Subject:
Year. Month. Date. ( )

جلسه چهارم: 7,14

تعریف نحو (Syntax):

مجموعه‌ای که ترکیبات کلمات کنار هم که معنی داشته باشد ← Syntax

ترتیب عمل کننده باعث ایجاد برنامه valid بشود.

عناصر نحوی زبان

کاراکترها

ASCII
[ ] 8bit

0-127 → حروف انگلیسی
128-255 → زبان توسعه یافته (extended)

الان برای رفع مشکل برای ایجاد حالت استاندارد برای تمام زبان‌ها به جای 8 بیت از 16 بیت استفاده می‌کنیم

unicode
[ ] → 16bit

شناسه‌ها (identifier: ترکیبی از حروف و ارقام برای نام‌گذاری

کلمات اضافی (noise words) (برنامه‌نویس خواسته ← noise ← ناخواسته)

افزایش قابلیت انعطاف پذیری زبان.

فرمت آزاد و ثابت:

point: main frame رده‌هایی از server مانند IBM تولید کند. یک frame اصلی که بر اساس Time sharing

به ماشین‌هایی که وصل است سرویس می‌دهد.

فرمت ثابت یا آزاد ← اگر بتوانیم در هر جایی که می‌خواهیم بنویسیم و هر چقدر فاصله که می‌خواهیم بین دستورات بگذاریم ← free

در زبان‌های مثل assembly، فرمت free نداریم بلکه فرمت آزاد است.

ویژگی یک زبان خوب:

خوانایی، قابلیت نوشتاری، قابلیت بررسی، سهولت ترجمه، عدم وجود ابهام

(زبان خوب ← برنامه‌نویسی راحت است کدها را خوب می‌کنند

compiler راحت‌تر است و میزان خطا کمتر است

PAPCO

Subject:
Year. Month. Date. ( )

۱. خوانایی: درک آن برای برنامه‌نویس راحت‌تر است.

a) طول identifier نباید محدود باشد

b) keyword ها کلمه‌ها با مفهوم متداول جمله کاربردی

c) امکان وجود comment

d) تفاوت‌های نحوی منعکس‌کننده تفاوت‌های معنایی باشند

۲. قابلیت نوشتاری: بین خوانایی و نوشتاری تناقض وجود دارد خوانایی کوتاه یعنی قدر بنویسیم تا compile راحت شود

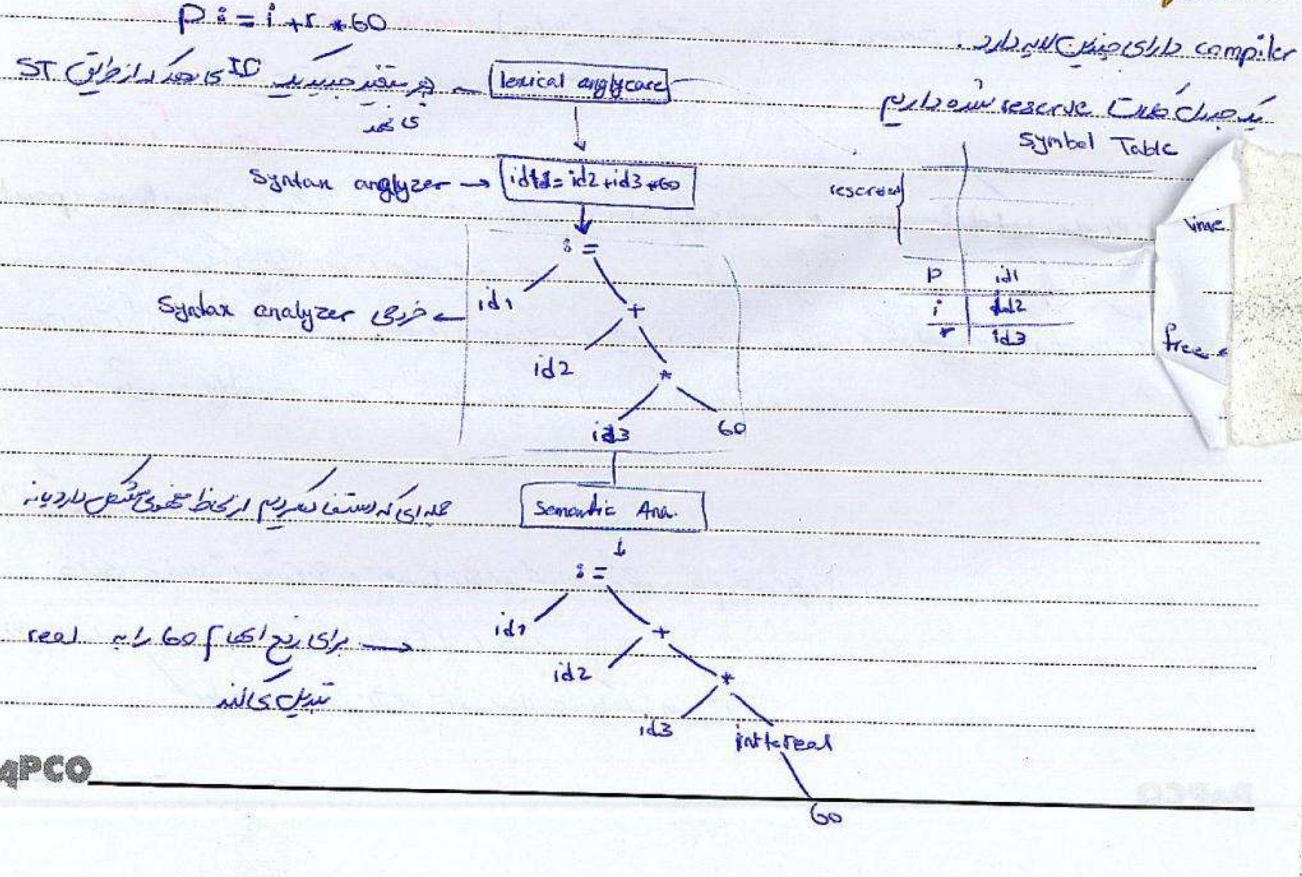
نوشتاری: به راحتی کاربر از آن بهره ببرد.

روش‌های ترکیب زیر برنامه‌ها نسبت به هم:

متداول‌ترین روش ۱ و ۲: برای برطرف کردن ایراد در ۲ باید کل برنامه را دوباره compile کنیم ولی در ۳ باید module مربوطه را compile کنیم

مراحل compile

compiler دارای چندین لایه دارد.



Subject:

Year. Month. Date. ( )

↓

Intermediate Code generation

کد را دستکاری کرده و بهینه کنند

int t_1 := inttoreal(60)

$t_2 := id_3 * t_1$

$t_3 := id_2 + t_2$

$id_3 := t_3$

↓

code optimizer

$t_1 := id_3 * 60.0$, inttoreal

$id_1 := id_2 + t_1$

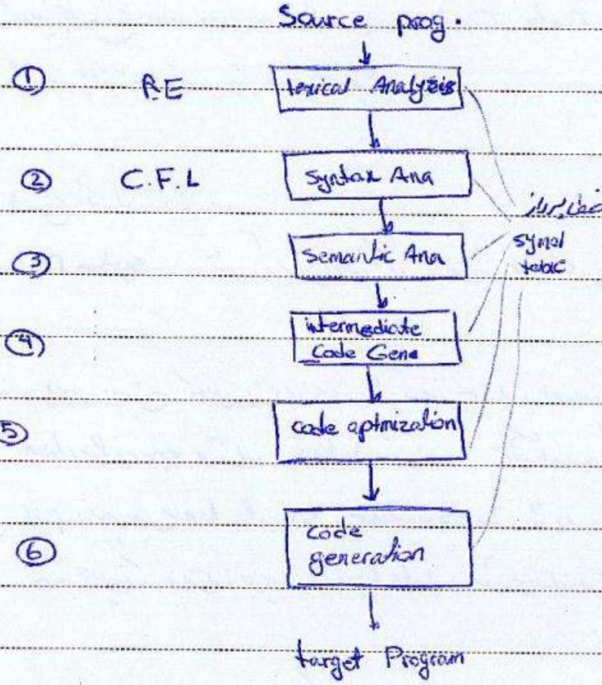
↓

code Generation

```
movF   id3, R2
MULF   #60.0, R2
movF   id2, R1
ADDF   R2, R1
movF   R1, id1
```

جلسه پنجم ۷/۱۶/

Compile:



۱. برنامه ورودی کاراکتر به کاراکتر خوانده شده و به دنباله‌ای از توکن‌ها تبدیل می‌شود (به هر توکن یک id نسبت داده می‌شود تا در بقیه مراحل بتوان به آن رجوع کرد)
کلمات کلیدی، عملگرها، جداکننده‌ها، ثابت‌ها، شناسه‌ها

2. برنامه از نظر خطاهای نحوی بررسی می‌شود و با استفاده از توکن‌ها درخت تجزیه (pars) ایجاد می‌شود

3. استفاده از درخت تجزیه مرحله قبل برای رفع خطاهای معنایی احتمالی مثل تبدیل int به real

4. تولید یک کد میانی برابر با برنامه اصلی با 2 خصوصیت.
a) تولید و تبدیل ساختار آن باید ساده باشد
b) ترجمه به زبان مقصد برای این کد باید به راحتی صورت گیرد

5. کد بهینه می‌شود تا کد میانی به نحوی بهبود یابد و مختصر شود.

6. تولید کد به زبان مقصد (مجموعه‌ای از دستورات زبان ماشین است).

خطاپرداز

error handeling

کار ایرادهای معنایی و نحوی است و compiler هرچه بیشتر error را بگیرد برنامه ما زودتر به یک برنامه error free تبدیل می‌شود که با همه مراحل در ارتباط است

symbol table

⑦ با وقوع هر خطا این مرحله فعال می‌شود (مرحله خطاپرداز) و سعی بر برطرف کردن آن خواهد کرد تا در اولین خطاها بعد compile متوقف نشود

⑧ ثبت شناسه‌های استفاده شده در منبع، جمع‌آوری اطلاعات درباره مشخصات هر شناسه

Subject:
Year. Month. Date. ( )

Data

شده نیز ارجاع به خانه های حافظه ← Data Control

شده نیز ارتباط کامپیوتر با دنیای خارج ←

انواع داده:

System Defined : منظور داده هایی است که توسط زبان برنامه سازی تعریف می شود.

12 ← برای ساختن کامپیوترهای بزرگتر [illegible] استفاده ... خیلی کم است

specification می تواند بر Implementation تاثیر بگذارد. اگر بخواهیم یک مقدار boolean تعریف کنیم این

لازم داریم بین specification و Implementation تاثیر دارد.

خصوصیتی که می تواند جزئی از datatype باشد می تواند بر پیاده سازی آن تاثیر بگذارد

15 ← Declaration ـ دستوری که در اختیار compiler قرار می گیرد.

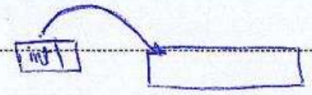
19 ← اعلان موجب پیاده سازی درست می شود storage representation →

اگر بدانیم حداکثر و حداقل مقدار متغیر در برنامه چقدر است باید در طول برنامه تغییر دهیم

21 ← بعضی جاها از برنامه نوشتیم x = 3.2 بدون هیچ اعلانی نداریم این یعنی پویا

Describtor ← اتصال نوع

مشخص کننده Type به صورت پویا



PAPCO

جلسه ششم 7,21

انتساب‌ها می‌توانند به صورت زیر باشند

A=B ← در صورتی که قبلاً برای A مقدار 16 نوشته بود الان 27 نوشته می‌شود و مقدارهایی که برای A در نظر گرفته شده بود عوض می‌شود.

وقتی در C می‌نویسیم A=B یعنی مقداری برمی‌گرداند که ممکن است sideeffect داشته باشد.

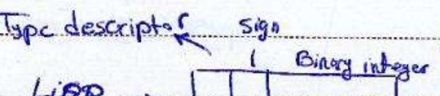
زبان‌هایی که initialization را فراهم می‌کنند بهتر است از زبان‌هایی که این را فراهم نمی‌کنند.

a=b=4 → در زبان C قابل قبول است
↘ در زبان pascal قابل قبول نیست

7 ← value ← منظور min و max مقدار است
کلاً DataType های اولیه بیشتر از یک ابزار کمک می‌گیرند
توابع ثابتی برای ← sin، cos، tan، ...
اگر چنان چیزی ضرب DataType های اولیه داشته باشیم اولین چیز یک نمایش است.

8 ←

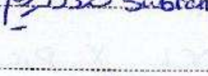
چندین روش برای نمایش Integer داریم



هنگامی که از Type descriptor استفاده می‌کنیم ← عمل جمع و ضرب و ... سخت‌افزاری انجام می‌شود
... می‌کنیم ← می‌توان شکل یکتایی به datatype ها داد

Subrange: کنترل داده توسط compiler در طول برنامه راحت‌تر است
وقتی به صورت subrange تعریف کنیم عملیات را چطور انجام دهیم ← روش سریع‌تری باشد
که بیت‌های کمتری به آن اختصاص دهیم و ... ایجاد کنیم

در پیاده‌سازی این کار را انجام می‌دهیم
↓
عملیات را تبدیل می‌کنیم تا از زبان‌های سخت‌افزاری استفاده کنیم

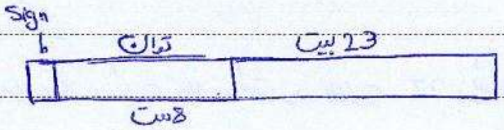


از 4 بیت استفاده می‌کنیم ← λ=1..10

Subject:
Year. Month. Date. ( )

۱ ←

روش نمایش floating point خاص است. IEEE 754

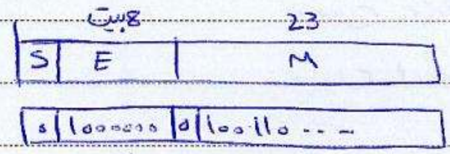


این علامت توان مورد توجه قرار نگرفته است

در slide، علامت توان هم حساب شده است.

$(4.75)_{10} = (100.11)_2$

$(-1)^S \times (0.M)_{10} \times 2^{E-127}$



آیا برای اعداد floating point عملگرهای مقایسه را به کار ببریم مثلا در حلقه استفاده می‌کنیم؟ نه

ممکن است در حلقه‌ی بی‌نهایت بار بچرخد.

یکی از دلایل این است که ممکن است ترتیب بخورند و گرد بشوند ... دچار مشکل می‌شویم و

ترتیب انجام عملیات محاسباتی روی اعداد floating point ممکن است باعث شود ترتیب های مختلفی بخورد این

شکل جواب آن یکسان نیست.

در حالت دقت مضاعف با 64 بیت نمایش می‌دهیم.

float

1. spec

1.1. attribute — دقت معمولی → 32 بیت

" مضاعف — 64 بیت

fixed point

گاهی می‌خواهیم حساب بانکی‌ها را بکنیم که این عدد dot داشته باشیم ولی نه به معنای floating point

Implied radix point → نقطه اعشاری ضمنی

DCL X Pic 999V99 → COBOL در زبان

کامپایلر مکان نقطه‌ی اعشاری ضمنی را نگه می‌دارد و محاسبات بعد از انتساب نقطه جدا می‌شود و در خروجی نمایش در

Subject:
Year. Month. Date. ( )

خروجی نقطه عوض خواهد شد.

Complex Datatype

یک عدد مختلط ← از روش نمایش تشکیل استفاده شده و همان تئوری اعداد floating point را می‌توانیم برایش حساب کنیم

قسمت حقیقی و موهومی ← می‌توانند از نحوه نمایش اعداد پیروی کنند

یک سری operation ← به‌صورت ابزاری تعریف نشدند.

norm → اندازه

Rational Datatype:

یک صورت داریم یک مخرج، می‌خواهیم اعداد چندجمله‌ای را با این روش نمایش را انجام دهیم.

$\frac{x^2+4x}{x^3+11}$ ← Rational Datatype

value ← محدودیت وجود ندارد می‌توانیم تا جایی که می‌توانیم آن را بزرگ بگیریم

نامشخص است، یعنی می‌تواند هرچی باشد.

Enumeration:

Spec 1. att. — تعداد مقادیر، خود مقادیر

2. val — صریحاً توسط برنامه‌نویس مشخص می‌شود

3. operation — مقایسه، انتساب،

Successor → بعدیش را پیدا کن

predecessor → قبلیش را پیدا کن

Implem. 1. storage rep. — نمایش بیتی رو پیدا

چون خیلی ساختمان داده آن ساده است خیلی راحت می‌توانند آن را تحت ابزاری پیاده‌سازی کرد

Subject:
Year. Month. Date. ( )

Character

Spec

1. att = کپی ندارد

2. value → EBCDIC مقادیر یا ASCII مقادیر، تمام مقادیر unicode

3. operation → مقایسه، isdigit، upper case، isletter

انتساب.

PAPCO

Subject: Data: 8,5

pointer را به عنوان داده اولیه داریم و از آن برای ساخت داده ساخت یافته استفاده می‌کنیم

pointer

Datatype به pointer ساختاری شده و اسم ندارد و آدرس هست و هر جای برنامه می‌توان datatype را تولید کرد. creation operation

برای اینکه Datatype

pointer در c و پاسکال و ... static است یعنی به بعضی از نوع‌ها می‌تواند اشاره کند pointer.type عوض کند

پیاده‌سازی: مجازی: ← محتوای آدرس واقعی حافظه را دارد

نرم‌افزاری ← آدرس activation record پیدا شدن DS هایی که برای مدیریت هستند در پشته هستند (آدرس‌هایی که به جای متغیرها را نشون می‌دند نسبت به یک آدرس base و ... آدرس دقیقاً را محاسبه کنیم

سخت‌افزاری ← جابجا کردن عنصر در حافظه باعث می‌شود که جواب بدی. مقدار قبلی خودش اشاره می‌کنیم و بعد از آن مقدارش عوض می‌شود به اینکه سرعت دسترسی بیشتری شود

File :

پیچیده‌ترین Data type

مکان آن‌ها در حافظه اصلی است ولی file در حافظه جانبی است

life time برای file خیلی بیشتر از متغیرها ... در اجرای برنامه است (تا پایان اجرای برنامه)

file ← متشکل از جنبه‌های مختلف به طور خطی چنین component دارد.

تا جایی را می‌خوانیم یا با آن کار کردیم. File position pointer

MICRO®

Subject: Data:

فایل های sequential

Master file ، file of Emp_Rec

مختص برای فایل text

عملیات روی فایل دارای اهمیت زیادی است. برای حالت read

readmode open() : اطلاعات مربوط به مشخصات file در مکان آن از O.S دریافت کرده و فضای بافر و مسیر اطلاعات فایل را اختصاص می‌دهد (file position pointer) به ابتدای فایل set می‌شود.

writemode open() : یک فایل جدید ایجاد می‌کند و file position pointer را به ابتدای آن set می‌کند

EOF می‌تواند نتایج تعریف شده پس بدهد.

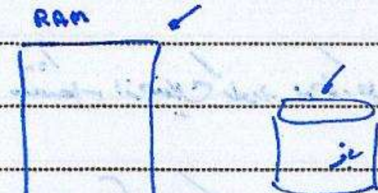
به آخر فایل رسیده یا نه Eof : file (in) → Boolean (out)

file position pointer به منظور f.p.p

close() فضای اختصاص یافته به file را آزاد می‌کند و به OS اطلاع داده می‌شود که می‌تواند از این فضا برای کارهای دیگر استفاده کند

Read() یک رکورد از بافر به داخل متغیر منتقل می‌کنیم (وقتی بافر تمام شد باعث انتقال block بعدی به بافر می‌شود)

write() انتقال متغیرهای داخل برنامه به داخل بافر (وقتی بافر پر شد در حافظه جانبی می‌نویسیم) ریختن اطلاعات یعنی مدیریت اطلاعات که در حافظه جانبی هم همه نیست و بر عهده OS است.



بین hard و Ram بافری هست که داخل Ram یا hard است وقتی پر بشود داخل hard نوشته می‌شود.

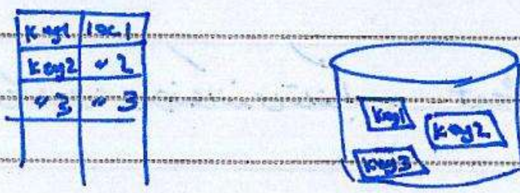
MICRO®

Subject: Data:

② فایل های Direct access

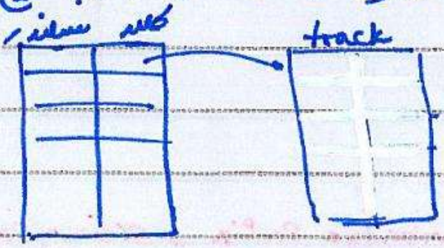
یک table داریم که به جای اینکه مکان های بعدی فایل ها را در hard پشت سر هم نداشته باشیم

در این فایل ها هر رکورد در محل متفاوتی می توانند باشند ولی در این جا یک جدول نگهداری می شود



③ ISAM (Indexed Sequential access method)

رکوردها به صورت sequential روی حافظه جانبی قرار می گیرند و یک فهرست برای بازیابی سریع ایجاد می شود را می توانیم



کنترل ترتیب اجرا:

سطوح برای Sequence Control

به برنامه نویس می توانیم Sequenc cont را تغییر دهیم

1.1 ← درج صریح توسط برنامه نویس ← صریح

1.2 خود برنامه این تعیین می کند ← ضمنی

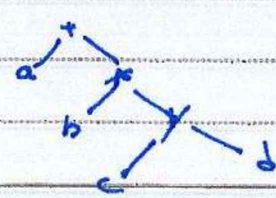
1.2 ← به وسیله برنامه نویس به دستورات مثلاً if-then-else , حلقه ها...

2.2 ← خود زبان به ترتیب فیزیکی دستورات را اجرا می کند جلوی رو

تنظیم ترتیب اجرا به صورت ضمنی توسط ترتیب فرضی تعیین می شود.

Functional compo ← ترتیب ذهنی

a+b*c/d



ترتیب اعمال عملگر برای چپ یا راست مهم است

Subject: Data:

فرض کنیم Func(C) side effect دارد.

که مقدار متغیر A تاثیر داشته باشد

فرض کنیم Func(C) دارای اولویت است یعنی به صورت ترتیب اولی

محاسبه A را تغییر دهد.

این می‌تواند باعث عوارض جانبی که اصطلاحاً side effect می‌گویند

مثال ۱)

(A+B)*(A-B)

1.1 *(+(A,B), -(A,B))

1.2 (*(+A,B), (-A,B))

1.3 * + AB - AB

2. AB + AB - *

3. (A+B) * (A-B)

روش‌های ordinary Prefix و cambridge Prefix

✓ احتیاج به پرانتزهای زیادی دارند

✓ از دید برنامه‌نویس خوانا نیستند

✓ برای عملگر با هر تعداد عملوندی قابل استفاده‌اند

✓ compiler ها به راحتی تبدیل می‌کنند به کد داخلی

✓ compiler نیازی به دانستن تعداد عملوند برای هر عملیات ندارند

✓ کامپایلر تبدیل آن‌ها به راحتی انجام می‌دهد

MICRO®

Subject: Data:

روش Post fix

✓ احتیاج به پرانتز نداریم
✓ از دید برنامه‌نویس خوانا نیست
✓ برای عملگر با هر تعداد آرگومان قابل استفاده است
✓ compiler تبدیل را به سادگی انجام می‌دهد
✓ 〃 برای محاسبات نیاز به دانستن تعداد عملوندها دارد.

روش infix

✓ احتیاج به پرانتزهای متعدد (برای رفع ابهام) : قوانین تقدم
✓ از دید برنامه‌نویس خوانایی دارد
✓ تنها برای عملگرهای Binary قابل اعمال است.

عملگرهای unary با استفاده از شیوه‌ی نمایش prefix پیاده‌سازی می‌شود.
〃 〃 مقایسه‌ای و منطقی با نمایش محدود از روش infix پیاده‌سازی شده است
و برای عملگرهای تعریف شده از سوی برنامه‌نویس از روش ordinary Prefix استفاده می‌شود.

Subject: Data:

پس از برگی نوع داده ساختیافته ← ایجاد داده که در سیستم وجود ندارد.

abstraction: ۱) تجرید (خلاصه‌سازی)

مثال ← چون در زبان assembly تعداد دستوراتش خیلی بیشتر است

Info - Hiding ← اکثری پنهان از encapsulation طرح شده ← که اجرای داریم که برای داده‌های کاری الند ری نخواهیم تغییر شخصی بکنند

11 ← Data ی داخل class را تا زمانی که برنامه‌نویس نخواهد نمی‌توانیم با آن کار کنیم

13 ← اعضای زیربرنامه ← خیلی مهم در تعریف زیربرنامه است.

14 ← ۱) زیربرنامه این خطا را اجرای تر نشان می‌دهد.

3 ← احتیاج داریم چنین چیزی را بدانیم

6 ← مثل اینکه random

20 ← دو تا متغیر داریم DS ← که خودش parameter

ثابت‌ها typedef ← در این تعریف عملیات تعریف نمی‌شوند

abstract ← operation هم تعریف می‌کنیم

Set ←

File ← در حافظه جانگه داری می‌شود | روش نمایش به روش بیشتر نیست

set ← یک لیست (مرتب / نامرتب) از component های متمایز (در مجموعه تکرار نداریم)

← ساده‌ترین نوع پیاده‌سازی دارد یعنی اینکه آرایه‌ای که عناصر را اعضا کنیم تغییری کند یا نه

اگر مجموعه خیلی زیاد حافظه صرفه‌جویی را بدارد ← (1. Bit string rep)

S = {a,b,c,d,e,f,g,h} ⇒ A = {a,b,f} ← در اینجا عناصر می‌توانند جابجا شوند

A = (1 1 0 0 0 1 0 0)

operations: 1. تست عضویت ← خیلی ساده

2. اجتماع و اشتراک ← با and و or ← اجتماع

3. حذف و اضافه به مجموعه کردن به مجموعه ← با صفر و یک کردن بیت متناظر

2. Hash coded Rep

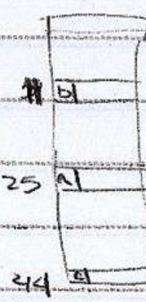
هر عضو مجموعه با یک Hash function به مکانی از حافظه map می‌شود

A = {a,b,c}

Hash(a)=25

Hash(b)= 11

Hash(C)=44



operation ها: این مزیت را دارند که عناصر لزوما دارای ترتیب نباشند و سر جای خود باشند

تست عضویت ← ساده

عملیات حذف و اضافه ← ساده

اجتماع و اشتراک ← نوشتن برنامه‌های آن از لحاظ زمانی پیچیده‌تر

Subject: Data:

Call By Value : برنامه اصلی اطلاعاتی به برنامه فرعی منتقل می‌کند ولی از فرعی به اصلی اطلاعات منتقل نمی‌شود

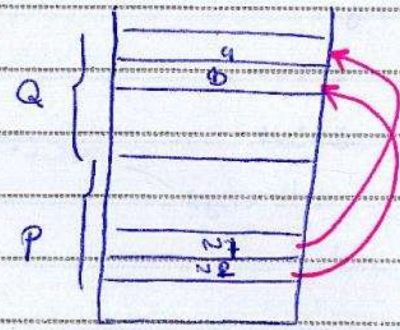
Call By Result : پارامتر به صورت خروجی است. اطلاعات از برنامه اصلی به فرعی منتقل می‌شود

تابع random ← که آنجا به ورودی نمی‌باشد

Call By Value

```
procedure Q(a, b)
begin

end
begin
    Q(N1, N2)
end
```



Call By Reference آدرس پارامتر حقیقی در پارامتر فرمال کپی می‌شود

Address (a) = Address (N1)

Address (b) = Address (N2)

Call By Value Result ← در این روش قبل از شروع مقادیر N1، N2 در a، b کپی می‌شود و پس از پایان کار Q مقادیر a، b در N1، N2 کپی می‌شود

برای زبان‌های فراخوانی شده دارای Address space متفاوت باشد

مانند RPC ← بعضی از ویرایش‌های fortran

address space ← وقتی برنامه اجرا می‌شود فضای مربوط به آن اختصاص داده می‌شود

Call By Name ←

MACRO ← برای یک کامپایلر، به جای فراخوانی macro یک سری دستورات اجرا شود یعنی این دستورها کپی می‌شود و می‌توانند پارامتر داشته باشند

Subject: Data:

پارامترهای حقیقی به جای پارامترهای فرمال در کل بدنه جایگزین می‌شود. در حالیکه نام‌های محلی

که ممکن است تغییر نام داده می‌شوند.

عملگرها هر بار به عنوان تابعی که ارزیابی می‌شود. این کد را که Thunk نام دارد

مثال تفاوت بین 5,4 :

```
Procedure R (I, J : integer)
  begin
    I = I + 1
    J = J + 1
    write (i, j)        3, C[2]+1
  end

main ( )
  m = 2
  R(M, C[M])
  print (M, C[M])
end
```

: call by reference

I, M → [2]

C[2] → [C[2]]

print (M, C[M]) → (3, C[3]-)

: call by name

دقیقاً کد را کپی می‌کنیم

3, C[3]+1 خروجی

MICRO®