أسلوب الوحدات

Modularization

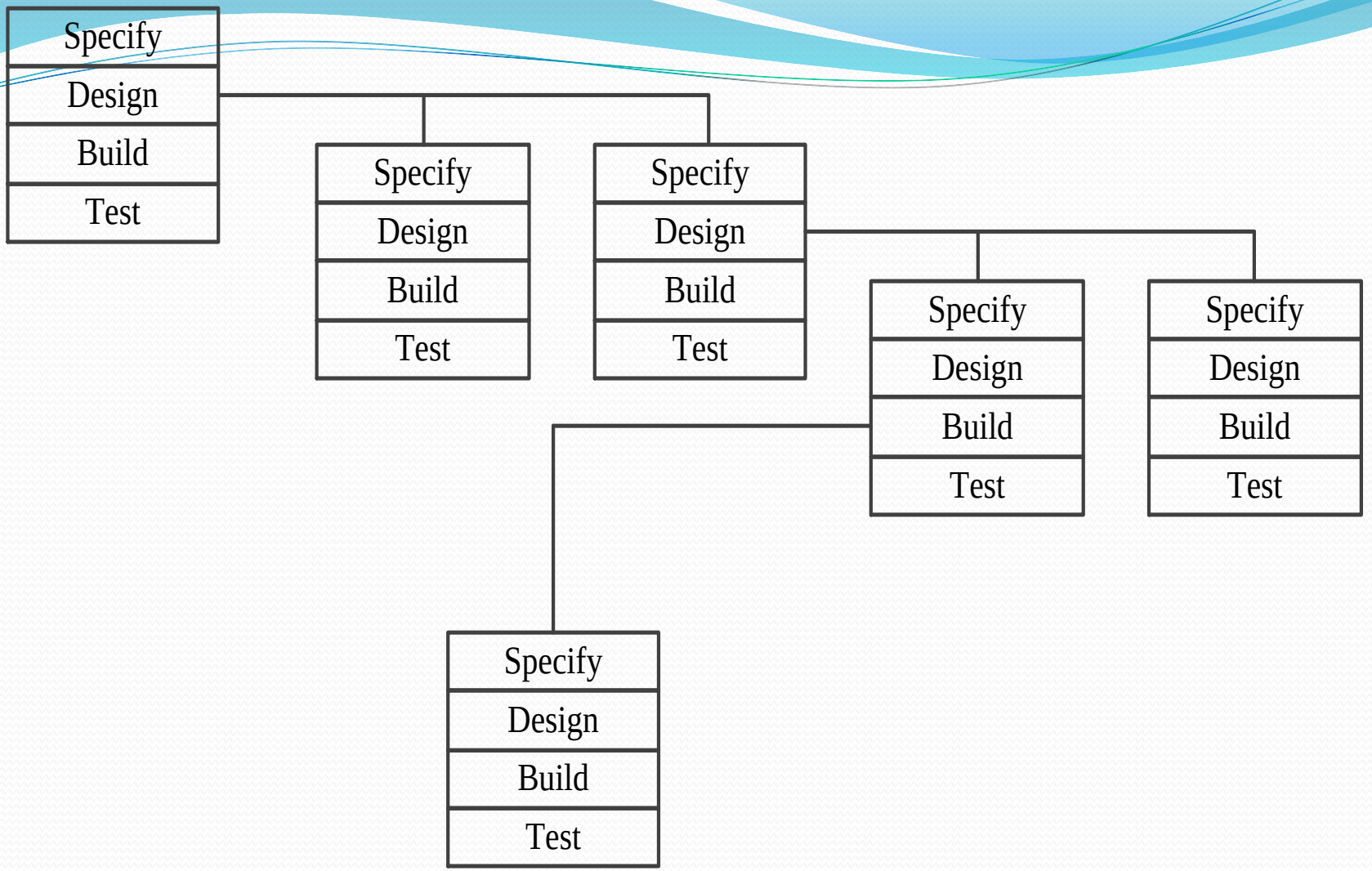
مقدمة

- في مرحلة التصميم، وعند اتباع أسلوب (من القمة إلى أسفل) Top-down (وحتى في أغلب الحالات الأخرى)، فإن أول ما يؤخذ بعين الأهمية هو التنظيم العام للبرنامج، وذلك لعلاقته بالأعمال الرئيسية التي من المزمع انجازها. والطريقة المتبعة في ذلك هي أن يتم بناء عدد من المقاطع الرئيسية المتخصصة في البرنامج. مثلاً مقطع للتهيئة (Initialization)، وللإدخال (Input)، والمعالجات (Processing)، ومقطع للمستخرجات (Output)، ومقطع يضع نهاية للبرنامج (Termination). هذا من جهة، ومن جهة أخرى، فإن معظم البرامج تضم مقاطع متخصصة للتعامل مع الحالات الاستثنائية، كمعالجة المدخلات غير الصحيحة (المكتشفة في مقطع الإدخال)، أو الإجراءات الواجب اتخاذها عند محاولة قسمة عدد ما على آخر قيمته صفر (في مقطع المعالجات).

- أما عند القيام بصياغة برنامج كبير معقد بعض الشيء، فإنه من المستحسن تقسيم البرنامج إلى عدة أجزاء، أو إلى عدة وحدات برمجية Modules. وتمتاز كل من هذه الوحدات بكونها صغيرة نسبياً ومستقلة بحد ذاتها؛ إذ أن كل منها تشكل وحدة منطقية، تقوم بإنجاز عمل واحد أو عدد محدود من الأعمال الكلية الواجب انجازها في البرنامج الكبير.
- وسنأتي في هذا الفصل على شرح التفاصيل الأساسية التي يقوم عليها أسلوب الوحدات.

تصميم الوحدات

- أن هذا الأسلوب يهتم بتقسيم مواصفات البرنامج الواحد إلى وحدات برمجية (Modules) بسيطة ومتخصصة ومستقلة. ويتم الاتصال بين هذه الوحدات المستقلة عن طريق حدود مشتركة بينها، معرفة ومحدودة بصورة جيدة. ولكون الوحدات تتمتع بخاصية الاستقلال لم يعد مهماً أن يكون العمل داخل كل وحدة معروفاً من قبل الوحدات الأخرى التي تؤلف البرنامج. إن هذا، بدوره ينعكس على التغييرات التي تحدث على التصميم. إذ غالباً ما تنحصر التغييرات في وحدة منفردة من دون أن تؤدي إلى أي تأثير جانبي قد ينتشر إلى الوحدات المجاورة.
- أن الصفات الرئيسية التي يتصف بها تصميم الوحدات هي بصورة أو بأخرى موازنة للصفات التي تنطبق على نظام المجموعة البشرية. حيث تعمل كل مجموعة من البشر بصورة مستقلة عن المجاميع الأخر، طالما أن سلسلة الأوامر نافذة المفعول. وكما أن المنظومات البشرية تكون متأثرة.



شكل رقم (1) تصميم الوحدات

- بالشخص المسؤول عنها، كذلك فإن تصميم البرامج لتي تضم عدداً من الوحدات المستقلة يعتمد على أسلوب المبرمج الذي وضع التصميم. وفي الإمكان حصر أهم الصفات التي تمتاز بها الوحدات بما يأتي:

✓ تتولى كل وحدة إنجاز مهمة منطقية واحدة، أو عدد صغير من المهمات.

✓ تكون الوحدات البرمجية Modules مستقلة، غير معتمدة على بعضها. وتمر كل وحدة في دورة خاصة بها وهي (تحديد مواصفاتها، تصميمها، بنائها، ومن ثم اختبارها) كما هو موضح في شكل رقم (1). حيث تتم صياغة كل وحدة على حدى بالاستعانة بلغة برمجة معينة، ومن ثم تعطى أسماً يشير لها وتجمع Compile وكأنها برنامج مستقل. بعد اختبارها ومعالجة كافة الأخطاء فيها، يستخدم اسم الوحدة من قبل أجزاء البرنامج الأخرى لغرض استدعائها وتنفيذها, ومن أمثلة الوحدات المستخدمة في لغات البرمجة المختلفة:

- Cobol Subprograms
- Fortran Subprograms, PL/ 1 Procedures

✓ تكون الوحدات مغلقة. أي أن لكل وحدة مدخلاً واحداً ومخرجاً واحداً.

إن التصميم الأساسي لأية وحدة Module يتضمن استقبال البيانات التي تصل إلى الوحدة، ومعاملتها بكونها مدخلات Input ومن ثم إجراء واحد أو أكثر من التحويلات عليها، بعدها يتم استخراج النتائج Output المطلوبة.

ولتوضيح ما تقدم من نقاط نستعين بالمثال الآتي:

بعد دراسة موصفات برنامج أسمه SALARY، تبين أن من ضمن الأعمال التي يقوم هذا البرنامج بإنجازها، عملية إيجاد مجموع محتويات الحقول Fields التي تكون قيد أسمه Total-Record فضلاً عن أعمال البرنامج الأخرى. وقد اتضح أن هذه العملية مستقلة ومتخصصة ومعرفة بصورة جيدة. وبالإمكان تصميم وبناء وحدة خاصة بها نسميها Sum. أن البيانات المدخلة Input لهذه الوحدة هي محتويات القيد Total-Record في حين ان النتائج المستخرجة Output ستكون عبارة عن قيمة واحدة فقط وتدون في منطقة تدعى Total-Values.

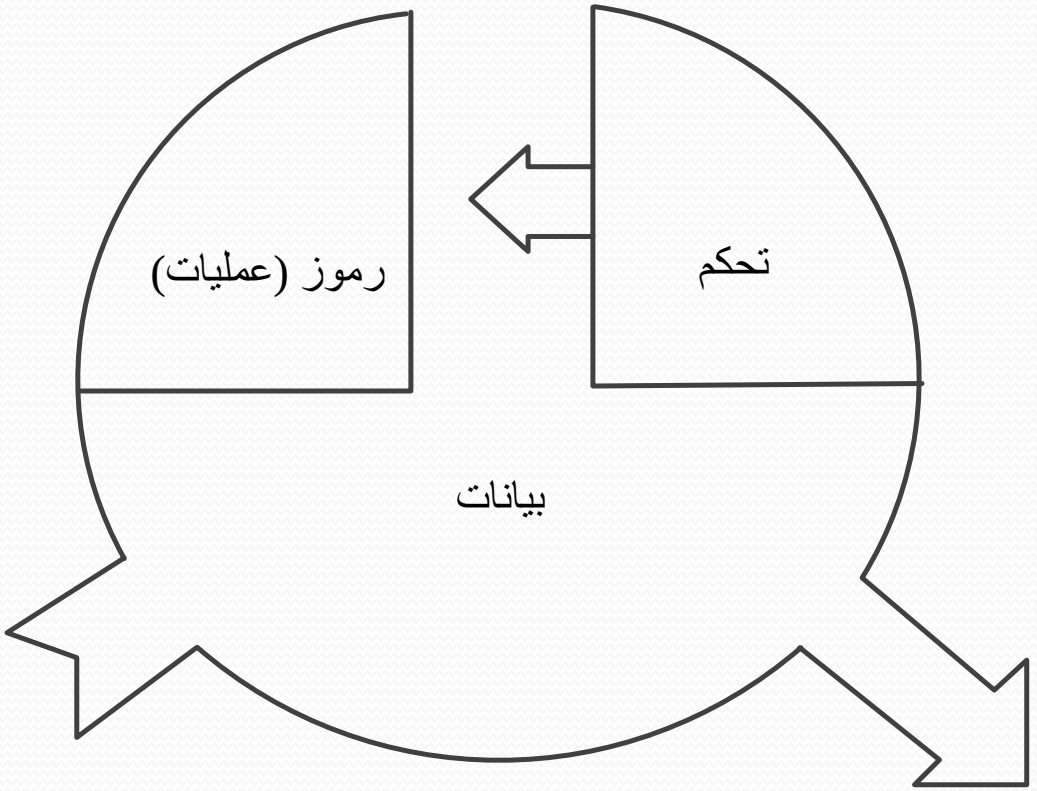
أن تصميم وحدة Sum يتم على ضوء المواصفات المتوفرة وترمز وتجمع وتفحص، بعدها تصبح جاهزة للاستعمال. وبإمكان البرنامج الأصلي SALARY أو أي وحدة أخرى أن تستدعيها، كما هو موضح في المثال الآتي باستخدام لغة كوبول Cobol:

```
CALL    Sum    Using Total-Record, Total-Values.
```

وعند تنفيذ هذا الايعاز تنجز عملية Sum.

مكونات الوحدات

بعد أن تعرفنا على الصفات الرئيسية التي تتصف بها الوحدات Modules بصورة عامة نجد أن من المهم تحديد مكونات الوحدة البرمجية على وجه الدقة. تتكون كل وحدة من ثلاثة أجزاء أساسية وهي: البيانات Data، الرموز أو (العمليات) Procedures، التحكم Control كما هو موضح في شكل رقم (2).



شكل رقم (2) مكونات الوحدة

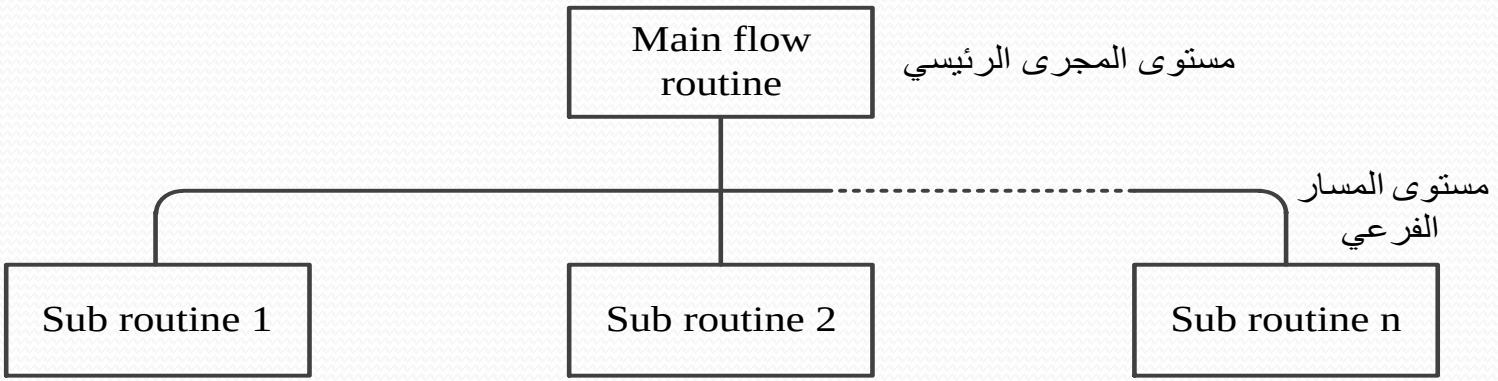
تعد البيانات مدخلات بالنسبة للوحدة Module، تتخذ بصددها بعض الإجراءات اللازمة عن طريق الرموز، ومن ثم تتركها بكونها مستخرجات. أما دور التحكم، فهو الذي يقود عملية التنفيذ. أي بمعنى آخر، يقوم باختيار العمليات التي يجب القيام بها على بيانات معينة. وتعد كل من البيانات والرموز عناصر مؤثر فيها؛ أما التحكم فهو العنصر الفعال.

الأسلوب المتبع في بناء هرمية الوحدات

هناك طريقتان مختلفتان لبناء هرمية البرامج التي تعتمد أسلوب الوحدات وهي:

1. **طريقة المجرى الرئيسي / والمسار الفرعي Main flow / Subroutine Method**

في هذه الطرية تبنى هرمية البرنامج بحيث تحتل الوحدة المسيطرة على المجرى الرئيسي قمة الهرم. وتتضمن هذه الوحدة كافة القرارات المتخذة وذات الأهمية، والتي بموجبها تستدعى المسارات الفرعية Subroutines المتخصصة لإنجاز عملية واحدة. ويمكن تمثيل ذلك في الشكل رقم (3) والذي يدعى بـ ((الهرمية التي تهبط هبوطاً مسطحاً)).

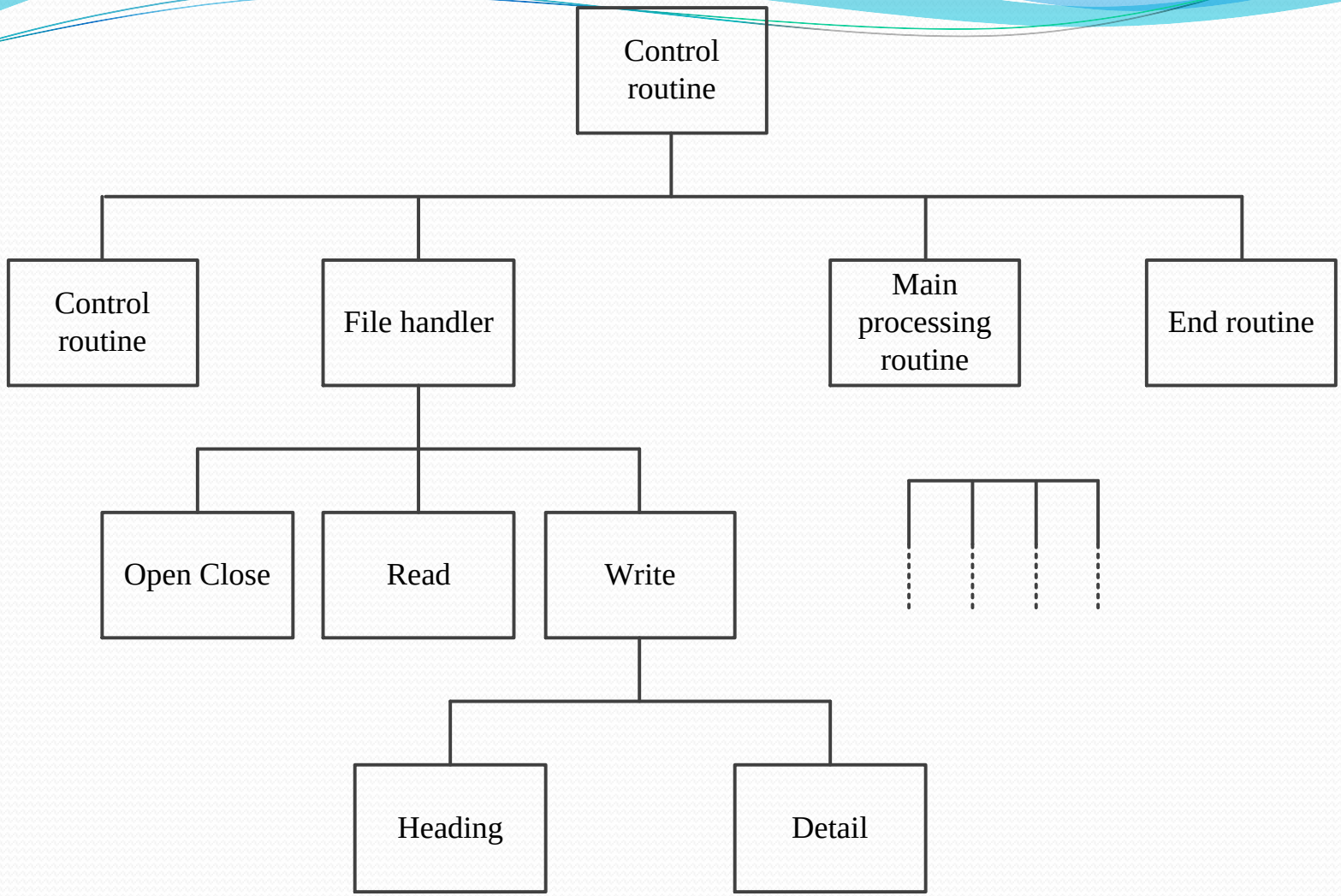


شكل رقم (3) الهرمية التي تهبط هبوطاً مسطحاً

ينطوي الهيكل الهرمي المبني بهذه الطريقة على نقطة ضعف واحدة، الا وهي التعقيد الذي تتضمنه الوحدة الرئيسية. ويعود سبب ذلك إلى كثرة عدد القرارات التي تحتويها هذه الوحدة. أن هذا من ثم يؤدي إلى صعوبة إجراء أي تغيير فيها، مع العلم أن أغلب التغييرات تتركز في هذه الوحدة. ولكن، على الرغم من وجود هذا الاختلال، يجد معظم الذين يتعاملون مع برامج تعتمد هذه الطريقة سهولة في قراءة البرامج وفي تتبعها.

2. طريقة هرمية الطبقات Layered hierarchy method

أن الهرمية الناتجة عن استخدام هذه الطريقة هي ليست بالهرمية المجردة، وإنما هي مركبة، تميل إلى تكوين طبقات حسب أهمية البيانات المستخدمة فيها، أو حسب تعقيدات الترميز. أن ما يمتاز به الهيكل المبني بهذه الطريقة والموضح في شكل رقم (4) هو سهولة إجراء التغييرات عليه. ولكن بعض المبرمجين الذين يتعاملون مع برامج تعتمد هذه الطريقة يجدون صعوبة في قراءتها وتتبعها.



شكل رقم (4) هرمية الطبقات

أن الطبقات العليا من هذه الهرمية تتضمن كل القرارات المتخذة بخصوص العمال الواجب إنجازها، أما الطبقات الدنيا فهي التي يتم فيها إنجاز الأعمال الحقيقية.

ولأجل الحصول على هيكل مبسط لكل جزء من أجزاء البرنامج، يجب الاقلال، قدر الامكان، من استخدام تعبير GO TO الذي يسهم بدوره في جعل البرنامج معقداً يصعب تتبعه. ويمكن الاستعاضة عنه بالاستعانة بتراكيب البرمجة المهيكلة الرئيسية الثلاثة، والتي سبق أن عرضت بالتفصيل ، وهي: تركيب السلسلة Sequence، والاختيار IF-THEN-ELSE ، والتكرار الدائري Loop، فضلاً عن تركيب الاختيار المتعدد CASE.

أن اعتماد تراكيب البرمجة المهيكلة في بناء البرامج يؤدي إلى نتائج قيمة وملموسة، إذ يتجلى ذلك في سهولة قراءة البرامج ومن ثم تتبعها. أن عملية تقسيم البرامج الكبيرة إلى وحدات منطقية مستقلة، يمكن تمثيلها بكتاب مقسم إلى فصول، والفصول إلى أجزاء الفصول، حيث تسهل عند ذاك قراءة الكتاب وفهمه.

من الملاحظ خلال الحياة العملية أن البرامج الأسهل تطبيقاً، والتي تتقبل إجراء تغيرات فيها بسهولة، هي التي تتألف من وحدات بسيطة مستقلة غير معتمد بعضها على البعض الآخر. ويعود السبب في ذلك إلى أن حل المشاكل المستعصية المقدمة يكون أسرع وأسهل إذا ما تمكن المبرمج من تقسيم المشاكل إلى أجزائها، ومن ثم معالجة كل جزء منها على حدة؛ بعيداً عن الأجزاء الأخر. وبالعكس تزداد الصعوبة في إيجاد حل مناسب عند محاولة النظر إلى جميع وجود المشكلة في آن واحد.

تقسيم البرامج إلى وحدات

أنه لمن المهم أن نحدد هنا الأسس التي ترتكز عليها عملية تقسيم البرامج إلى وحدات. إذ أن هناك على الأقل ثلاثة أسس رئيسية يتم بموجبها التقسيم:

1. وفق عمل البرنامج Program function.
2. وفق تسلسل الإنجاز Sequence of execution.
3. وفق هرمية مجرى التحكم في البرنامج Modularization by hierarchy of program control flow.

أن الطريقة الأولى التي تعتمد التقسيم إلى وحدات حسب عمل البرنامج ترتكز على معرفة كافة الأعمال التي يقوم بها البرنامج. ويفضل مستخدمو هذه الطريقة فصل العمليات المنجزة في البرنامج التي تنفذ أكثر من مرة واحدة في أثناء التشغيل عن غيرها من العمليات. وهناك أمثلة عديدة على عمليات من هذا النوع وهي: قراءة مجموعة من الحروف، أو التفتيش عن قيد معين، أو تدقيق التواريخ للتأكد من صحتها، أو كتابة المستخرجات أو اختبار قيم معينة.. وغيرها. وتعرف هذه الأعمال قدر المستطاع بصورة مستقلة بحيث أنها لا تتأثر بالتغييرات التي تحدث على البرنامج.

أن هذه الطريقة محبذة جداً، خاصة لمراكز الحاسبات التي تسعى إلى بناء مكتبات Subroutine Library تضم مجموعة برامج صغيرة متخصصة، يمكن الاستعانة بها عن طريق استخدامها في اكثر من نظام واحد متى ما دعت الحاجة اليها. تمتاز وحدات الأفعال المتخصصة بكونها تقدم جاهزة للمبرمج عند الطلب، إذ يأخذ بنظر الاعتبار عند بنائها مبدأ المحافظة على استقلاليتها وعدم اعتمادها على أفعال أخر قدر المستطاع.

وكلما احتاج المبرمج إلى استدعاء CALL إحدى هذه الوحدات من مكتبة البرامج المتخصصة، فما عليه إلا أن يحدد اسم الوحدة والمدخلات Input التي تحتاج اليها ومن ثم المستخرجات Output التي تنتج عنها.

ويمكن الاستعانة بالمثال الآتي لتوضيح ما تقدم:

في معظم برامج التصحيح والتدقيق Validation programs في الأنظمة المختلفة، يحتاج المبرمج إلى تدقيق حقل معين؛ ذلك هو الحقل الذي يضم التاريخ (يوم/شهر/سنة). أن التدقيق، في هذه الحالة، سيتضمن عدداً من الخطوات معرفة ومحددة وهي: يجب أن لا تقل قيمة جزء الحقل (يوم) عن (1)، ولا تزيد قيمته عن (30 أو 31) اعتماداً على الشهر المذكور في جزء الحقل (شهر). كذلك بالنسبة لجزء الحقل (شهر)، فيجب أن لا تقل قيمته عن (1) ولا تزيد عن (12). أما جزء الحقل (سنة) فيدقق لاحتمال كون السنة كبيسة أو لا، إذ أن هذا يؤثر في جزء الحقل (يوم) في حالة كون جزء الحقل (شهر) هو (2) أي شباط. فإذا تم بناء وحدة متخصصة تقوم بهذا التدقيق وأعطيت اسم (LBDATE) وخزنت في مكتبة البرامج تحت هذا الاسم، فإن استدعاءها باستخدام لغة كوبول (COBOL) سيتم كما يأتي:

```
CALL 'LABDATE' USING DATE, FLAG.
IF FLAG = 0
```

حيث أن DATE عبارة عن منطقة Area تضم التاريخ (يوم/شهر/سنة)، أما FLAG فهو مؤشر يأخذ قيمة صفر أو واحد. ويتوضح من خلال اختباره فيما إذا كان التاريخ الذي تم تدقيقه صحيحاً أو لا. أي وعلى وجه الدقة فإن DATE هي المدخلات Input التي تحتاج اليها الوحدة البرمجية LBDATE، أما المؤشر FLAG فهو المخرجات Output الذي يعكس لنا نتيجة التدقيق الذي تم في الوحدة البرمجية مفترضين أن القيمة صفر تشير إلى خطأ في التاريخ.

أن التطبيق الواعي لهذه الطريقة يؤدي إلى مرونة وفعالية، وإنتاجية أكبر بالنسبة للمبرمجين.

أما الطريقتان الأخريان فهما الأقل شيوعاً من الطريقة الأولى. فالتقسيم إلى وحدات حسب تسلسل الإنجاز Sequence of execution يعني تجزئة المشكلة إلى الأجزاء الرئيسية التي يجب أن تتم بصورة متعاقبة، ابتداءً من أول البرنامج. أن مساوئ هذه الطريقة تكمن في أنها، بالرغم من عزلها للوحدات البرمجية بعضها عن البعض الآخر، تسمح لهذه الوحدات بأن تتشاطر البيانات المشتركة؛ لذلك فإن أي تعديل على أحد البيانات قد يؤدي من ثم إلى إجراء تعديلات على أكثر من وحدة واحدة. أن الجمع بين هذه الطريقة وطريقة التقسيم إلى وحدات حسب عمل البرنامج يؤدي إلى اختزال هذه الصفة السيئة. أما الطريقة الثالثة فينطبق عليها ما تقدم أيضاً.

الفوائد الناجمة عن استخدام أسلوب تصميم الوحدات

بعد أن اسهبنا في تفاصيل تصميم الوحدات ومن ثم مكوناتها والطرق المتبعة في بنائها، نستطيع أن نستخلص ونحدد بعض الفوائد الناتجة عن استخدام هذا الأسلوب وهي:

1. كفاءة المبرمج: أن المسارات الفرعية Routines المشتركة في أكثر من برنامج واحد، يمكن برمجتها بأسلوب عام وخزنها في مكتبة البرامج. ويتمكن المبرمجون من استدعائها من مكتبة البرامج متى دعت الحاجة اليها من دون الرجوع إلى كتابتها ثانية: وتنعكس آثار ذلك في زيارة كفاءة وإنتاجية المبرمج، إذ أن الوقت الكلي المستغرق في كتابة البرنامج ستقلص نتيجة للعون الذي تقدمه مكتبة البرامج.
2. سهولة صيانة البرامج: عند إجراء بعض التغييرات والتعديلات الضرورية على برنامج مؤلف من وحدات منفصلة مستقلة، تكون هذه التغييرات متمركزة عادة في وحدة أو وحدتين فقط من البرنامج. وفي هذه الحالة، تنحصر جهود المبرمج في أماكن محددة من دون الحاجة إلى الخوض في جميع أرجاء البرنامج لإجراء التغيير اللازم فضلاً عن ذلك، فإن أجزاء البرنامج التي تكون أكثر عرضة للتغييرات من غيرها، يمكن تحديدها وفصلها وتعيين وحدات خاصة بها منذ البداية.

3. الانضباط في حل المشكلة: أن اعتماد اسلوب الوحدات في تصميم البرامج يفرض على المبرمج إجراء تحليل نظامي ومتكامل للبرنامج؛ من خلال هذا التحليل يستطيع المبرمج أن يشخص ويحدد كافة الأعمال التي يجب أن ينجزها البرنامج ويعيّن وحدات خاصة قبل البدء بترميزها.
4. أن أكثر هذه الفوائد أهمية تتجلى في سهولة حصر وتشخيص الأخطاء Debugging في البرامج ومن ثم تصحيحها. إذ أن أسلوب تصميم الوحدات يقلل الجهد المبذول لإجراء هذه العمليات، وذلك لسهولة تحديد المجال الذي قد يقع الخطأ فيه، وهذا ما يدعى بالحاجز. فالوحدات تتيح حصر الأخطاء، وحجزها في قسم واحد بحيث أنها لا تؤثر على الأقسام الأخرى من البرنامج.

حجم الوحدة البرمجية

أن حجم الوحدة له أثر بالغ في حصر الأخطاء. فكلما صغر حجم الوحدة سهلت مهمة حجز الخطأ فيها وعزله. إذ أن ايجاد خطأين متداخلين فيما بينمها أصعب بكثير من ايجاد خطأين منفصلين. أن صغر حجم الوحدة له أثر بيّن في ذلك، إذ تقل فرصة إيجاد خطأين أو أكثر، متداخلين. وعندئذ تسهل عملية تشخيص الأخطاء.

لقد دارت مناقشات كثيرة حول تحديد حجم الوحدة، وقد اقترح بعضهم المقاييس الآتية لقياس حجم الوحدة:

1. أن عدداً من مبرمجي ومستخدمي حاسبات شركة IBM يقترحون أن يتم تحديد حجم الوحدة على أساس مقدار الترميز المهيأ لشغل حجم معين من الذاكرة هو (4096) رمز Bytes.
2. وبطريقة مشابهة، فإن عدداً من مبرمجي ومستخدمي حاسبات شركة Honeywell وشركة Univac وغيرهم، اقترحوا أن تتألف الوحدة البرمجية من مجموع الترميز الذي يشغل حجماً مقداره (512) كلمة Word في الذاكرة أو مضاعفات هذا الحجم، أي (1024) كلمة، أو (2048) كلمة ... الخ.

3. أن بعضاً من مدراء (قسم البرامج) في مراكز الحاسبات يقترحون مقياساً زمنياً لتحديد حجم الوحدة. إذ تقاس الوحدة بمقدار الترميز الذي يستطيع مبرمج واحد كتابته وتشخيص الأخطاء فيه خلال مدة شهر واحد أخذين بنظر الاعتبار: أن معدل ما ينتجه المبرمج في اليوم الواحد من تعايير Statements صحيحة يتراوح بين (10-15) تعبيراً. ومحصلة ذلك تعني أن حجم الوحدة يتراوح بين (200-300) تعبير برمجي.

وهناك مقترحات عديدة أخرى، ولكن الاعتقاد السائد أن حجم الوحدة المناسب هو الذي يتراوح بين (100-200) تعبير. وفي النهاية نستطيع القول: أن حجم الوحدة البرمجية يتحدد بالعمل الذي تقوم به تلك الوحدة.

أن ما تقدم يعطي صورة واضحة عن الفوائد التي يمكن جنيها نتيجة استخدام أسلوب تصميم الوحدات. ولكن يجب أن لا يغيب عن الأذهان أن تقسيم البرنامج إلى وحدات ينطوي في جوانبه على مصدر جديد لحدوث الأخطاء ناجم عن ربط تفاعل الوحدات فيما بينها. إذ أن ربط الوحدات فيما بينها يتم عن طريق تمرير عدد من المتغيرات Arguments من وحدة إلى أخرى. ومن المحبذ أن يكون عدد المتغيرات التي تستقبلها كل وحدة صغيراً. ويفضل، بصورة عامة، كحد مشترك، أن لا يزيد عدد المتغيرات المدخلة عن سبعة.

أن المقومات الرئيسية لبناء برامج ناجحة تعتمد الوحدات تكمن في المواصفات الصارمة والمحددة للحاجز الذي يشكل حدوداً مشتركة بين أي وحدتين. ومن أجل التقدم بعملية تشخيص الأخطاء بصورة سليمة يجب أن تطبع المدخلات لكل وحدة للتأكد من أنها قرئت كما ينبغي. أن هذه المدخلات هي المتغيرات التي تمر إلى الوحدة بواسطة قائمة المتغيرات Argument List. لهذا فإن طبع هذه المتغيرات من داخل الوحدة في أثناء تشخيص الأخطاء يعطي صورة واضحة تساعد في التأكد من أن جميع القيم قد ادخلت إلى الوحدة بصورة صحيحة.

أن الخطأ المشترك والشائع الذي يقع فيه المبرمجون عند تعاملهم مع الوحدات هو تمرير قيم غير صحيحة إلى المسارات الفرعية Subroutines. ويشمل ذلك كون عدد المتغيرات التي تمر ليس هو ذاته الذي حدد في المسار الفرعي. أو أن المتغيرات لم تمرر بالصيغة الصحيحة. أو أن هناك خطأ في ترتيب أو تسلسل المتغيرات.