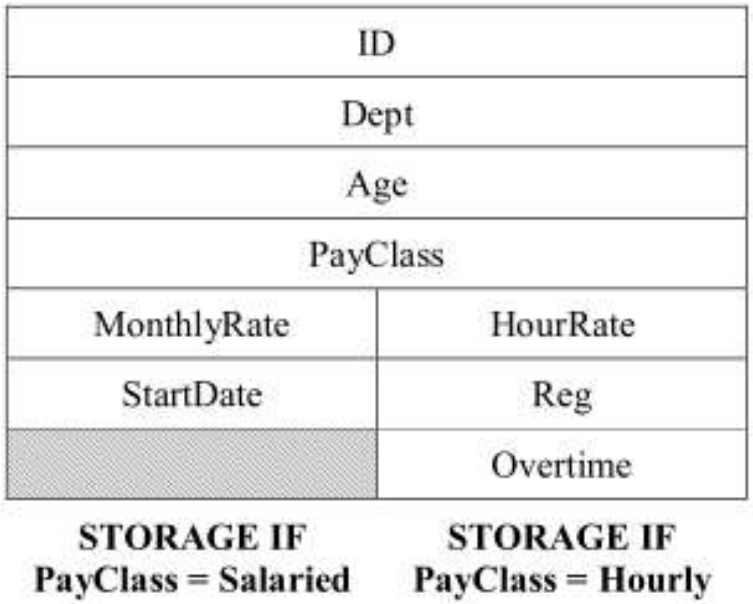
$$\text{آدرس عنصر i ام رکورد} = \alpha + \sum_{j=1}^{i-1} (\text{اندازه فیلد } j)$$

```
Struct employee{
        int ID;
        int Age;
        float Salary;
  }A;
```

ID	
Age	
Salary	

ساختمان داده ها (ادامه)

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



رکوردها

رکوردهای طول متغیر

- در رکوردهای طول متغیر عناصر ممکن است در یک زمان وجود داشته باشند و در زمان دیگر وجود نداشته باشند.

ساختمان داده ها (ادامه)

لیست ها

- مشخصات و نحو: لیستها همانند بردارها حاوی دنباله مرتبی از اشیا هستند.
- طول لیست معمولا ثابت نیست و در حین اجرای برنامه تغییر می کند.
- عناصر لیست معمولا ناهمگن هستند.
- اولین عنصر لیست را معمولاً راس می گویند.

ساختمان داده ها (ادامه)

لیست ها(ادامه)

- پیاده سازی: مدیریت حافظه منظم که برای بردارها و آرایه ها مفید است در اینجا قابل استفاده نیست.
- معمولاً از سازمان مدیریت حافظه پیوندی استفاده می شود.
- قلم لیست یک عنصر اولیه است که معمولاً شامل شی داده ای به اندازه ثابت است.
- لیسپ سه فلید اطلاعات نیازدارد:
  - یک فیلد نوع
  - دو فیلد اشاره گر لیست

فرم کلی هر گره

Type	Head	Tail

ساختمان داده ها (ادامه)

لیست ها(ادامه)

- شکلهای گوناگون لیستها:
- پشته ها و صفها
- درختها
- گرافهای جهت دار
- لیستهای خاصیت

ساختمان داده ها (ادامه)

مجموعه ها

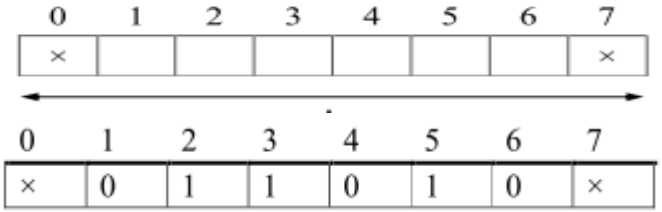
- مجموعه شی داده ای است که شامل مقادیر نامرتب و مجزا است.
- عملیات اصلی روی مجموعه ها عبارتند از:
  - عضویت
  - درج و حذف یک مقدار
  - اجتماع

ساختمان داده ها (ادامه)

مجموعه ها(ادامه)

- پیاده سازی:
  - مجموعه ساختمان داده ای است که عناصر مرتب را نشان می دهد.
  - مجموعه مرتب لیستی است که مقادیر تکراری آن حذف شده اند.
  - مجموعه نامرتب دو نمایش حافظه دارد
    - نمایش بیتی مجموعه ها: اگر اندازه مجموعه کوچک باشد مفید است. هر بیت نشان دهنده وجود یا عدم وجود یک عنصر در مجموعه است.

```
For  A=set  of  1..6 ;
```



A=[2,3,5]

OR: اجتماع

AND : اشتراک

AND رشته اول با مکمل رشته دوم: تفاضل

- نمایش درهم سازی مجموعه ها

انواع داده انتزاعی Abstract Data Type (ADT)

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.
- نمایش حافظه مربوط به مقادیر حقیقی و صحیح کالماً بسته بندی شده است یعنی از برنامه نویس پنهان است.
- برنامه نویس بدون اینکه از جزئیات نمایش حافظه این انواع اطلاع داشته باشد از اشیای داده آنها استفاده می کند.
- برنامه نویس فقط نام نوع و عملیاتی را برای دستکاری آن نوع فراهم می بیند

انواع داده انتزاعی(ادامه)

تکامل مفهوم نوع داده

انتزاع داده ها

- نوع داده انتزاعی :
- مجموعه ای از اشیای داده معمولاً با استفاده از یک یا چند تعریف نوع
- مجموعه ای از عملیات انتزاعی بر روی آن انواع داده
- بسته بندی تمام آنهابه 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.

انواع داده انتزاعی (ادامه)

پنهان سازی اطلاعات

- برای نوشتن برنامه بزرگ باید از استراتژی تقسیم و حل استفاده کرد
- طراحی ماژول معمولاً به دوروش انجام می شود:
  - ماژولهای تجزیه تابعی: ماژول ها تجزیه تابعی برنامه را نشان می دهند ، که ساختارهای رویه ها ، توابع ، زیر برنامه ها از آن نتیجه می شود.
  - ماژولهای تجزیه داده ای

انواع داده انتزاعی (ادامه)

پنهان سازی اطلاعات (ادامه)

- فلوچارت انتزاعی از ساختار کنترل سطح دستور برنامه است.
- روشهای طراحی برنامه ها:
  - اصلاح مرحله ای
  - برنامه نویسی ساخت یافته
  - برنامه نویسی پیمانه ای
  - برنامه نویسی بالا به پایین

انواع داده انتزاعی (ادامه)

پنهان سازی اطلاعات(ادامه)

- زبان برنامه سازی انتزاع را به دو روش پشتیبانی می کند:
  - با تدارک کامپیوتر مجازی که کاربرد آن ساده تر و قدرت آن بیش از کامپیوتر سخت افزار است.
  - زبان امکاناتی را فراهم می کند که برنامه نویس می تواند انتزاعها را به وجود آورد.
- بسته بندی اصلاح برنامه را آن می کند.
- زیربرنامه ها مکانیزم بسته بندی را شکل می دهند که تقریباً در هر زبانی وجود دارد.

بسته بندی با زیربرنامه ها

- زیر برنامه یک عملیات انتزاعی است که توسط برنامه نویس تعریف می شود.
- دو دیدگاه از زیربرنامه در اینجا مهم است:
  - سطح طراحی برنامه : روش نمایش عملیات انتزاعی است که برنامه نویس در مقایسه با عملیات اولیه موجود در زبان ، تعریف می کند.
  - سطح طراحی زبان: طراحی و پیاده سازی امکاناتی است که برای تعریف و فراخوانی زیر برنامه می باشد.

بسته بندی با زیربرنامه ها(ادامه)

زیر برنامه ها و عملیات انتزاعی

- مشخصات زیربرنامه:
  - نام
  - امضای یا نمونه اولیه زیربرنامه
  - فعالیتی که توسط زیربرنامه انجام می شود.
- پیاده سازی زیربرنامه شامل:
- پیاده سازی توسط بدنه زیربرنامه تعریف می شود که متشکل از اعلان داده های محلی است
- دستوراتی که عملکرد زیربرنامه را مشخص می کند.

بسته بندی با زیربرنامه ها (ادامه)

تعریف و فراخوانی زیربرنامه

- نعریف زیربرنامه چیزی است که در برنامه نوشته می شود و تنها اطلاعاتی است که در زمان ترجمه وجود دارد یعنی نوع متغیرهای زیر برنامه مشخص است.
- تعریف زیربرنامه خاصیت ایستای یک برنامه است.
- درحین اجرای برنامه اگر زیربرنامه ای فراخوانی شود سابقه فعالیتی از آن زیربرنامه ایجاد می شود.
- تعریف زیربرنامه قالبی برای ایجاد سابقه فعالیت در حین اجرا است.
- شی داده در حین اجرا برنامه ایجاد می شود:
  - در حین ورود به زیربرنامه
  - توسط عملیاتی مثل malloc یا New

```
Float Fn(float x, int y)
Fn : Real * Integer → Real
```

بسته بندی با زیربرنامه ها (ادامه)

تعریف و فراخوانی زیربرنامه(ادامه)

فعالیت یا فراخوانی زیربرنامه Activation

- فعال سازی زیر برنامه فقط در حین اجرای برنامه وجود دارد.
- با هر بار فراخوانی زیربرنامه ، آن فعال می شود.
- فعالیت زیر برنامه دارای طول عمر است که از فراخوانی زیربرنامه شروع می شود و تا از بین رفتن آن ادامه دارد.

بسته بندی با زیربرنامه ها (ادامه)

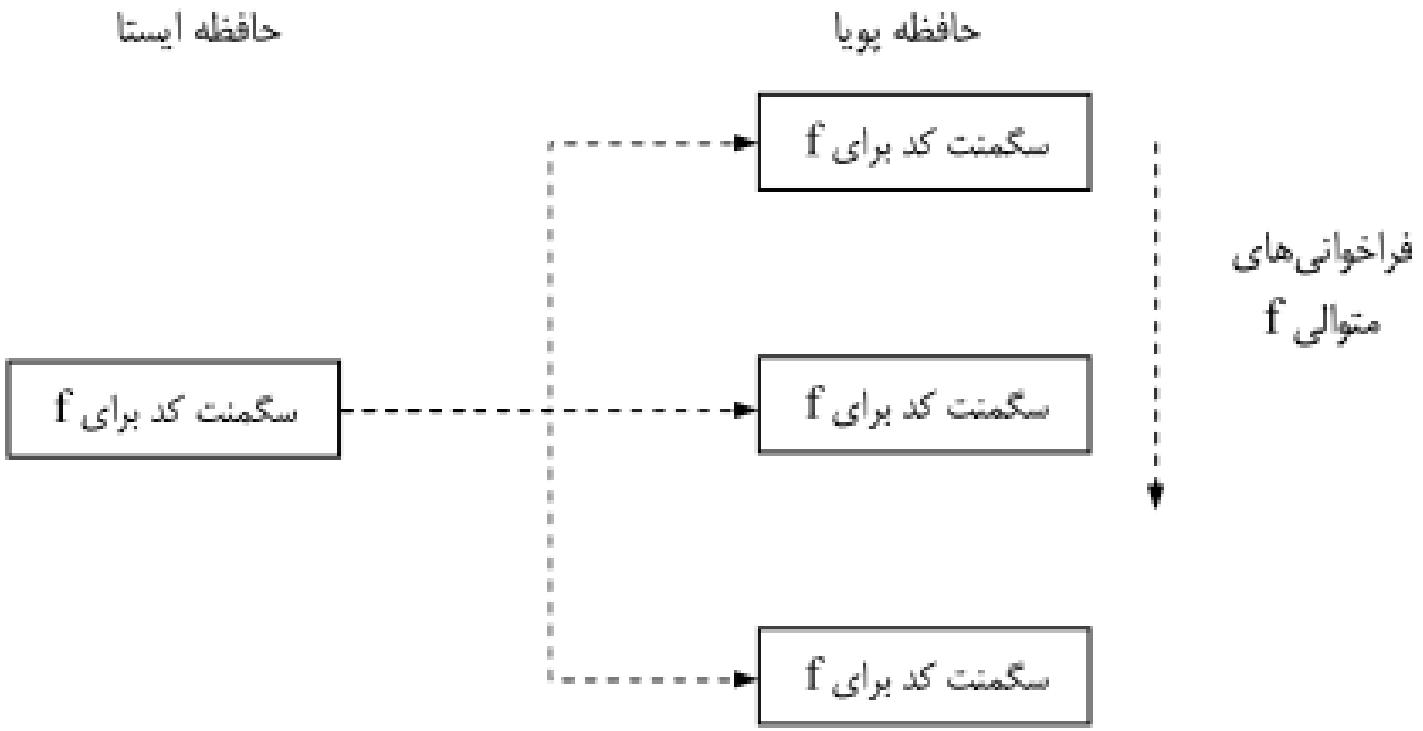
تعریف و فراخوانی زیربرنامه(ادامه)

پیاده سازی تعریف و فراخوانی زیربرنامه

- شامل دو بخش تقسیم می شود:
  - **بخش ایستا** که سگمنت کد نام دارد و حاوی ثوابت و کد اجرایی است. این بخش در حین اجرای زیربرنامه باید ثابت باشد. در نتیجه یک کپی از آن بین تمام فعالیت های زیربرنامه به اشتراک گذاشته خواهد شد.
  - **بخش پویا** که رکورد فعالیت نام دارد ، 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 تمامی فعالیت های زیربرنامه یکسان است اما مقادیر متفاوتی در آنها وجود دارند.

بسته بندی با زیربرنامه ها (ادامه)

تعریف و فراخوانی زیربرنامه(ادامه)



بسته بندی با زیربرنامه ها (ادامه)

تعریف و فراخوانی زیربرنامه(ادامه)

```
Float FN(float x,int
y)
{
      const intval=2;
      #define finalva
10;
      float M(10);
      int N;
      N=intval;
      if (n<finalval)
      {
            …
      }
      return
(20*x+M(N))
```

سگمنت کد زیربرنامه FN

رکورد فعالیت FN (الگو)

بسته بندی با زیربرنامه ها (ادامه)

تعریف زیربرنامه به عنوان اشیای داده

- ترجمه عملیاتی است که تعریف زیربرنامه را به شکل رشته کاراکتری گرفته شی داده زمان 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 فعالیت را اجرا می نماید.

بسته بندی با زیربرنامه ها (ادامه)

تعریف زیربرنامه به عنوان اشیای داده

- **Prolog**: این اعمال باید قبل از اجرای دستورات زیر برنامه گنجانده شوند. انجام این مقدمات توسط مترجم قبل از اجرای کد زیر برنامه انجام می شود و شامل عملیاتی مثل : تنظیم رکورد فعالیت ، انتقال پارامترها ، ایجاد پیوند برای ارجاع های غیر محلی و ... (عملیات انتقال پارامترها ، push کردن ثبات ها و فلگ ها)
- **Epilog** : این اعمال هنگام خاتمه زیر برنامه انجام می شود تا نتایج را برگرداند و حافظه رکورد فعالیت را آزاد کنند. برای این اعمال نیز دستوراتی توسط مترجم در انتهای کد اجرایی قرار داده می شود. (عملیات انتقال نتایج ، pop کردن ثبات ها و فلگ ها)

دستورات مربوط به Prolog
کد اجرائی
دستورات مربوط به Epilog

تعریف نوع (Type definition)

- یک زبان برنامه سازی باید امکاناتی جهت تعریف نوع جدید با توجه به نوع های ائلیه موجود در زبان فراهم سازد. هر نوع شامل یک Name و یک Description می باشد.
- پیاده سازی: اطلاعات موجود در اعلان متغیرها در زمان ترجمه برای تعیین نمایش حافظه اشیا و اهداف مدیریت حافظه و کنترل نوع بکار می رود.

```
Type Rational:record
      Numerator=integer;
      Denominator: integer;
End;
Var A, B, C: Rational;
```

تعریف نوع(ادامه)

هم ارزی نوع

- کنترل نوع چه به صورت ایستا و چه به صورت پویا مقایسه بین نوع آرگومان های واقعی و نوع داده هایی است که عملیات انتظار آن را دارد.
- اگر این 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.
- دو روش برای بررسی هم ارزی یا مساوی بودن نوع داریم:
  - هم ارزی نام
  - هم ارزی ساختار
  - معایب:
    - هر شی که در انتساب بکار می رود باید دارای نام باشد
    - یک تعریف نوع باید در سراسر برنامه یا بخش بزرگی از برنامه قابل استفاده باشد

تعریف نوع(ادامه)

هم ارزی نوع (ادامه)

- هم ارزی نام : دو شی داده از دو نوع ، هنگامی هم ارز نام دارند که نام نوع آنها یکسان باشد.
  - معایب:
    - هر شی که در انتساب بکار می رود باید دارای نام باشد یعنی انواع داده بی نام وجود ندارد
    - یک تعریف نوع باید در سراسر برنامه یا بخش بزرگی از برنامه قابل استفاده باشد

```
Type   Vect1:array [1...10] of integer;
       Vect2:array [1...10] of integer:
Var X, Z: Vect1;  Y: vect2;
Procedure sub (A: Vect1)
Begin
     ...
end;
Begin
     X:=Z
     X:=Y;
     Sub(Y);
End;
```

(X: =Z) در هم ارزی نام معتبر است چون هر دو متغیر دارای نام نوع یکسان هستند.

(X:=Y) در هم ارزی نام معتبر نیست چون X از نوع Vect1 و Y از نوع Vect2 است.

تعریف نوع(ادامه)

هم ارزی نوع(ادامه)

- هم ارزی ساختاری: دو نوع داده هنگامی هم ارز ساختاری دارند که ساختار داخلی آنها یکسان باشد یعنی تمام اشیاء داده آنها از یک گونه نمایش حافظه استفاده کنند.
- در مثال قبل Vect1 و Vect2 هم ارز ساختاری هستند زیرا تعداد عناصر ، نوع و ترتیب آنها یکسان است.
  - مزیت: انعطاف پذیری برنامه را افزایش می دهد.
  - معایب
    - آیا ترتیب فیلدها باید یکی باشد ...
    - دو متغیر ممکن است به طور تصادفی از نظر ساختاری یکسان باشند
    - تعیین هم ارزی ساختاری در مورد انواع پیچیده هزینه ترجمه دارد.

تعریف نوع(ادامه)

تعریف انواعی که پارامتردارند

- پیاده سازی: تعریف نوع پارامتردار به عنوان الگویی در زمان ترجمه منظور می شود با این تفاوت که وقتی کامپایلر اعلان یک متغیر را با لیست پارامترهایی که بعد از نام نوع می آید را ترجمه می کند.

فصل پنجم

کنترل ترتیب اجرا

کنترل ترتیب اجرا

دو جنبه کار :

- کنترل ترتیب اجرای عملیات (عملیات اولیه و عملیات تعریف شده توسط کاربر)
- کنترل داده ، کنترل انتقال داده ها بین زیر برنامه ها و برنامه ها می باشد.

کنترل ترتیب ضمنی و صریح

در حالت معمولی دستورات یک برنامه پشت سر هم اجرا می شوند اما در مواردی ترتیب اجرا بنا به دلایلی نظیر دستورات شرطی یا حلقه ها عوض می شوند.

ساختارهای کنترل ترتیب به چهار دسته:

- ساختارهایی که در عبارات محاسباتی مورد استفاده قرار می گیرند کانند تقدم عملگرها و پرانتزها.
- ساختارهایی که بین دستورات یا گروهی از دستورات به کار می روند مانند جملات ترکیبی ، جملات شرطی ، حلقه ها.
- برنامه نویسی اعلانی مثل پرولوگ.
- کنترل ترتیب در زیر برنامه ها مانند فراخوانی زیربرنامه ها که کنترل برنامه را از نقطه ای به نقطه ای دیگر انتقال می دهند.

کنترل ترتیب ضمنی و صریح

ساختارهای کنترل ترتیب ممکن است ضمنی یا صریح باشد:

- ساختار کنترل ضمنی: توسط زبان تعریف شده اند و بکار گرفته می شوند مانند تقدم عملگر ضرب نسبت به جمع.
- ساختار کنترل ترتیب صریح: برنامه نویس تهیه می کند تا ساختارهای ضمنی تعریف شده توسط زبان را عوض کند مانند استفاده از پرانتز در عبارات ریاضی یا استفاده از دستورات go to.

ترتیب اجرا در عبارات محاسباتی

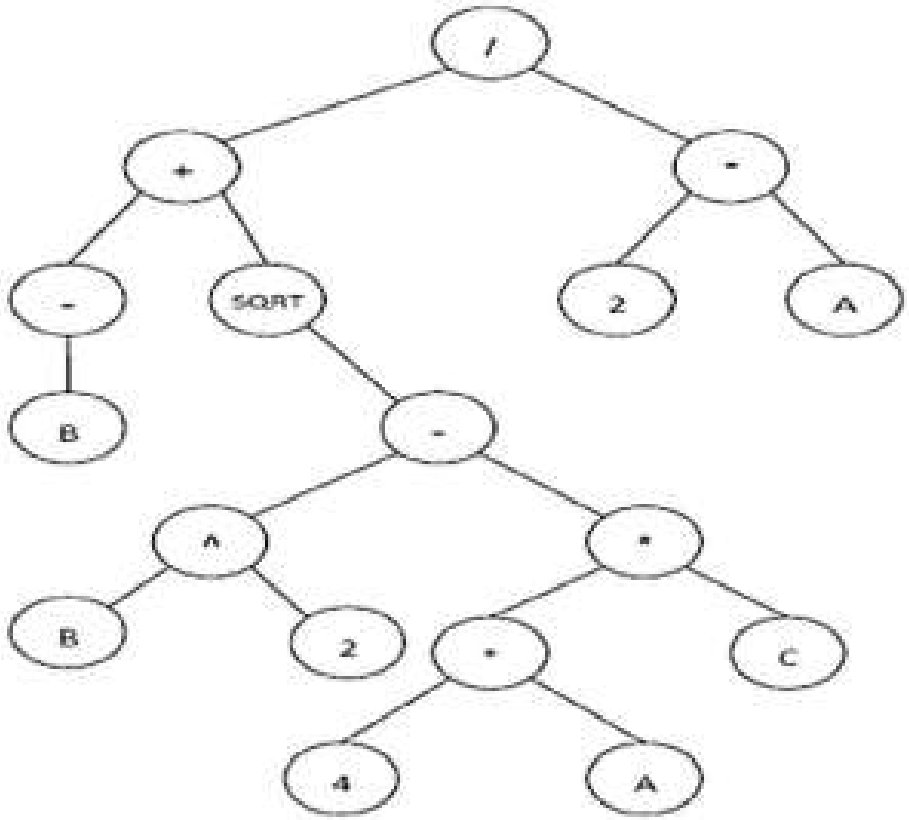
نمایش درختی عبارات محاسباتی

- با در نظر گرفتن عملیات در عبارات آرگومانهای عملیات را عملوند می نامیم.
- مکانیزم کنترل ترتیب در عبارات ترکیب تابعی است یعنی عملیات و عملوندهایش مشخص می شود.
- نمایش درختی ساختار کنترلی عبارت را نشان می دهد.
- در این روش ریشه درخت عملیات اصلی ، برگ ها معرف داده ها و گره های بین ریشه و برگ ها عملیات میانی را نشان می دهند.

ترتیب اجرا در عبارات محاسباتی

نمایش درختی عبارات محاسباتی (ادامه)

$$\frac{-B + SQRT\,(B^2 - 4*A*C)}{2*A}$$



ترتیب اجرا در عبارات محاسباتی (ادامه)

نمایش درختی عبارات (ادامه)

نحو عبارات

- در برنامه ها باید درختها را به صورت خطی مشخص کرد
- **نشانه گذاری پیشوندی یا perfix:** در این روش عملگرها قبل از عملوندهایشان قرار می گیرند. مثال (a+b)*c به صورت *+abc می شود.
- **نشانه گذاری پسوندی یا postfix:** در این روش عملگرها بعد از عملوندهایشان قرار می گیرند. مثال (a+b)*c به صورت ab+c* می شود.
- **نشانه گذاری میانوندی یا infix:** در این روش عملگرهای دودویی در بین عملوندهایشان قرار می گیرند و تقدم عملگرها توسط پرانتز تعیین می شود. مثال (a+b)*c

ترتیب اجرا در عبارات محاسباتی (ادامه)

نمایش درختی عبارات (ادامه)

معنای عبارات

- ارزشیابی Postfix و Prefix:

برای ارزیابی عبارات Prefix و Postfix، در هر دو روش عملوندها در پشته Push می شوند و با رسیدن به عملگر، از بالای پشته دو عمل Pop انجام می شود با این تفاوت که در روش Postfix عبارات از چپ به راست و بالای پشته دومین عملوند محسوب می شود در حالیکه در روش Prefix عبارات از راست به چپ و بالای پشته اولین عملوند محسوب می شود. علاوه

- ارزشیابی Infix: چون تقدم عملگرها مشخص نیست باید به همراه یکی دیگر از فرم های نمایش استفاده شود که مناسب نیست.

کنترل ترتیب بین دستورات

دستورات اصلی

- انتساب به اشیای داده
  - دستور انتساب: هدف اولیه انتساب مقدار راست عبارت را به مقدار چپ آن نسبت دهد.
  - دستورات ورودی
  - سایر عملیات انتساب
- شکلهای مختلف کنترل ترتیب سطح دستور
  - ترکیب: دستورات پشت سر هم و به ترتیب اجرا می شوند مثل جملات بین begin و end در یک بلاک
  - انتخاب : در این دستورات دو یا چند مسیر جهت اجرا وجود دارد مانند if و case
  - تکرار: دنباله ای از دستورات که به صورت تکراری اجرا می شوند مانند for و while

کنترل ترتیب بین دستورات (ادامه)

دستورات اصلی (ادامه)

- کنترل ترتیب ضمنی
  - دستور goto: در زبان های اولیه بیشتر مورد استفاده قرار می گرفت ولی با ایجاد مفهوم برنامه نویسی ساخت یافته استفاده از آن توصیه نمی شود.
  - Goto غیرشرطی: کنترل را به یک خط خاص انتقال می دهد. goto next;
  - Goto شرطی: در صورتیکه شرط ذکر شده درست باشد کنترل منتقل می شود.

    If (a==0) goto next;
  - دستور break: برای خارج شده از بلاکی از برنامه استفاده می شود.

کنترل ترتیب بین دستورات (ادامه)

دستورات اصلی (ادامه)

- طراحی برنامه نویسی ساخت یافته
- امتیاز goto :
  - اگر برچسبها از نظر نحوی ساده باشندمستقیماً توسط سخت افزار پشتیبانی می شود و کارایی آن بالااست
  - استفاده از آن در برنامه های کوچک ساده است
  - برای برنامه نویسان اسمبلی و کسانی که با زبانهای قدیمی برنامه نویسی می کنند آشنا است
  - هدف کلی برای نمایش شکلهای دیگری از کنترل است

کنترل ترتیب بین دستورات (ادامه)

دستورات اصلی (ادامه)

- طراحی برنامه نویسی ساخت یافته (ادامه)
- معایب goto :
  - عدم وجود ساختار سلسله مراتبی برنامه
  - ترتیب دستور ات در متن برنامه لازم نیست با ترتیب اجرا یکی باشد.
  - گروهی از دستورات ممکن است اهداف متعددی داشته باشد.
  - برنامه نویسی ساخت یافته

کنترل ترتیب بین دستورات(ادامه)

کنترل ترتیب ساخت یافته: ترکیبی از انواع دستورات در یک برنامه داریم

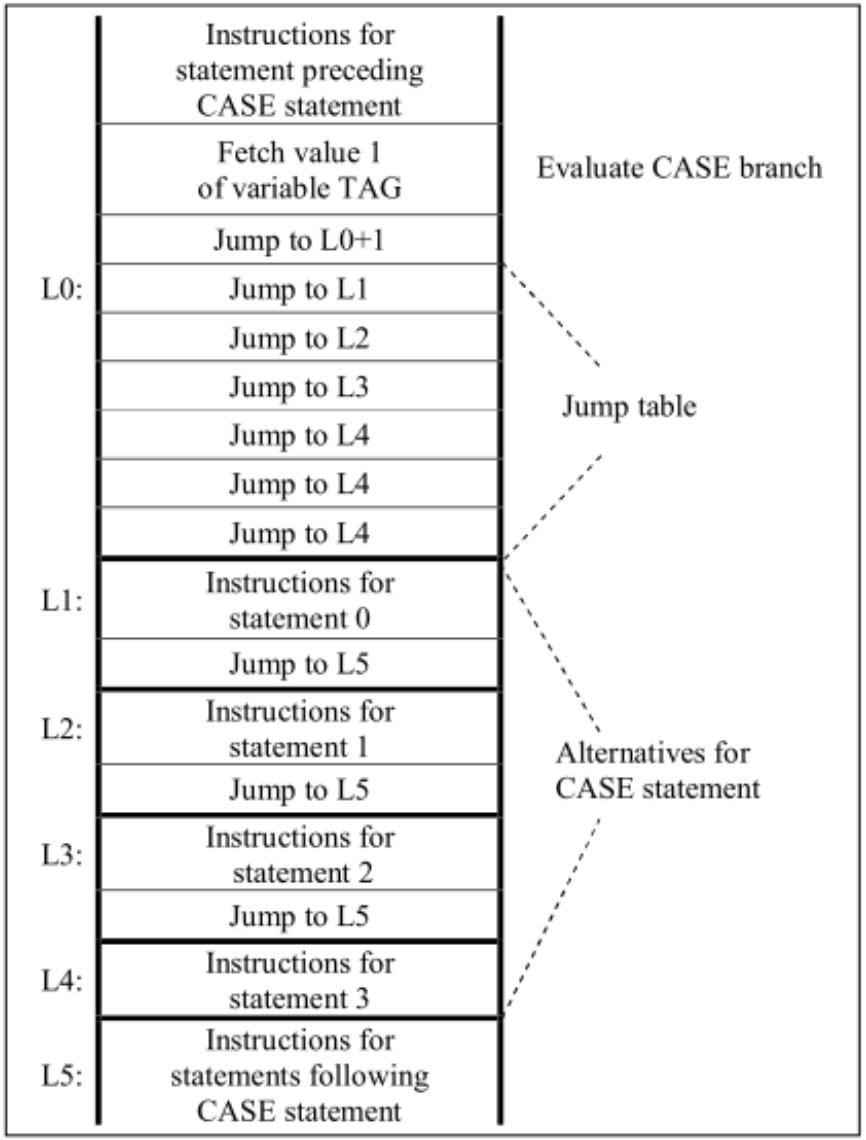
- دستورات شرطی (if-else و case)
- دستور if با پرش سخت افزاری پیاده سازی می شوند ولی case به کمک جدول پرش پیاده سازی می شود تا از تست های تکراری یک متغیر جلوگیری شود و کارایی اجرا بالا رود.

- جدول پرش آرایه ای است که به صورت ترتیبی در حافظه ذخیره شده و هر یک از عناصر آن یک دستور پرش غیر شرطی است. ابتدا عبارتی که شرط دستور case را پدید آورده ارزیابی شده و نتیجه آن به یک مقدار صحیح تبدیل می شود که آفستی را در جدول پرش نشان می دهد.

- هزینه ترجمه case به دلیل ساختن جدول پرش بیشتر از if تودرتو است ولی هزینه اجرای آن از if های تودرتو خیلی کمتر است.

کنترل ترتیب بین دستورات(ادامه)

```
Case tag is
      when 0 => begin
            statement 0
      end;
      when 1 => begin
            statement 1
      end;
      when 2 => begin
            statement 2
      end;
      when others => begin
            statement 3
      end;
end case
```



کنترل ترتیب بین دستورات(ادامه)

کنترل ترتیب ساخت یافته (ادامه)

- دستورات تکرار
  - تکرار ساده
  - تکرار در صورتی که شرط برقرار باشد.
  - تکرار با افزایش یک شمارنده
  - تکرار مبتنی بر داده ها
  - تکرار نامتناهی

فصل ششم

کنترل زیر برنامه

کنترل ترتیب زیر برنامه

- زیربرنامه ساده فراخوانی – برگشت: هر برنامه متشکل از یک برنامه اصلی است که در حین اجرا می تواند زیربرنامه هایی را فراخوانی کند و هر زیربرنامه زیربرنامه های دیگر را .
- دستور فراخوانی زیربرنامه مثل این است که قبل از اجرا یک کپی از زیربرنامه در نقطه ای که فراخوانی صورت می گیرد قرار داده شود.

کنترل ترتیب زیر برنامه (ادامه)

فرضیه های موجود:

- زیربرنامه ها نمی توانند بازگشتی باشند.
- نیاز به دستور فراخوانی صریح است
- زیربرنامه ها در هر فراخوانی باید به طور کامل اجرا شوند
- کنترل به نقطه فراخوانی بر می گردد.
- در هر زمان فقط یک زیربرنامه کنترل را در دست دارد.

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت

- پیاده سازی: نیاز به چیزهای دیگر:
- بین تعریف زیربرنامه و سابقه فعالیت آن تفاوت وجود دارد.
- سابقه فعالیت دو بخش دارد : سگمنت کد و رکورد فعالیت
- سگمنت کد در حین اجرا تغیر نمی کند.
- رکورد فعالیت در هر بار اجرای زیربرنامه ایجاد می شود و با خاتمه زیربرنامه از بین می رود.

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)

پیاده سازی پشته ای

- ساده ترین تکنیک مدیریت حافظه زمان اجرا پشته است.
- برای کنترل مدیریت حافظه نیاز به اشاره گر پشته است.
- در پاسکال یک پشته مرکزی و یک ناحیه حافظه به طور ایستا تخصیص می یابد.
- پشته در لیسپ به صورت فراخوانیهای زیربرنامه به صورت تو در تو می باشد.

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)

- **اشاره گر دستور فعلی(CIP):** که به دستور داخل سگمنت کد که قرار است اجرا شود اشاره می‌کند و مترجم دستوری را که CIP به آن اشاره می‌کند بازیابی کرده و آن را اجرا می‌کند.
- **اشاره گر محیط فعلی(CEP):**اشاره گری است که به رکورد فعالیت فعلی (مربوط به قطعه کدی که در حال اجرا است) اشاره می‌کند بنابراین چون تمام رکوردهای فعالیت یک زیر برنامه از یک سگمنت کد استفاده می‌کنند دانستن دستور فعلی کافی نیست باید اشاره گر CEP هم باشد تا رکورد فعالیت مورد نظر را مشخص نماید. به عنوان مثال وقتی دستوری درکد، به متغیر x مراجعه می‌کند آن متغیر در رکورد فعالیت وجود دارد، هر رکورد فعالیت آن زیربرنامه، شیء داده ای به نام x دارد که ممکن است محتویاتش با دیگری فرق داشته باشد.

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)

نقطه بازگشت:

شی داده ای که توسط سیستم تعریف می شود و در هر زیر برنامه دارای مقداری است که نشان می دهد به هنگام بازگشت از این زیربرنامه به کدام نقطه و کدام محیط باید برگشت انجام شود. این نقطه بازگشت در رکورد فعالیت زیر برنامه جاری ذخیره می شود و آن را با زوج (ep,ip) نشان می دهند.

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)

در زمان شروع اجرای برنامه اصلی، اشاره گر CEP به رکورد فعالیت برنامه اصلی اشاره می‌کند و CIP به اولین دستور از سگمنت کد برنامه اصلی اشاره می‌کند وقتی کنترل اجرا به دستور فراخوانی زیر برنامه رسید یک رکورد فعالیت از آن زیر برنامه ایجاد می‌شود و CEP به آن اشاره می‌کند و CIP به اولین دستور سگمنت کد زیر برنامه اشاره خواهد کرد. جهت برگشت صحیح و درست از یک زیر برنامه، مقادیر CEP و CIP را می‌توان در رکورد فعالیت زیر برنامه ای که فراخوانی شده است ذخیره کرد تا در زمان بازگشت از زیر برنامه، اجرا از نقطه بعد از فراخوانی زیر برنامه از سر گرفته شود. شکل زیر نمونه ای از عملیات فوق را برای برنامه اصلی که دو بار زیر برنامه A و یک بار زیر برنامه B را فراخوانی کرده، نشان می‌دهد. زیر برنامه A خودش یکبار زیر برنامه B را صدا می‌زند. بنابراین به طور خلاصه خواهیم داشت:

عملیاتی که در هنگام فراخوانی (Call) زیر برنامه رخ می دهند:

الف- ایجاد رکورد فعالیت جدید در زیربرنامه

ب- ذخیره کردن CEP و CIP جاری به عنوان نقطه بازگشت در رکورد فعالیت ایجاد شده

ج- مقدار دهی آدرس رکورد فعالیت ایجاد شده و ابتدای زیربرنامه فراخوانی شده در داخل CEP و CIP به عنوان مقدار جدید آنها (این کار باعث انتقال کنترل اجرا به ابتدای زیرب رنامه فراخوانی شده و محیط جدید می گردد)

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)

عملیاتی که در هنگام بازگشت(Return) زیربرنامه رخ می دهند:

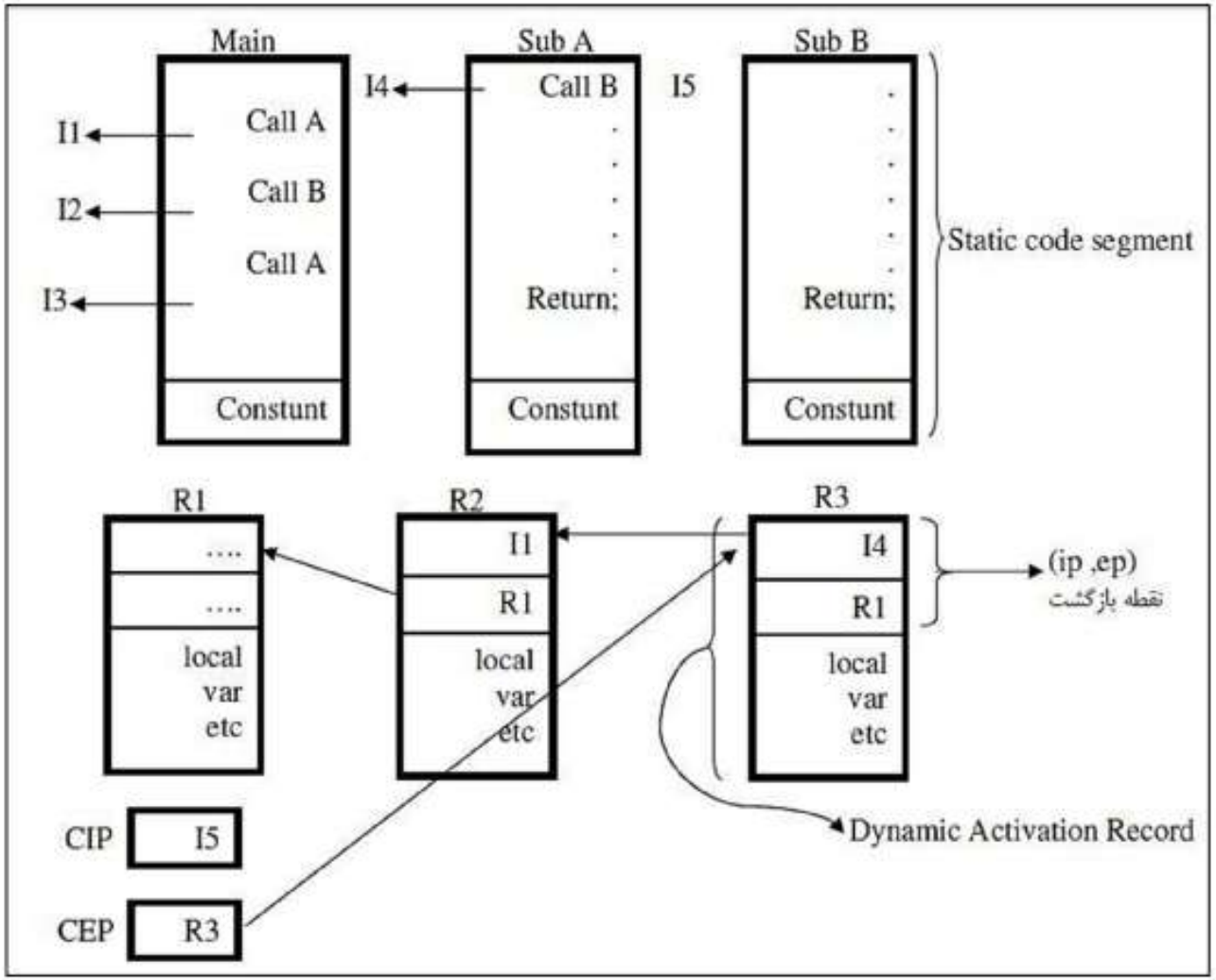
الف- کپی کردن مقدار نقطه بازگشت در یک محل موقت

ب- از بین بردن حافظه مربوط به رکورد فعالیت جاری

ج- کپی مقادیر نقطه بازگشت در CEP و CIP (این کار منجر به بازگشت کنترل به محل فراخوانی بر می گردد)

کنترل ترتیب زیر برنامه (ادامه)

زیربرنامه های فراخوانی - برگشت (ادامه)



کنترل ترتیب زیر برنامه(ادامه)

زیربرنامه های بازگشتی

- مشخصات: اگر فراخوانی بازگشتی زیربرنامه امکانپذیر باشد A می تواند هر زیربرنامه ای از جمله خودش را فراخوانی کند.
- پیاده سازی: در هنگام فراخوانی هر زیربرنامه رکورد فعالیت جدیدی ایجاد می شود و با دستور برگشت از بین می رود.

```
Int fact (int n) {
If(n<=1) ret 1;
else ret fact(n-1)*n ;}
```

کنترل ترتیب زیر برنامه(ادامه)

زیربرنامه های بازگشتی (ادامه)

پیاده سازی:

برای پیاده سازی زیر برنامه‌های بازگشتی، به دلیل امکان وجود چند رکورد فعالیت به طور همزمان، هر دو اشاره گرCEP وCIP باید استفاده شوند. پیاده سازی زیر برنامه های بازگشتی مشابه پیاده سازی زیر برنامه های فراخوانی- بازگشت ساده است با این تفاوت که برای مدیریت رکوردهای فعالیت از پشته مرکزی استفاده می شود. به ازای هر Call، رکورد فعالیت زیر برنامه مربوطه ایجاد شده و آدرس نقاط برگشت در رکورد فعالیت ایجاد شده در پشته ذخیره می شوند (Push) و به ازای هر Return، آدرس نقاط برگشت از رکورد فعالیت موجود در پشته استخراج شده و در اشاره گرهای CEP و CIP کپی می شود و سپس رکورد فعالیت مربوطه از بالای پشته برداشته می شوند (Pop).

صفات کنترل داده ها

اسامی و محیطهای ارجاع

- اشیای داده به دو روش به عنوان عملوند یک عملیات مورد استفاده قرار می گیرند:
  - انتقال مستقیم: مثل 2*z در عبارت x=y+2*z و نامی برای آن در نظر گرفته نشده است.
  - مراجعه از طریق شی داده ای که دارای نام است. مثال x=y+2*z در اینجا z دلالت بر داده ای دارد مه باید در عمل ضرب استفاده شود.
- انتقال مستقیم برای کنترل داده ها بین عبارت بکار می رود.

صفات کنترل داده ها (ادامه)

اسامی و محیطهای ارجاع (ادامه)

- عناصری از برنامه که دارای نام هستند (عناصر مشترک):
  - اسامی متغیرها
  - اسامی پارامترهای مجازی
  - اسامی زیربرنامه ها
  - اسامی انواع تعریف شده
  - اسامی ثوابت تعریف شده
  - برچسب دستورات
  - اسامی استثناها
  - اسامی عملیات اولیه مثل +و* و sort
  - اسامی ثوابت لیترال مثل 25/3و 17

صفات کنترل داده ها (ادامه)

اسامی و محیطهای ارجاع (ادامه)

وابستگیها و محیطهای ارجاع

- وابستگی :

انقیاد هر نام به یک شی داده یا هر نام به یک زیر برنامه خاص را وابستگی گویند. در آغاز برنامه اصلی در تعریف متغییرها، هر متغییر را به یک شی داده مقید می کنیم و با این کار وابستگی بین شناسه (نام) تعریف شده و شی داده مربوط را ایجاد می کنیم.

- محیط ارجاع:

هر زیر برنامه ای دارای مجموعه ای شناسه است که در طول اجرا از آن‌ها استفاده می‌کند به این مجموعه، محیط ارجاع آن زیر برنامه می گویند. محیط ارجاع زیر برنامه در حین اجرا متغیر نیست. این محیط ارجاع در حین ایجاد رکورد فعالیت زیر برنامه ایجاد و تنظیم می‌گردد و با از بین رفتن رکورد فعالیت زیر برنامه از بین می‌رود. مقادیر موجود در اشیا داده ممکن است تغییر کنند ولی وابستگی نام‌ها به اشیا داده و زیر برنامه‌ها (محیط ارجاع زیر برنامه) تغییر نمی‌کنند. انواع محیط ارجاع عبارتند از:

صفات کنترل داده ها (ادامه)

اسامی و محیطهای ارجاع (ادامه)

وابستگیها و محیطهای ارجاع(ادامه)

- انواع محیطهای ارجاع
  - **محیط ارجاع محلی**: شامل کلیه اسامی که هنگام ورود به زیر برنامه ایجاد شده ، مثل اسامی پارامترهای مجازی ، متغیرهای محلی و زیر برنامه هایی که درون زیر برنامه مورد نظر تعریف می شود.
  - **محیط ارجاع غیرمحلی**: اسامی که داخل زیر برنامه قابل دسترس هستند ولی هنگام ورود به زیر برنامه ایجاد نمی شوند. مثل متغیرهای محلی static در زبان C.
  - **محیط ارجاع سراسری**:کلیه اسامی که شروع اجرای برنامه اصلی پدید آمده و در کل برنامه ها قابل دسترس هستند.
  - **محیط ارجاع از پیش تعریف شده**: کلیه اسامی که توسط کامپایلر تعریف شده و تمامی زیر برنامه ها می توانند از آنها استفاده کنند مثل int و float.

صفات کنترل داده ها (ادامه)

اسامی و محیطهای ارجاع (ادامه)

وابستگیها و محیطهای ارجاع(ادامه)

```
Program main;
      var A,B,C:real;
      procedure sub1(A:real);
            var D:real
            procedure sub2(C:real);
                  var D:real;
            begin
                  …
            end;
      begin
            …
            sub2(B);
            …
      end;
begin
      …
      sub1(A);
      …
end.
```

محیط ارجاع برای sub2	محیط ارجاع برای sub1	محیط ارجاع برای main
محلی: C و D غیرمحلی از sub1: A ، sub2 غیرمحلی از main: A ، sub2، B	محلی: A ، D و sub2 غیرمحلی: B ، C و sub1	محلی(سراسری): A ، B ، C و sub1

صفات کنترل داده ها (ادامه)

اسامی و محیطهای ارجاع (ادامه)

وابستگیها و محیطهای ارجاع(ادامه)

- **قابلیت مشاهده** Visibility: اگر یک وابستگی برای یک زیر برنامه بخشی از محیط ارجاع آن برنامه باشد ، می گوییم آن وابستگی در زبر برنامه قابل رویت است. در غیر این صورت به آن وابستگی پنهان گفته می شود.
- **حوزه پویا**: از مجموعه ای از رکوردهای فعالیت زیر برنامه که توسط آن زیر برنامه قابل مشاهده است ، تشکیل شده است.
- **عملیات ارجاع**: عملیاتی که یک شناسه و یک محیط ارجاع را گرفته ، شناسه را در محیط پیدا کرده و یک شی داده یا زیر برنامه را بر می گرداند.

صفات کنترل داده ها(ادامه)

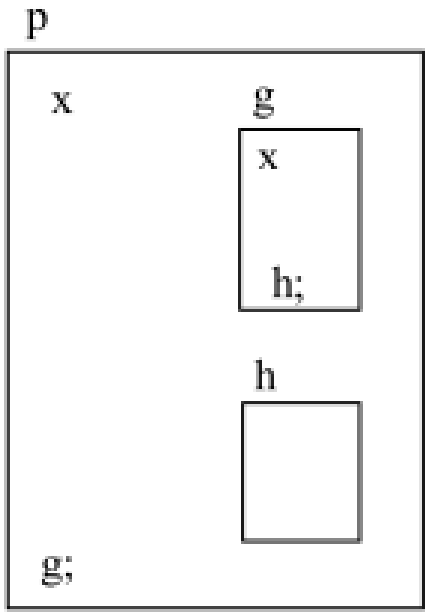
حوزه ایستا و پویا

- **حوزه پویای** وابستگی مربوط به یک شناسه مجموعه ای از سابقه های فعالیت زیربرنامه است که وابستگی در حین اجرا قابل مشاهده است. در حوزه پویا برای پیدا کردن وابستگی یک شناسه (اسم) باید در زمان اجرا رکورد فعالیت جاری جستجو شده و در صورت عدم پیدا شدن وابستگی ، رکوردهای فعالیت زنجیره پویا به ترتیب جستجو شوند تا وابستگی یافت شود.
- **حوزه ایستا**: اغلب فرآیندها یکبار در زمان ترجمه انجام شوند. در حوزه ایستا رکوردهای فعالیت زیر برنامه های در برگیرنده زیر برنامه جاری در ساختار برنامه جستجو می شوند.

صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

- قاعده حوزه پویا :متغیر به نزدیک ترین تابعی در زنجیره فراخوانی ها ارجاع می شود.
- مثال حوزه پویا: اگر تابع f تابع g را صدا بزند و g هم تابع h را صدا بزند و متغیر x هم در f و هم در g تعریف شده باشد ولی در h تعریف نشده باشد ، هنگام استفاده از x در h ، کنترل به x موجود در g ارجاع می شود



صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

- قاعده حوزه ایستا:

به طور کلی قاعده حوزه ایستا، ارجاع‌ها را به اعلان اسامی در متن برنامه مربوط می‌کند ولی قاعده حوزه پویا ارجاع‌ها را با وابستگی‌های اسامی (فراخوانی‌ها) در حین اجرای برنامه ربط می‌دهد. همواره ایده آل 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 برقراری این حالت مشکل است.

برخی از مزایای حوزه ایستا عبارتند از: الف- امکان بررسی کنترل نوع ایستا ب- امکان انقیاد در زمان ترجمه ج- سرعت اجرای بیشتر د- هزینه کمتر ه- امکان ایجاد قوانین مشخص و واضح ساختار بلوکی

نکته: اغلب زبان‌هایی که از حوزه ایستا استفاده می کنند مانند C و پاسکال و Ada، کنترل نوع ایستا را انجام می دهند. بنابراین در این زبان ها، ساختارهای زمان اجرا ساده تر شده و برنامه سریع تر اجرا می شود.

صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

قاعده حوزه ایستا: اگر بخواهیم خروجی برنامه را در حوزه ایستا بررسی کنیم در صورت وجود نداشتن یک اسم به صورت محلی در یک بلاک، باید به بلاک‌های بیرونی تر آن مراجعه کرده و از مقادیر آنها استفاده کرد.

قاعده حوزه پویا: اگر بخواهیم خروجی برنامه را در حوزه پویا بررسی کنیم در صورت وجود نداشتن یک اسم به صورت محلی در یک بلاک، باید سراغ بلاکی برویم که زیر برنامه را فراخوانی کرده است (قاعده تازه ترین وابستگی)

نکته: در دامنه پویا عمل وابستگی اسامی متغیرها به اشیا با توجه به سلسله مراتب فراخوانی روال‌ها (معنایی) و در زمان اجرا صورت می‌گیرد. در دامنه ایستا عمل وابستگی اسامی متغیرها به اشیا با توجه به تودرتویی تعریف روال‌ها (نحوی) و در زمان ترجمه انجام می‌شود.

صفات کنترل داده ها(ادامه)

```
Procedure A( )
      Var   x: integer;
      Procedure   B ( )
      Begin
            X: =x+1;
            Write(x);
      End;
      Procedure C ( )
            Var x: integer;
      Begin
            X: =30;
            B ( );
      End;
Begin
      X:=7;
      B ( );
      C ( );
End;
```

حوزه ایستا و پویا (ادامه)

حوزه پویا:

مرتبه اول B ( ) توسط A صدا زده می شود پس x درون B ( ) به x درون A ارجاع می شود. مرتبه دوم( ) B توسط C( ) صدا زده می شود پس x درون B ( ) به x درون C ( ) ارجاع داده می شود. بنابراین خروجی ۳۱ و ۸ خواهد بود.

حوزه ایستا:

در این برنامه اگر از حوزه ارجاعی ایستا استفاده شود هنگام صدا زدن B( ) چون زیر برنامه B( ) متغیر محلی x را ندارد، از متغیر محلی x مربوط به زیربرنامه دربر گیرنده آن یعنی A( ) استفاده کرده و خروجی آن برابر ۷+۱ یعنی ۸ می شود. سپس با صدا زدن زیر برنامه C( )، داخل این زیر برنامه یک متغیر محلی X با مقدار ۳۰ ساخته می‌شودکه ربطی به X موجود در A( ) ندارد. حال پس از صدا زدن B( ) درون زیر برنامه C( )، دوباره زیر برنامه B( ) از X مربوط به A( ) استفاده کرده و خروجی آن ۹=۸+۱ می شود. بنابراین، خروجی نهایی ۹ و ۸ می‌شود.

صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

- **پیاده سازی حوزه ایستا و پویا**

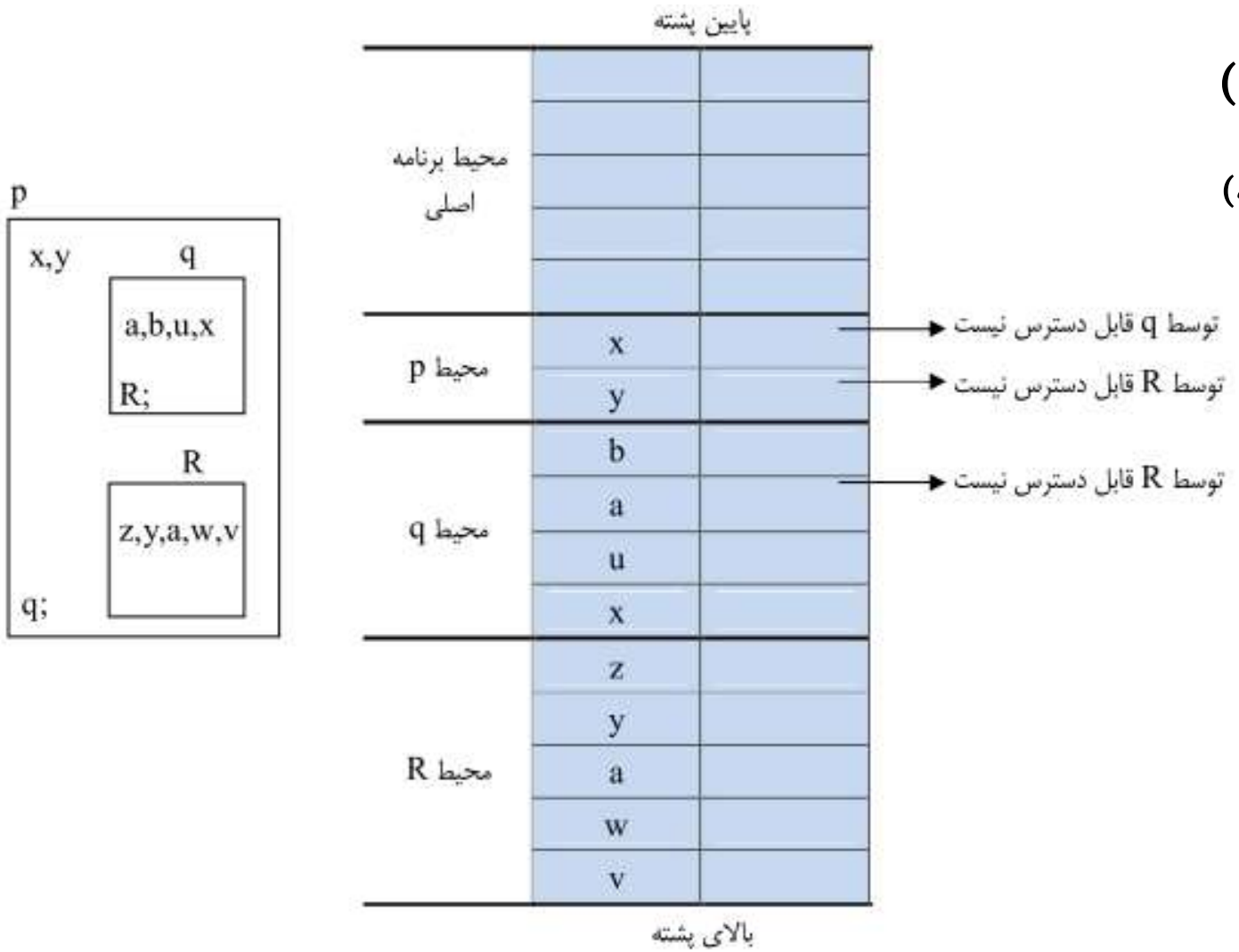
پیاده سازی حوزه پویا:

یکی از روش‌های پیاده سازی حوزه پویا، استفاده از پشته مرکزی است. فرض کنید زیر برنامه p، زیر برنامه q را صدا بزند و زیر برنامه q نیز زیر برنامه R را صدا بزند (p→q→R) . هنگامی که R اجرا می‌شود پشته مرکزی به شکل زیر خواهد بود. جهت پردازش ارجاع غیر محلی x، پشته با شروع از محیط محلی R به طرف عقب جستجو می‌شود تا تازه ترین وابستگی برای x پیدا شود. در این حال، برخی از وابستگی‌های موجود در پشته با وابستگی‌های بعدی برای همان شناسه، مخفی می شوند(مانند x,y,a که برای زیر برنامه R مخفی می شود).

صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

- پیاده سازی حوزه پویا (ادامه)



SCP: Static Chain Pointer
DCP: Dynamic Chain Pointer

صفات کنترل داده ها(ادامه)

حوزه ایستا و پویا (ادامه)

پیاده سازی حوزه ایستا

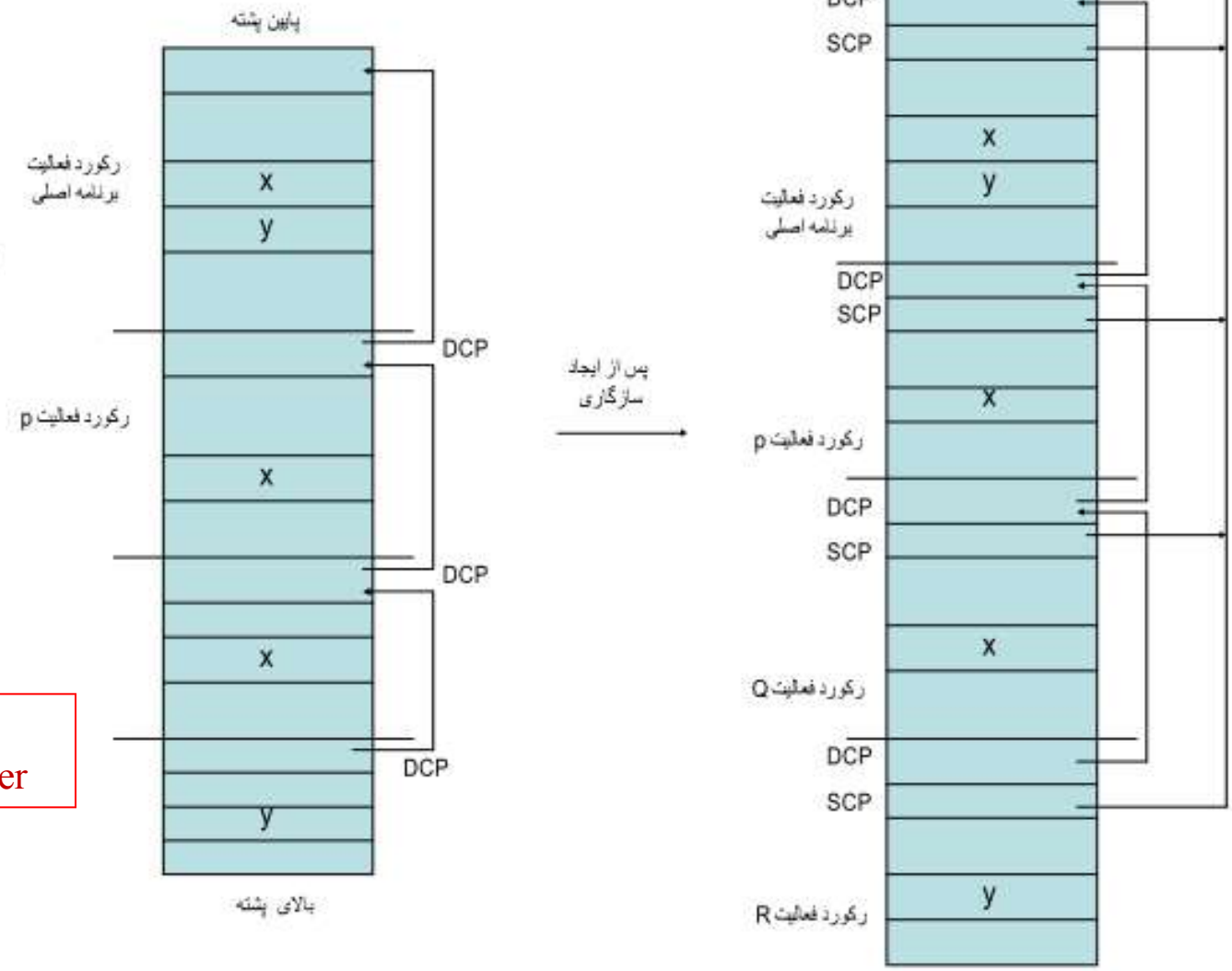
در زبان‌هایی مانند پاسکال و Ada که از ساختار بلوکی استفاده می‌کنند پردازش ارجاع‌های غیر محلی پیچیده تر است. در شکل صفحه بعد که نمونه ای از قواعد حوزه ایستا را برای برنامه ساخت یافته بلوکی در پاسکال نشان می‌دهد زیربرنامه R از Q فراخوانی می‌شود و زیر برنامه Q نیز از P فراخوانی می‌شود. زیربرنامه‌های P و Q و main متغیر x را تعریف می‌کنند. در داخل R، x به طور غیر محلی ارجاع می‌شود. طبق قاعده حوزه پویا، هنگام اجرای R، متغیر x باید متغیر تعریف شده در Q باشد در حالیکه طبق قاعده حوزه ایستا متغیر x به x موجود در برنامه main اشاره دارد و باید هنگام اجرای دستور x=x+1 از آن استفاده شود لذا سازگاری بین قواعد حوزه ایستا و پویا لازم است.

برای ایجاد سازگاری، باید حوزه ایستا در زمان اجرا کنترل شود و در هر رکورد فعالیت یک اشاره گر زنجیره ایستا (SCP) ذخیره شود که آدرس رکورد فعالیت در برگیرنده را نگهداری می کند. از این طریق می‌توان زنجیره ایستای برنامه را دنبال کرد. در این مثال، به هنگام اجرای R، با توجه به SCP که به رکورد فعالیت برنامه main اشاره می‌کند و با استفاده از یک آفست (تفاوت مکان)، متغیر x موجود در برنامه main در اختیار x قرار می‌گیرد. در روش حوزه ایستا، از اشاره گر زنجیره ایستا (SCP) و در روش حوزه پویا، از اشاره گر زنجیره پویا (DCP) برای پیدا کردن وابستگی استفاده می شود. اشاره گر زنجیره ایستا (SCP)، آدرس رکورد فعالیت زیر برنامه سطح بالاتر (بیرونی تر) را نگهداری می کند ولی اشاره گر زنجیره پویا (DCP)، آدرس رکورد فعالیت فراخوانی شده توسط زیر برنامه جاری را نگهداری می کند. برنامه زیر را در نظر بگیرید:

پیاده سازی حوزه ایستا (ادامه)

برای درک مفاهیم زنجیره ایستا و پویا برای اجرای برنامه فوق شکل زیر را مشاهده کنید.

```
Program main;
      var x,y: integer
      procedure R
            var y: real
      begin
            x:=x+1
      end;
      procedure Q
            varx :real
      begin
            R;
      end
      Procedure P
            var x: Boolean
      begin
            Q;
      end
begin
      P;
End;
```



SCP: Static Chain Pointer
DCP: Dynamic Chain Pointer

صفات کنترل داده ها(ادامه)

ساختار بلوکی

- مفهوم ساختار بلوک در زبانهای ساخت یافته بلوگی مثل پاسکال پیدا شد.
- هر زیربرنامه یا برنامه به صورت مجموعه ای از بلوکهای تودرتو سازماندهی می شود.
- ویژگی مهم بلوک : محیط ارجاع جدیدی را معرفی میکند.
- با مجموعه ای از اعلان ها برای اسامی شروع می شود و سپس مجموعه ای از دستورات قرار می گیرد که به آن اسامی مراجعه می کنند.
- اسامی قابل دستیابی غیر محلی را به دو صورت حوزه ایستا و حوزه پویا می توان مشخص کرد.

صفات کنترل داده ها(ادامه)

ساختار بلوکی (ادامه)

- قواعد حوزه ایستا در برنامه ساختیافته بلوکی:

- اعلان‌های ابتدای هر بلوک، محیط ارجاعی محلی آن بلوک را پدید می‌آورند. اگر داخل یک بلوک، از اسمی استفاده شود که در اول آن بلوک به صورت محلی تعریف شده است کامپایلر از آن اسم محلی استفاده می‌کند و به سراغ دیگر اسامی نمی‌رود.
- اگر در بلوکی از اسمی استفاده شود که در آن به صورت محلی وجود ندارد، یک بلوک به سمت بیرون رفته و در اسامی بلوک در بر گیرنده آن دنبال آن اسم می‌گردد. اگر اسم را پیدا نکرد این عملیات را به سمت بلوکهای بیرونی تکرار می‌کند تا بالاخره آن اسم را یافته و از آن استفاده می‌کند.
- اسامی محلی موجود درون یک بلوک از دید بلوک خارجی آن مخفی است.
- بلوک می تواند دارای نام باشد. نام بلوک بخشی از محیط ارجاع محلی در برگیرنده آن محسوب می شود.

صفات کنترل داده ها(ادامه)

داده های محلی و محیطهای ارجاع محلی

- داده های ساده ترین ساختار را دارند.
- محیط محلی زیر برنامه شامل پارامترهای مجازی و متغیرهای محلی آن زیر برنامه می باشند.
- متغیر محلی در زیر برنامه sub2 عبارتست از d و متغیر محلی در زیر برنامه sub1 عبارتست از b

```
Procedure sub1(a:real);
      Var b:real;
       Procedure sub2(c:real);
            Var d:real;
             Begin
                   c:=c+b;
             End;
      Begin
           ....
       End;
```

صفات کنترل داده ها(ادامه)

داده های محلی و محیطهای ارجاع محلی(ادامه)

پیاده سازی: وابستگی متغیرهای محلی به دو روش deletion و Retention می باشد

- روش Retention(نگهداری):

در این روش، در اولین ورود به زیربرنامه متغیر تعریف و اجرا می شود. در موقع بازگشت از زیر برنامه متغیر از بین نمی رود و در فراخوانی‌های بعدی از آخرین مقدار متغیر استفاده می شود. کوبول و فرترن از روش نگهداری استفاده می کنند. در پیاده سازی این روش، محل نگهداری اشیاء داده ای (جدول محیط محلی L.E.T) در کد سگمنت برنامه می باشد.

- روش Deletion(حذف):

در این روش متغیر در اولین ورود به زیربرنامه ایجاد می شود و به هنگام بازگشت از بین می رود. زبان‌های Ada,Lisp,C,Pascal,SNOBOL4,APL,Java,C++ از روش حذف استفاده می کنند. در پیاده سازی این روش محل نگهداری اشیا داده (جدول محیط محلی L.E.T) در رکورد فعالیت زیربرنامه (حافظه پشته) می باشد.

صفات کنترل داده ها(ادامه)

```
Procedure Q;
     Var x:integer:=30;
      Begin
          Write(x);
           X:=x+1;
      End;
Procedure P;
     Begin
         Q;
         Q;
         Q;
     End;
```

داده های محلی و محیطهای ارجاع محلی(ادامه)

مثال:

۱. نگهداری: اگر وابستگی x نگهداری شود مقدار اولیه x:=30 فقط بار اول صدا زدن Q انجام می گیرد و پس از خروج از Q مقدار x حفظ می شود. بار اول که Q داخل p صدا زده می شود، مقدار اولیه ۳۰ چاپ شده و x برابر ۳۱ می شود. بار دوم که Q صدا زده می شود مقدار قبلی ۳۱ چاپ شده و x برابر ۳۲ می شود و بار سوم این مقدار ۳۲ چاپ می شود. بنابراین، خروجی نهایی ۳۲ و۳۱ و۳۰ می باشد.

۲. حذف: در این روش در هر بار خروج از Q، x حذف شده و در هر بار ورود به Q یک x جدید با مقدار اولیه ۳۰ ساخته می شود. لذا در این حالت خروجی نهایی برنامه ۳۰ و ۳۰ و۳۰ می شود.

صفات کنترل داده ها(ادامه)

داده های محلی و محیطهای ارجاع محلی(ادامه)

پیاده سازی روش های حذف و نگهداری

- محیط محلی هر زیر برنامه در یک جدول محیط محلی (Local Environment Table (LET)) شامل نام ، نوع و ... ذخیره می شود.

در روش نگهداری،L.E.T در سگمنت کد تشکیل می‌شود چون سگمنت کد به طور ایستا تخصیص می‌یابد و در حین اجرا باقی می‌ماند. هر متغیر موجود در بخش محیط محلی سگمنت کد نگهداری می‌شود درحالیکه در روش حذفL.E.T به عنوان قسمتی از رکورد فعالیت آن زیربرنامه می‌باشد و در پشته مرکزی تشکیل می‌شود. چون رکورد فعالیت یک زیربرنامه به هنگام ورود به زیر برنامه در پشته مرکزی ایجاد می‌شود و با خروج از زیر برنامه از بین می‌رود. بنابراین برای زیر برنامه‌های برگشتی روش حذف بسیار مناسب می‌باشد چون روش نگهداری فضای زیادی تلف می‌کند.

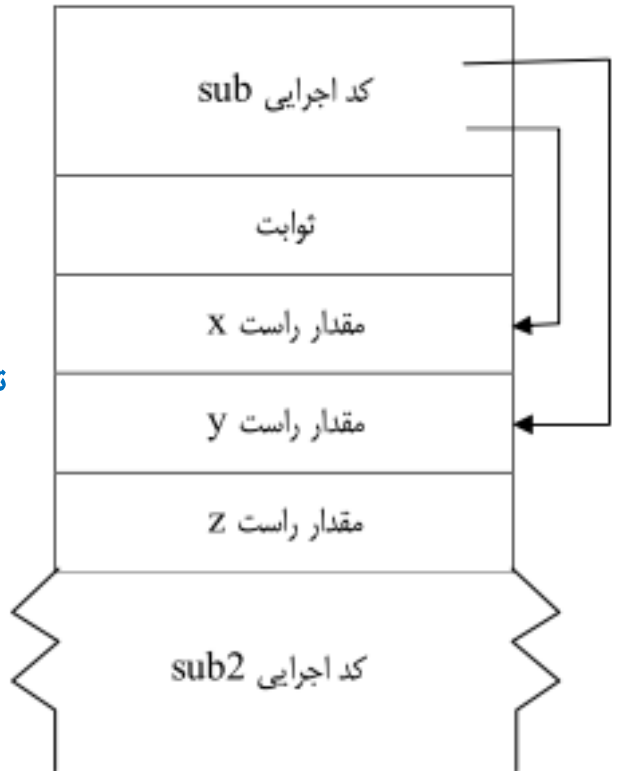
پیاده سازی روش های حذف و نگهداری (ادامه)

```
Procedure sub (x:integer) is
      y:real;
      z:array (1..3) of real;
      procedure sub2 is
      begin
            …
      end {sub2};
begin
      …
end {sub};
```

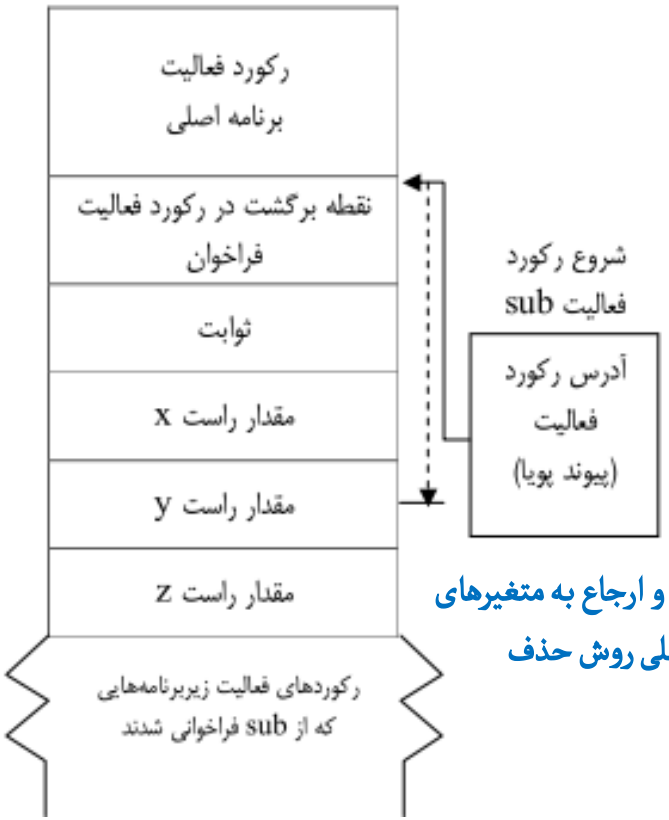
LET برای sub در زمان ترجمه

نام	نوع	محتویات Lvalue
x	Integer	پارامتر با مقدار
y	Real	متغیر محلی
z	Real	آرایه
		توصیفگر : [1..3]
Sub2	Procedure	اشاره گر به سگمنت کد

سگمنت کد مربوط به sub به طور ایستا اختصاص می‌یابد



تخصیص و ارجاع به متغیرهای محلی روش نگهداری



تخصیص و ارجاع به متغیرهای محلی روش حذف

پارامترها و انتقال پارامترها

گاهی اوقات مواردی پیش می آید که در حین اجرای زیربرنامه ها ، بایستی یک سری اطلاعات بین زیر برنامه ها به اشتراک گذاشته شود.

چهار روش اصلی برای اشتراک:

- **محیطهای اشتراک صریح** : داده اشتراکی صراحتا مشخص می شود مثل کلاس ها در C++
- **محیطهای اشتراک ضمنی**: داده ها بین زیر برنامه ها مشترک هستند ولی تعریف صریحی از آنها انجام نمی شود. مثل import کردن UNIT در پاسکال
- **قواعد حوزه پویا و ایستا**
- **وراثت**: انتقال اطلاعات بین زیر برنامه ها ، از طریق قرار دادن آنها در یک سری پارامترها و فرستادن آنها به زیر برنامه فراخوان امکان پذیر است

پارامترها و انتقال پارامترها(ادامه)

پارامترهای مجازی و واقعی

به پارامترهای زیر برنامه در هنگام تعریف آن پارامترهای مجازی یا رسمی (فرمال) گفته می‌شود. به پارامترهای زیر برنامه هنگام فراخوانی، پارامترهای واقعی می‌گویند. مثلاً در تکه برنامه زیر a,b پارامترهای مجازی و x,y پارامترهای واقعی هستند.

```
Int fn(int a,int b){
      Return a+b;
}
Main( ){
   Int x=5,y=6,z;
   Z=fn(x,y);
}
```

پارامترها و انتقال پارامترها (ادامه)

پارامترهای مجازی و واقعی (ادامه)

تناظر بین پارامترهای مجازی و واقعی

- **تناظرموقعیتی:** دراین تناظر که در اکثر زبان‌ها استفاده می‌شود، اولین پارامتر واقعی به اولین پارامتر مجازی، دومین پارامتر واقعی به دومین پارامتر مجازی و ...نگاشت می‌شود. زبان‌های C و پاسکال از این روش استفاده می کنند.
- **تناظر براساس نام:** در برخی از زبان‌ها مانند Ada از تناظر نام استفاده می‌شود یعنی پارامترها براساس نام نگاشت می شوند و ترتیب آنها مهم نیست مثلا دستور فراخوانی زیر در زبان Ada را در نظر بگیرید:

```
Sub(x,y) {      تعریف
        ...
        ...
}
Sub  ( y=>27 , x=>B )  فراخوانی
```

در حین فراخوانیSub، پارامتر واقعی B با پارامتر مجازی x و پارامتر واقعی ۲۷ با پارامتر مجازی y متناظر می‌شوند.

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

توضیح چهارروش متداول :

- فراخوانی با مقدار (Call by value)
- فراخوانی با ارجاع (Call by refrence)
- فراخوانی با نام (Call by name)
- فراخوانی با نتیجه (Call by result)
- فراخوانی با مقدار- نتیجه (Call by value-result)
- فراخوانی با مقدار ثابت (Call by const)

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با مقدار

در حالت معمولی در زبان‌هایی مانند C و پاسکال پارامترها به صورت فراخوانی با مقدار به زیربرنامه فرستاده می‌شوند، در این روش در هنگام فراخوانی مقادیر (مقدار راست) پارامترهای واقعی در پارامترهای مجازی(رسمی) کپی می‌شوند ولی در هنگام برگشت از زیر برنامه فراخوانی شده، مقادیر پارامترهای رسمی هیچ تاثیری روی پارامترهای واقعی ندارند.

```
Void fn( int x, int y){
      x=0; y=0 ;
      Printf ("%d %d", x ,y);
};
Main ( )
      Int a=1, b=3 ;
      Printf ("%d  %d" , a ,b );
      Fn (a ,b );
      Printf ("%d %d" , a, b ); }
```

قبل از فراخوانی	۱	۳
در زیر برنامه فراخوانی شده	۰	۰
بعد از فراخوانی	۱	۳

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با ارجاع

این روش، متداول ترین روش انتقال پارامترهاست. در هنگام فراخوانی آدرس پارامترهای واقعی (مقدار چپ) به زیربرنامه فرستاده می‌شود. بدین ترتیب تغییراتی که به پارامترهای مجازی، درون تابع داده می‌شود، به پارامترهای واقعی متناظر در برنامه صدا زننده اعمال می‌شود. زبان C این نوع فراخوانی را با استفاده از اشاره گرها پیاده سازی کرده است.

```
Void fn(int x , int y){
      x=-4; y=-6;
      Printf("%d %d ",x ,y); //-4     -6
}
Main(){
      Int a=1,b=3;
      Printf("%d %d",a,b);     //1        3
      Fn(&a,&b);
      Printf("%d%d,a,b); //-4-6
}
```

قبل از فراخوانی	۳ ۱
در زیر برنامه فراخوانی شده	-۶ -۴
بعد از فراخوانی	-۶ -۴

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با نام

این روش کمتر مورد استفاده زبان‌ها می‌باشد انتقال پارامتر با نام در الگول از اهمیت بالایی برخوردار است ولی به دلیل سربار اجرایی بالا، روش کاربردی و معروفی نیست. در این روش نام پارامتر واقعی جایگزین نام پارامتر مجازی در زیربرنامه فراخوانی شده می‌گردد. در این حال هر ارجاع به پارامتر مجازی مستلزم ارزیابی مجدد پارامتر واقعی متناظر با آن است. برای این کار در نقطه فراخوانی زیربرنامه، پارامترهای واقعی ارزیابی نمی‌شوند تا زمانی که در زیر برنامه به آنها مراجعه شود. برای پارامترهایی که از نوع متغیر ساده هستند مانند int، فراخوانی با نام از دید برنامه نویس از نظر نتیجه خروجی معادل فراخوانی با ارجاع است ولی اگر از نوع متغیر ساده نباشند ممکن است نتایج با هم فرق کند.

مثال: در procedure sub(x:integer) اگر بخواهیم فراخوانی sub(y) را در متن برنامه اصلی داشته باشیم می‌توان اینگونه فرض کرد که در بدنه روال sub به جای همه xها، y قرار داده می‌شود حال در هر مراجعه به y یک ارزیابی مجدد از y صورت می‌گیرد از نظر کاربر نتایج فراخوانی با نام و فراخوانی با ارجاع یکی است ولی نتایج میانی متفاوتی از هر دو دیده می‌شود.

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با نام (ادامه)

```
var  i:integer;
Function     f(x:integer):integer;
Begin
      i=2;
       x=1;
      i=3;
       x=3;
End;
Begin
      a:array [1..3] of integer={1,2,3};
      i=1;
       print(a[1],a[2],a[3])     //output:1,2,3
      f(a[i]);
       print(a[1],a[2],a[3])     //output:1,1,3
end;
```

```
function  f(a[i]);
begin
i=2;
a[i]=1;
i=3;
a[i]=3;
end;
```

با فراخوانی تابع f، نام پارامتر a[i] جایگزین نام پارامتر مجازی x در زیر برنامه f می‌شود سپس با هر بار رجوع به x مقدار a[i] محاسبه می‌شود.

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با نتیجه

پارامتری که به این شیوه ارسال می‌شود فقط جهت برگرداندن نتیجه به برنامه فراخوان استفاده می‌شود. دراین روش در هنگام فراخوانی، مقادیر پارامترهای واقعی در پارامترهای رسمی کپی نمی شوند ولی در هنگام بازگشت آخرین مقادیر پارامترهای رسمی در پارامترهای واقعی کپی می شوند. به عبارتی دیگر می‌توان گفت در این نوع فراخوانی، تابع پارامتر ورودی ندارد و فقط دارای پارامتر خروجی است.

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با مقدار - نتیجه

در این روش در هنگام فراخوانی، مقادیر پارامترهای واقعی در پارامترهای مجازی کپی می‌شوند. داخل زیربرنامه هر تغییری روی پارامتر مجازی انجام گیرد روی پارامتر واقعی اثر ندارد و در هنگام بازگشت از زیربرنامه، مقادیر پارامترهای مجازی در پارامتر واقعی کپی می‌شوند. یعنی پارامتر واقعی تا زمان خاتمه زیربرنامه مقدار اصلی خودش را حفظ کرده و پس از اجرای زیر برنامه مقدار جدیدی می‌گیرد. این روش فراخوانی در زبان Algol-w استفاده شده است.

پارامترها و انتقال پارامترها(ادامه)

روشهای انتقال پارامترها

فراخوانی با مقدار ثابت

هنگامی که پارامتر به صورت مقدار ثابت یا const انتقال داده می‌شود در حین اجرای زیربرنامه نمی‌توان پارامتر مجازی را تغییر داد و این پارامتر مجازی ثابت، در حین اجرای زیربرنامه مانند یک مقدار ثابت محلی عمل می‌کند. از دید برنامه فراخوان این پارامتر ثابت، فقط یک آرگومان ورودی برای زیربرنامه است و مقدارش چه به صورت سهوی و چه به منظور برگرداندن نتیجه، قابل تغییر نیست.

```
int fn(const int a, int b)
```

پارامترها و انتقال پارامترها(ادامه)

پیاده سازی انتقال پارامتر

- چون هر رکورد فعالیت زیربرنامه، مجموعه متفاوتی از پارامترها را دریافت می‌کند حافظه پارامترهای مجازی زیر برنامه به بخشی از رکورد فعالیت زیربرنامه تخصیص می‌یابد و نه در سگمنت کد. بنابراین، اگر محل ذخیره پارامترهای رسمی به عنوان قسمتی از رکورد فعالیت زیربرنامه فراخوانی شده باشد و پارامتر رسمی به نام P از نوع T داشته باشیم یکی از دو روش زیر بسته به مکانیزم انتقال پارامتر، پیاده سازی می‌شود.
  - P به عنوان شی داده محلی از نوع T است در صورتیکه ارزش اولیه آن یک کپی ارزش پارامتر واقعی باشد.
  - P به عنوان شی داده محلی از نوع اشاره گری به T است در صورتیکه ارزش اولیه، اشاره گری به پارامتر واقعی باشد.

- روش اول برای فراخوانی مقدار–نتیجه، فراخوانی با مقدار و فراخوانی با نتیجه استفاده می‌شود روش دوم برای انتقال پارامترها از طریق ارجاع و از طریق نام استفاده می‌شود. از هر دو روش می‌توان برای پیاده سازی انتقال از طریق مقدار ثابت استفاده کرد و برگرداندن مقدار با تابع نیز با روش اول انجام می‌شود.

فصل هفتم

مدیریت حافظه

مدیریت حافظه

- با اینکه برنامه نویس با مدیریت حافظه سروکار دارد و باید برنامه هایی بنویسد که از حافظه به خوبی استفاده کند احتمالاً کنترل مستقیم او بر حافظه اندک است.

عناصری که به حافظه نیاز دارند

- سگمنت کد برای برنامه ترجمه شده کاربر
- برنامه های زمان اجرای سیستم
- ثوابت و ساختمان داده های تعریف شده توسط کاربر
- نقاط برگشت زیربرنامه ها
- محیطهای ارجاع
- حافظه های موقت در ارزیابی عبارات
- حافظه های موقت برای انتقال پارامترها
- بافرهای ورودی – خروجی
- داده های خراب سیستم
- عملیات فراخوانی و برگشت از زیربرنامه
- عملیات ایجاد و از بین بردن ساختمان داده ها
- عملیات درج و حذف اجزای ساختمان داده ها

مدیریت حافظه تحت کنترل برنامه نویس و سیستم

مشکل کنترل برنامه نویس بر روی مدیریت حافظه:

- مسئولیت سنگینی را متوجه برنامه نویس می کند و ممکن است با مدیریت حافظه تحت کنترل سیستم تداخل ایجاد کند.
- مراحل مدیریت حافظه
  - تخصیص اولیه
  - بازیابی حافظه
  - فشرده سازی و استفاده مجدد

مدیریت حافظه ایستا

- ساده ترین شکل تخصیص ، تخصیص ایستا است . این تخصیص در زمان ترجمه انجام می شود و در طول اجرا ثابت است.

مدیریت حافظه هرم

- هرم بلوکی از حافظه است که در آن قطعاتی از حافظه به روش غیرساخت یافته تخصیص می یابند و آزاد می شوند.
- تکنیکهای مدیریت حافظه هرم برحسب اینکه اندازه عناصر تخصیص یافته ثابت باشد یا متغیر به دو دسته تقسیم شوند.

مدیریت حافظه هرم (ادامه)

مدیریت حافظه هرم با عناصر طول ثابت

- برای تخصیص یک عنصر اولین عنصر لیست فضای آزاد از لیست حذف می شود و اشاره گری به آن عنصر به عملیات درخواست کننده حافظه برگردانده می شود.
- بازیابی: شمارش ارجاعها و زباله روبی
  - برنامه نویس یا سیستم آنها را بر می گرداند.
  - شمارش ارجاع
  - زباله روبی
    - نشانه گذاری
    - جاروکردن

مدیریت حافظه هرم (ادامه)

مدیریت حافظه هرم با عناصر طول ثابت (ادامه)

- بخش نشانه گذاری زباله روب کار دشواری است .
- سه فرضیه در مورد این فرآیند نشانه گذاری وجود دارد:
  - هرعنصر فعال باید از طریق زنجیره ای از اشاره گرها با شروع از خارج هرم قابل دستیابی باشد.
  - باید بتوان اشاره گرهای خارج از هرم را که به عنصری در هرم اشاره می کند تعیین کرد
  - باید بتوان در داخل هر عنصر فعال هرم فیلدهایی را تعیین کرد که حاوی اشاره گرهایی به عناصر دیگر هرم اند.

مدیریت حافظه هرم(ادامه)

- مدیریت حافظه هرم با عناصر طول متغیر

تخصیص اولیه و استفاده مجدد

به دلیل متغیربودن طول عناصر دو امکان برای استفاده مجدد وجود دارد:

- استفاده از لیست فضای آزاد برای تخصیص حافظه
- با انتقال تمام عناصر فعال به یک طرف هرم فضای آزاد لیست را فشرده کنید.

مدیریت حافظه هرم(ادامه)

- مدیریت حافظه هرم با عناصر طول متغیر

استفاده مجدد از لیست فضای آزاد

برای مدیریت تخصیص مستقیم از این نوع لیست فضای آزاد چند تکنیک وجود دارد:

- روش اولین جای مناسب
- روش بهترین جای مناسب

مدیریت حافظه هرم(ادامه)

- مدیریت حافظه هرم با عناصر طول متغیر

بازیابی با بلوکهای طول متغیر

- روش برگشت صریح فضای آزاد شده به لیست فضای آزاد ساده ترین تکنیک است اما مسئله های مربوط به زباله ها و ارجاعهای معلق وجود دارد.
- باید تشخیص دهیم که انتهای یک عنصر کجاست وعنصریعدی از کجا شروع می شود.
- ساده ترین راه حل این است که در کنار بیت زباله روبی در اولین کلمه هر بلوک طول آن بلوک نگهداری شود.

مدیریت حافظه هرم(ادامه)

- مدیریت حافظه هرم با عناصر طول متغیر

فشرده سازی و پراکندگی حافظه

- دو روش برای فشرده سازی وجود دارد:
  - فشرده سازی جزئی
  - فشرده سازی کامل