

بسم الله الرحمن الرحيم

جامعة السودان المفتوحة

برنامج الحاسوب

مفاهيم لغات البرمجة

Programming Language Concepts

رمز المقرر ورقمه: حسب: 4023

إعداد المادة العلمية: د. محمد الحسن مصطفى

التحكيم العلمي: د. السمانى عبد المطلب أحمد

تصميم تعليمي : أ. منال محمد بشير التنقاري

التدقيق اللغوي: أ. الهدي عبد الله محمد

التصميم الفني : منى عثمان أحمد النقة

منشورات جامعة السودان المفتوحة، الطبعة الأولى 2008

جميع الحقوق محفوظة لجامعة السودان المفتوحة، لا يجوز إعادة إنتاج أيّ جزء من هذا الكتاب، وبأيّ وجه من الوجوه، إلاّ بعد الموافقة المكتوبة من الجامعة.

مقدمة المقرر

بسم الله الذي علم بالقلم، علم الإنسان ما لم يعلم، و الصلاة والسلام على أشرف خلق الله اجمعين، الرسول الهادي وعلى اله وصحبه ومن تبعه إلى يوم الدين.

عزيزي الدارس،

يبين يدريك مقرر "مفاهيم لغات البرمجة" الهدف الاساسي من هذا المقرر هو إعطاؤك معرفة بالأدوات المهمة التي تمكنك من القدرة على تقويم لغات البرمجة المختلفة الحالية و مستقبلاً، كما أنه يهتم بالبنية العامة للغات البرمجة. و أيضاً هنالك هدف آخر الا وهو تأهيلك لدراسة تصميم و بنية المترجمات.

تم تقسيم هذا الكتاب الى عشرة وحدات، كل وحدة تتحدث عن مفهوم من مفاهيم لغات البرمجة.

يحتوي الكتاب في الوحدة الأولى على اساسيات مفاهيم لغات البرمجة، وهي تتحدث عن المفاهيم الأساسية في لغات البرمجة قبل الدخول إلى مفاهيم لغات البرمجة. الوحدة الثانية نتحدث فيها عن مراحل تطور مجموعة من لغات البرمجة، في الوحدة الثالثة نتناقش الأسماء، الربط، التدقيق النوعي و المجال. أما في الوحدة الرابعة فيتم مناقشة أنواع البيانات.

بعد ذلك نتحدث عن تصميم البرامج الفرعية وتطبيقها في لغات البرمجة و التعابير في الوحدتين الخامسة و السادسة على التوالي.

الوحدة السابعة تتحدث عن أنواع البيانات المجردة، والوحدة الثامنة تتحدث عن معالجة الاستثناءات.

و أخيراً نتحدث الوجدتان الأخيرتان عن أساليب لغات البرمجة المختلفة و هي لغات البرمجة الموجهة بالكائنات، و البرمجة الوظيفية، و البرمجة المنطقية .

الأهداف العامة للمقرر

عزيزي الدارس، عند انتهائك من دراسة هذا المقرر ينبغي أن تكون قد حققت الأهداف التالية وهي:

- التعرف على الأسباب المؤدية إلي دراسة مفاهيم لغات البرمجة.
- فهم أهمية لغات البرمجة ومستوياتها.
- التعرف على مجالات البرمجة و استخداماتها.
- التعرف على الأسماء و متطلبات تصميمها و نماذجها و الكلمات الخاصة بلغات البرمجة.
- التعرف على المتغيرات و مفهوم الربط و مدى المتغير.
- التعرف على المصفوفات و السجلات و المؤشرات و العمليات عليها.
- التعرف على أساسيات البرامج الفرعية و طرق تمرير الباراميترات.
- التعرف على التعبيرات الرياضية وطرق التعامل معها.
- التعرف على أنواع البيانات المجردة.
- التعرف على مفهوم البرمجة الموجهة للكائنات.
- التعرف على الاستثناءات.

محتويات المقرر

اسم الوحدة	رقم الصفحة
1/ أساسيات مفاهيم لغات البرمجة	1
2/ تطور لغات البرمجة	21
3/ الأسماء ،الربط، التدقيق النوعي و المجال	49
4/ أنواع البيانات	69
5/ تصميم البرامج الفرعية وتطبيقها	85
6/ التعبيرات	103
7/ أنواع البيانات المجردة	119
8/ لغات البرمجة الموجهة بالكائنات	133
9/ معالجة الاستثناءات	149
10/ أمثلة اللغات	159



محتويات الوحدة

الصفحة	الموضوع
3	المقدمة
3	تمهيد
3	أهداف الوحدة
4	1. أسباب دراسة مفاهيم لغات البرمجة.
4	2. لغات البرمجة
4	3. مستويات لغات البرمجة
5	4. مجالات البرمجة
5	1.4. التطبيقات العلمية Scientific Applications
5	2.4. التطبيقات العملية التجارية Business Applications
6	3.4. الذكاء الاصطناعي (AI) Artificial Intelligence
6	4.4. برمجة النظم System Programming
6	5.4. اللغات النصية Scripting Languages
6	6.4. اللغات الخاصة Special Purpose Languages
6	5. معايير تقييم لغات البرمجة
7	1.5. القابلية للقراءة Readability
8	2.5. القابلية للكتابة Writability
8	3.5. الاعتمادية Reliability
10	6. طرق تطبيق لغات البرمجة
13	7. بيئات لغات البرمجة
13	1.7. لغات الأوامر
14	2.7. اللغات الوظيفية
15	3.7. اللغات المنطقية
16	4.7. لغات البرمجة الموجهة بالكائنات (الشيئية)
18	الخلاصة
18	لمحة مسبقة عن الوحدة التالية
19	إجابات التدريبات
19	مسرد المصطلحات
20	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الأولى التي تأتيك بعنوان "مفاهيم لغات البرمجة"، قبل الدخول إلى مفاهيم لغات البرمجة من الضروري التعرف على بعض المبادئ. في هذه الوحدة سوف نناقش أهمية دراسة التصميم العام للغات البرمجة و مفاهيم تقييم لغات البرمجة بالنسبة لطلاب علوم الكمبيوتر و محترفي تطوير البرمجيات. من بعد ذلك سوف نتطرق باختصار إلى وصف المجالات العامة للغات البرمجة . ثم نعرض قائمة المعايير التي بواسطتها يتم الحكم على لغات البرمجة. و أيضاً تحتوي الوحدة على ملخص أهم منهجيات تطبيق لغات البرمجة. ،أخيراً سوف نقوم بوصف بعض أمثلة بيئات البرمجة وأثرها على إنتاج البرمجيات.

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تذكر الأسباب المؤدية إلى دراسة مفاهيم لغات البرمجة.
- تفهم أهمية لغات البرمجة ومستوياتها.
- تعرف مجالات البرمجة و استخداماتها.
- تشرح معايير تقييم اللغات.
- توضح طرق تطبيق اللغات.
- تعرف بيئات لغات البرمجة.



1. الأسباب المهمة المؤدية إلى دراسة مفاهيم لغات البرمجة

من الأسباب المهمة المؤدية إلى دراسة مفاهيم لغات البرمجة الآتي:

- زيادة القدرة على التعبير عن الأفكار.
- وجود خلفية متطورة لاختيار لغة البرمجة المناسبة.
- زيادة المقدرة على تعلم لغات البرمجة الجديدة.
- فهم أفضل لأهمية التطبيق.
- زيادة القدرة على تصميم لغات جديدة.
- تقدم كلي في عملية البرمجة.

2. لغات البرمجة

عزيزي الدارس، يتم تطوير برامج الحاسوب باستخدام لغات البرمجة. وتتكون لغة البرمجة من مجموعة من الرموز والقواعد كأى لغة أخرى لتوجيه العمليات في الحاسوب. وهناك العديد من لغات البرمجة المستخدمة. ويتم تصميم كل منها لحل نوع خاص من المشكلات. ومن أهم لغات البرمجة المعروفة الفورتران، كوبول، الباسكال، سي، وجافا. يمكن لأي شخص يهدف لأن يصبح مبرمجا أن يتعلم إحدى هذه اللغات ويتقنها ليستطيع بعد ذلك إعطاء أوامره للحاسوب.

3. مستويات لغات البرمجة Levels of language

هناك العديد من لغات البرمجة معظمها ينتمي إلى نفس الفئة تسمى مجموعة الفئات المختلفة للغات البرمجة بمستويات لغات البرمجة، وذلك لأنها يمكن أن ترتب بشكل هرمي، وأدنى المستويات في هذا الهرم تحتله اللغات الأقرب إلى ما يستخدمه الحاسوب، أي النظام الثنائي (0,1) والمستوى الأعلى تحتله اللغات التي تظهر مشابهاة نوعا ما للغة الإنسان مثل الإنجليزية وعند الانتقال من أدنى المستويات إلى أعلاها تظهر المستويات الآتية:

1- لغات الآلة) متدنية المستوى Machine Language

2- لغات التجميع متدنية المستوى Assembly Language

3- لغات الجيل الثالث عالية المستوى High-Level Languages

4- مولدات التطبيقات عالية المستوى Application Generators قبل

الانتقال للحديث عن مستويات لغات البرمجة لابد من التأكيد من أن لغة الحاسوب تتكون من مجموعة صفوف من 0,1 وهي اللغة الوحيدة التي يفهمها وتسمى اللغة الثنائية أو لغة الآلة.



- (1) وضح الأسباب المؤدية لدراسة مفاهيم لغات البرمجة.
- (2) اذكر مستويات لغات البرمجة.

4. مجالات البرمجة

اليوم يتم استخدام الحاسوب و تطبيقاته في مجالات عديدة، مثل: مشاريع الطاقة النووية، و حفظ سجلات البيانات الشخصية... إلخ. لذا لقد تم تصميم لغات البرمجة بأهداف مختلفة لتلائم كثيراً من الاستخدامات في المجالات المختلفة.

في هذا الدرس

هنا سوف نقوم باستعراض مختصر لبعض مجالات استخدام تطبيقات الحاسوب و لغاته المختلفة.

1.4. التطبيقات العلمية Scientific Applications

تعتمد التطبيقات العلمية للحاسوب استخدام هياكل بيانات Data Structures مثل المصفوفات Arrays و المحددات Matrices، و لكن تحتاج إلى عمليات حسابية معقدة مستخدمة أرقاماً ذات كسور Floating Point Numbers كما تستخدم هياكل تحكم عامة مثل الحلقات Loops و الاختيار Selections.

اخترعت لغات المستوى العالي High Level Languages لتلبي تلك الاحتياجات، و التي أهمها الكفاءة. أول هذه اللغات التي استخدمت في المجال العلمي كانت لغة الفورتران FORTRAN ثم أتت بعدها لغة 60 ALGOL. كما أنها استخدمت في مجالات أخرى أيضاً.

2.4. التطبيقات العملية التجارية Business Applications

تم تطوير حواسيب خاصة في العام 1950م لاستخدامها في المجال التجاري مع لغاتها. و كانت أول لغة من المستوى ناجحة في هذا المجال هي لغة الكوبول COBOL.

تتسم اللغات التجارية بالتالي:

- 1- قدرتها على إنتاج تقارير
- 2- استخدام طرق دقيقة لوصف و تخزين البيانات الرقمية العشرية و النصية.
- 3- قدرتها على تحديد العمليات الرياضية العشرية.

بعد التطور الذي شهدته الحواسيب في الأعوام السابقة ظهرت العديد من التطبيقات لاستخدامها في مجالات الأعمال التجارية المختلفة الصغيرة، مثل: نظم الجداول الإلكترونية Spread Sheets و نظم قواعد البيانات Database Systems.

3.4. الذكاء الاصطناعي (AI) Artificial Intelligence

الذكاء الاصطناعي من مجالات الحاسوب التي تستخدم فيها البرمجة الرمزية symbolic Computing. بمعنى أن البرمجة تعتمد على الرموز (الأسماء) وليس الأرقام. ومعالجة هذه الرموز مستخدمين هياكل بيانات من نوع القوائم المرتبطة (Linked Lists) لذا نجد أن البرمجة فيها أكثر مرونة من المجالات الأخرى. أول اللغات التي طورت في مجال الذكاء الاصطناعي هي لغة LISP ثم بعد ذلك ظهرت لغة البرولوج PROLOG.

4.4. برمجة النظم System Programming

يعرف نظام التشغيل و كل أدوات الدعم الموجودة معه ببرمجيات النظم System Programs و التي تستخدم باستمرار، لذا لا بد أن تتسم بكفاءة التنفيذ بالإضافة إلى مواصفات لغات المستوى الأدنى لتسهيل عملية كتابة برامج الواجهات الخارجية للأجهزة.

5.4. اللغات النصية Scripting Languages

تقوم اللغات باستخدام قوائم من الأوامر و التعليمات تسمى نصاً في ملف يتم تنفيذه. من أهم اللغات النصية لغة ksh. أيضاً هنالك لغة awk.

6.4. اللغات الخاصة Special Purpose Languages

هنالك مجموعة من اللغات التي تستخدم للأغراض الخاصة، مثل: لغة PRG التي تستخدم لإنتاج تقرير للأعمال التجارية، و لغة APT و التي تستخدم في ماكينات لغات البرمجة، و أيضاً لغة GPSS التي تستخدم في مجال نظم المحاكاة.

5. معايير تقييم اللغات Language Evaluation Criteria

1- القابلية للقراءة Readability.

2- القابلية للكتابة Writability.

3- الاعتمادية Reliability.

1.5. القابلية للقراءة Readability

سهولة قراءة وفهم البرامج (ينظر إليها من ضمن مجال المشكلة)، أي أن يكون البرنامج ينفذ باللغة المناسبة لعمله.

الميزات التي تساهم في زيادة القابلية للقراءة هي :

أ/ سهولة اللغة :

أي لغة بها مكررات كثيرة يصعب تعلمها ومن ثم قراءتها في حالة عدم الإلمام بكل تفاصيلها
مثلاً في لغة C :

$m = m+1, m += 1, m++ , ++m$

توزيع العمل operation over load

مفيدة ولكن مضرّة في حالة السماح للمبرمج بتأليف عامل غير منطقي أما السهولة فيها فتتسبب أيضاً في عدم المقدرة على قراءة البرامج.

مثلاً: Assembly

ب/ التعامد Orthogonality :

- إن عدداً قليلاً من المكونات الأساسية من الممكن أن يعطي عدداً قليلاً من الطرق للتعبير عن صيغ التحكم و البيانات.
- أي مجموعة مكونة هي مجموعة شرعية.

ب/ أوامر التحكم Control statement :

يكون البرنامج سهل القراءة إذا كان من الامكان قراءته من الأعلى إلى الأسفل بدون عمل قفزات داخله.

اللغات القديمة لم يكن بها أوامر التحكم مثل:

If , while ,

وكانت تستخدم أمر goto للتحكم في س"مفاهيم لغات البرمجة"،ير البرنامج.

د/ أنواع البيانات و هياكلها Data Type & structures :

يجب أن يكون باللغة وسائل لتعريف أنواع بيانات وهياكل بيانات مما يسهل القراءة مثلاً

الحالتن التاليتين :

Employee – absent = 1

أم

Employee – absent = true

2.5. القابلية للكتابة Writability

إن سهولة استخدام اللغة لكتابة البرامج يتأثر بالقابلية للقراءة.
الميزات التي تساهم في زيادة القابلية للكتابة :-

أ/ **سهولة والتعامد : Simplicity & Orthogonality**

إن كثرة المكونات تعوق التحكم، مما يعني صعوبة بناء البرامج وصعوبة الاستفادة من مميزات اللغة .

ب/ **دعم تجريد البيانات : Support for Abstraction**

التجريد يعني القدرة على التصميم و الاستخدام مع درجات عالية من الاستخدام.
التجريد يمكننا من جمع خصائص مفردة للفئة في صفة واحدة عامة تؤخذ في الاعتبار.
داخل المجموعة، الخصائص التي تعرف العناصر مفردة هي التي تؤخذ في الاعتبار، مما يؤدي إلى تبسيط مهم جداً للعناصر داخل المجموعة .
يستخدم التجريد لتبسيط تعقيدات عمليات البرمجة.
هنالك نوعان من التجريد هما تجريد العمليات وتجريد البيانات.

د/ **البلاغة : Expressivity**

يجب أن تكون اللغة طرق للتعبير عن العمليات الحسابية بطريقة سهلة و غير معقدة مثلاً:

count ++

بدلاً عن

count = count +/

3.5. الاعتمادية Reliability

يعتبر البرنامج معتمداً إذا كان ينفذ تبعاً لمواصفاته تحت كل الظروف
المميزات التي تؤثر على اعتمادية البرامج :

أ- **اختبار النوع : Type Checking**

الاختبار عن وجود أخطاء في استخدام أنواع اللغة المعنية في زمن الترجمة أو زمن التشغيل.
اللغات التي لديها اختبار النوع هي أقوى.

ب- **معالجة الاستثناءات : Exception Handling**

وهو إمكانية البرنامج في التعرف على أخطاء زمن التنفيذ و زمن التشغيل وحالات التنفيذ الشاذة والتصرف حيالها مما يزيد من الاعتمادية .

هنالك مجموعة من اللغات تدعم معالجة الاستثناءات مثل لغات Ada , C++ , Java

ج- الأسماء البديلة Aliasinh

وجود أكثر من اسم لنفس الموقع بالذاكرة وتعتبر خاصية خطيرة وتقلل من الاعتمادية

د- القابلية للقراءة والقابلية للكتابة

كلما سهلت كتابة البرنامج كان أكثر صحة.

كلما صعبت قراءة البرنامج صعبت كتابته وصيانتة .

هـ- تركيب اللغة : Syntax considerations

تركيب اللغة يؤثر تأثيراً مباشراً على قابليتها للقراءة .

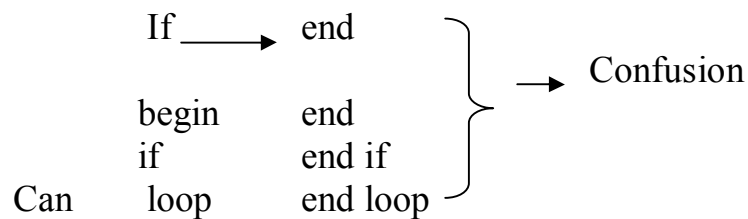
و- تركيب المتغيرات

الطول المسموح به يؤثر في إمكانية استخدام أسماء ذات معنى مفهوم

ز- الكلمات المحجوزة Reserved word

تؤثر الكلمات المحجوزة في شكلها ، و كيفية استخدامها داخل الأوامر المركبة في قابلية

القراءة للغات



أيضاً إمكانية استخدام الكلمات المحجوزة كأسماء للمتغيرات مثلاً AND

أسئلة التقويم الذاتي

1) استعرض باختصار بعض مجالات استخدام تطبيقات الحاسوب

ولغاته المختلفة.

2) وضح مع الشرح معايير تقييم اللغات.



6. طرق تطبيق اللغات Implementation Methods

عزيزي الدارس، تعتبر الذاكرة من الأجزاء الرئيسية في الحاسوب، حيث يتم تخزين البيانات. أما المعالج وهو عبارة عن مجموعة من الدوائر تقوم بمجموعة من العمليات الأساسية أو تعليمات الآلة مثل: العمليات الحسابية، والعمليات المنطقية بواسطة تعليمات صفرية تسمى لغة الآلة machine language .

يحتوى الحاسوب بالإضافة إلى لغة الآلة على مجموعة كبيرة ومهمة من البرمجيات تسمى نظام التشغيل operating system الذى يقوم بالمهام التالية:

مهام نظام التشغيل:

- إدارة موارد الحاسوب.
- عمليات الإدخال والإخراج.
- إدارة نظام الملفات.
- محرر النصوص والبرامج.
- الوظائف العامة الأخرى.

يتكامل نظام التشغيل مع نظام تطبيق البرمجة مكوناً ما يسمى بالحاسوب الوهمي . مثال ذلك يكون نظام التشغيل مع لغة C بـ كمبيوتر C الوهمي .

يتم تطبيق اللغة في الحاسوب بوحدة من ثلاث طرق وهي التالية:

- الترجمة compilation حيث يتم استخدام مترجم Compiler.
- التفسير Interpretation حيث يتم استخدام مفسر Interpreter.
- الهجين Hybrid

الآن سوف نستعرض عمليات التطبيق ببعض التفصيل في القسم التالي:

1- عملية الترجمة Compilation:

يقوم المترجم بترجمة البرنامج المصدري إلى لغة الآلة .

البرنامج المصدري source program

البرنامج المصدري: هو البرنامج الذي يكتب بواسطة المستخدم في لغة برمجة معينة.

- تتم عملية الترجمة في ثلاث مراحل هي :

أ/ أولاً :

يقوم محلل الرموز بتجميع الحروف في وحدة الرموز

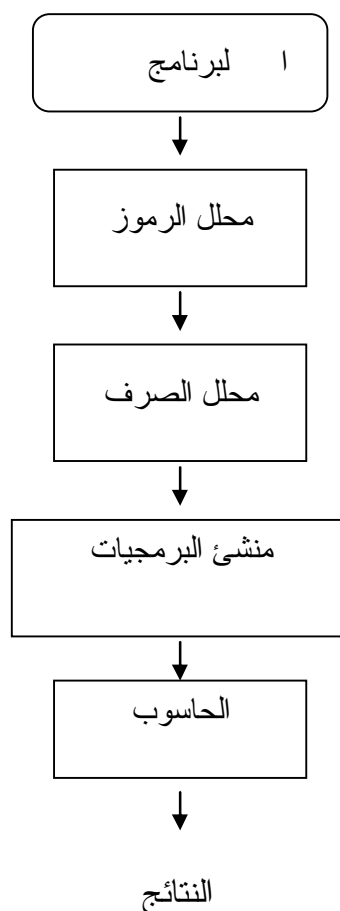
ب/ ثانياً :

يقوم محلل الصرف بأخذ وحدة الرموز من محلل الرموز ويقوم بإنشاء شكل هيكلي يسمى شجرة الهيكل وهي عبارة عن الهيكل الصرفي للبرنامج.

ج- ثالثاً :

يقوم منشئ البرمجيات بترجمة البرمجيات الوسطية إلى لغة الآلة المكافئة.

الشكل التالي يوضح كيفية عمل المترجم في الحاسوب

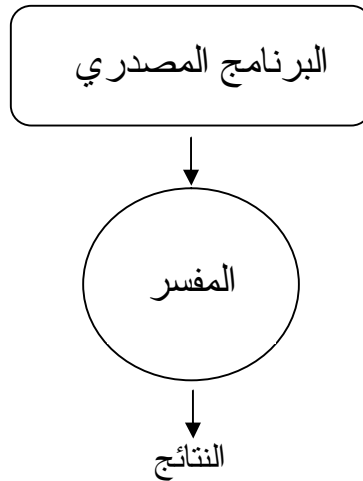


شكل 1.1 عملية الترجمة في الحاسوب

2- التفسير Interpretation :

يتم تفسير البرامج بواسطة المفسر Interpreter بدون ترجمة ، حيث يعمل المفسر كبرنامج محاكاة للآلة التي تتعامل مع برامج لغات البرمجة العالية، وليس تعليمات الآلة . يقوم المفسر بعملية تصحيح للأخطاء التي توجد في البرامج، ولكن يعمل ببطء شديد مقارنة مع المترجم لأنه يقوم بعملية فك تشفير لصيغ البرنامج . بالإضافة إلى أنه يحتاج إلى مساحة إضافية في الذاكرة .

يوضح الشكل التالي عملية التفسير في الحاسوب



شكل 1.2 عملية التفسير في الحاسوب

الفروقات بين المترجم و المفسر

عزيزي الدارس، المترجم أو المفسر عبارة عن برنامج يحول البرنامج المصدري (Source code) المكتوب بلغة عالية المستوى إلى البرنامج الهدي (Object-code) المكتوب بلغة الآلة.

الفرق بين المترجم والمفسر:

يشبه عمل المترجم ترجمة كتاب كامل من لغة إلى أخرى، بينما يشبه عمل المفسر ترجمة حوار بين شخصين عن طريق شخص ثالث يلعب دور المترجم، وبالتالي يمكن استنتاج الفرق بين المترجم والمفسر كما يلي:

المترجم يترجم جميع برنامج المستوى العالي مرة واحدة فقط . حيث ينتج برنامجاً تنفيذياً كاملاً. بينما يقوم المفسر بترجمة وتنفيذ جملة واحدة في الوقت الواحد بمجرد إدخالها إلى الحاسوب.

المفسر يوفر بيئة مناسبة للبرمجة لأنه أكثر مرونة.

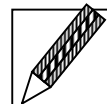
المفسر ينفذ بصورة ابطأ ويأخذ حيزاً أكبر في الذاكرة الرئيسية من المترجم.

وفي بعض لغات البرمجة عالية المستوى يتم استخدام كلا المترجم والمفسر مثل لغة البيسك.

تدريب (1)

إذا علمت أن لغة C++ تقوم بعملها بواسطة مترجم Compiler ، ما الذي يحدث إذا أردنا أن ننفذ الشفرة التالية المكتوبة بلغة C++

```
# include <iostream.h>
Int x
```



7. بيئات لغات البرمجة

عزيزي الدارس، هناك من يقسم لغات البرمجة إلى لغات عالية المستوى و أخرى منخفضة المستوى ،و منهم من يقسمها إلى :لغة الآلة،واللغات العالية المستوى ، ولغة التجميع.

و هناك أيضاً من يقسمها حسب البيئات التي تعمل عليها إلى:

1- لغات الأوامر Imperative Programming Languages .

2- اللغات الوظيفية Functional Programming Languages .

3- اللغات المنطقية Logic Programming Languages .

4- لغات البرمجة الموجهة بالكائنات (الشيئية) Object-oriented Programming Languages .

1.7. لغات الأوامر

أشهر لغات البرمجة التي تم تطويرها خلال الخمسين عاماً المنصرمة صممت لتعمل على كمبيوترات الـ *architecture von Neumann* حيث يتم تخزين البيانات و البرامج في الذاكرة نفسها أما وحدة الحساب و المنطق CPU التي تقوم بتنفيذ التعليمات تكون منفصلة تماماً عن الذاكرة. هذه اللغات تتعامل مع المتغيرات التي تُخزن في خلايا الذاكرة ،و جمل المساواة التي تعتمد على عملية التوصيل بين الذاكرة و وحدة الحساب و المنطق ،و جمل التكرار.

عملية التكرار هنا سريعة لأن التعليمات تُخزن في خلايا متجاورة من الذاكرة ،وهي الخاصية نفسها التي تحد من استخدام النداء الذاتي *recursion* للتكرار في لغات الأوامر. للحصول على نتيجة حسبة معينة يتم نقل المتغير من الذاكرة إلى وحدة الحساب و المنطق ثم يتم نقل ناتج الحسبة من وحدة الحساب و المنطق مرة أخرى إلى الذاكرة (المتغير الموجود في الطرف الأيسر من المساواة).برمجة لغات الأوامر هي *procedural programming*.

* البرمجة الشيئية و لغات الأوامر

إن أشهر لغات البرمجة الشيئية ما هي إلا لغات أوامر في أصلها، وبالرغم من أن طريقة لغات البرمجة الشيئية تختلف تماماً عن طريقة اللغات الإجرائية المستخدمة في لغات الأوامر إلا أن تطوير لغات الأوامر إلى لغات برمجة شيئية ليس بالأمر المؤرق.

* اللغات المرئية و لغات الأوامر

اللغات المرئية ما هي إلا جزء متفرع من لغات الأوامر .

أشهر هذه اللغات هي الـ Visual BASIC (1999) والتي أُستبدلت سنة 2002 بـ Visual BASIC.NET.

هذه اللغات هي تابعة للجيل الذي كان يُطلق عليه *الجيل الرابع*.

من أشهر ميزات هذه اللغات (المرئية) توفير واجهة برنامج رسومية يسهل للمستخدم التعامل معها مثل الفيجوال بيسك.

2.7. اللغات الوظيفية

هي اللغات التي تتبع القواعد الرياضية .الهدف من تصميم اللغات الوظيفية تقليد الدوال الرياضية بأكبر قدر ممكن. هذا الهدف هو ما جعلها تختلف تماماً عن لغات الأوامر في طريقة تعاملها مع المشكلات و حلها. في لغات الأوامر نحتاج لتخزين النتائج المؤقتة لأي تعبير رياضي أو لأي حسبة في مكان ما في الذاكرة، هذا المكان هو المتغير الذي تتدرج تسميته و اختيار نوعه تحت مهام المبرمج .مثلاً: لحل التعبير الرياضي:

$$(res = (x+y)/(a-b))$$

نحتاج لمتغيرات مؤقتة يُخزن فيها ناتج الجميع أولاً $(x+y)$ و متغير آخر يُخزن فيه ناتج الطرح $(a-b)$ ثم يُخزن ناتج القسمة مباشرة في المتغير الذي قام المبرمج باختياره ليحوي ناتج العملية الحسابية res في التعبير السابق.

ملاحظات حول البرمجة الوظيفية:

- في اللغات الوظيفية البحتة لا يوجد أي استخدام للمتغيرات أو جمل المساواة مما يعطي حرية أكبر للمبرمج بحيث لا يهتم للأماكن المحجوزة لبرنامج في الذاكرة.
- يتم التحكم في مسار تنفيذ البرنامج باستخدام الدوال الرياضية و الجمل الشرطية و النداء الذاتي للدوال بدلاً من تنفيذه بطريقة متسلسلة أو باستخدام التكرار كما هو الحال مع لغات الأوامر.
- تُعطي اللغات نفس النتيجة إذا أُعطيت نفس المتغيرات ، هذه الخاصية هي ما تُطلق عليها الشفافية المرجعية ، أي أنها ليست تأثيرات جانبية .
- بالرغم من أن تنفيذ اللغات الوظيفية يتم بواسطة المفسر إلا أنه يمكن عمل ترجمة لها. كذلك تحوي هذه اللغات تركيباً هيكلياً بسيطاً جداً.
- الدوال المعقدة يتم بناؤها باستخدام ما يُطلق عليه النموذج الوظيفي. أي أن الدوال تستخدم كمتغيرات مرسلّة أو قيم مرجعة أو كلاهما.
- بالرغم من أسبقية اللغات الوظيفية البحتة على تلك المستخدمة كلغات أوامر إلا أن ضعف تنفيذها على أجهزة von Neumann حد من استخدامها و انتشارها.

أشهر هذه اللغات و أقدمها هي لغة LISP.

3.7. اللغات المنطقية

هي مثال على اللغات المعتمدة على القوانين. البرمجة المعتمدة على منطق الرموز في لغاتها هي ما يُطلق عليها البرمجة المنطقية. و اللغات المعتمدة على منطق الرموز هي ما يُطلق عليها لغات البرمجة المنطقية.

البرمجة باستخدام هذه اللغات هي برمجة لا تتبع نظاماً أو ترتيباً معيناً، ولا توضح كيف نحصل على النتيجة أو كيف نحسبها لكنها توصف شكل هذه النتيجة. أي أن نظام الكمبيوتر هو الذي يحدد خط سير تنفيذ خوارزمية البرنامج المؤدية إلى النتيجة المطلوبة .على عكس لغات الأوامر و اللغات الوظيفية حيث تكون خوارزمية البرنامج موضحة بأدق تفاصيلها، و خط سير تنفيذ الأوامر و الجمل يكون موضعاً أيضاً وفق ترتيب معين. تلك الأولى (برمجة اللغات المنطقية) هي ما يُطلق عليها البرمجة غير الإجرائية.

البرمجة غير الإجرائية:

يُعطى فيها وصف للنتيجة لكن دون أن تُعطى تفاصيل الحسبة التي أدت إلى هذه النتيجة. اللغات المنطقية أيضاً لغات وراثية (عالية المستوى) تركز على منطق الحسبة. أكثر هذه اللغات شيوعاً هي لغة Prolog .

4.7. لغات البرمجة الموجهة بالكائنات (الشيئية)

عزيزي الدارس، البرمجة الشيئية تتضمن أربعة مبادئ أساسية:

- تجريد البيانات abstract data
- التوريث inheritance.
- تعدد الصور polymorphism.
- إضافة إلى إخفاء المعلومات information hiding.

و تنقسم هذه اللغات إلى لغات بحتة مثل لغتي C++ و Smalltalk ولغات مهجنة مثل لغة Java. البرمجة الموجهة بالكائنات تسرع من إمكانية تطوير برامج جديدة، و إذا أُستخدمت بالشكل الصحيح تسهل عملية الصيانة و إعادة الاستخدام و التعديل.

و هنا سنتحدث باختصار عن المبادئ الثلاثة الأساسية،:

أ- مبدأ التجريد

يخفي تفاصيل العمليات عن العميل سواء أكان المستخدم أو أي وحدة من البرنامج ، البيانات المجردة النوع أو الفئة classes هي البيانات التي تحقق الشرطين التاليين:

- أن تكون العمليات و تعريف البيانات في نفس الوحدة البنائية مما يسهل تنظيم البرنامج في هيئة وحدات منطقية يمكن عمل ترجمة لها بشكل منفصل.
- أن يكون تمثيل كائن object من نوع ما مخفي عن وحدات البرنامج التي تستخدم نفس النوع.

تكمُن أهمية إخفاء المعلومات أي إخفاء التمثيل الفعلي للبيانات في أن العميل لن يستطيع تغيير الكائن لا عنوة و لا عن طريق الخطأ، فالطريقة الوحيدة لتغييرها هي العمليات المسموح بها في اللغة.

ب- مبدأ الوراثة

هو مبدأ قوي يمكن من إعادة استخدام البرمجيات . هنالك عدة اصطلاحات مختصة بهذا المبدأ مثل:

- الفئة Class: وهي عبارة عن تجميع لبيانات و دالات تعمل على هذه البيانات.
- الكائن Object: هو عبارة عن وحدة فردة مخصصة من الفئة.

ج- مبدأ الربط الديناميكي

هو عملية ربط (عملية تحويل العناوين الرمزية ضمن البرنامج إلى عناوين متعلقة بمواقع التخزين في الذاكرة) يحدث أثناء تنفيذ البرنامج.

أسئلة التقويم الذاتي

- 1) كيف تقسم لغات البرمجة حسب البيئات؟
- 2) اذكر المبادئ الأساسية للغات البرمجة الموجهة للكائنات.



الخلاصة

عزيزي الدارس، في هذه الوحدة تعرفنا على أهمية دراسة التصميم العام للغات البرمجة و مفاهيم تقييم لغات البرمجة بالنسبة لطلاب علوم الكمبيوتر و محترفي تطوير البرمجيات. من بعد ذلك تطرقت الوحدة إلى وصف المجالات العامة للغات البرمجة . ومن ثم استعرضت قائمة المعايير التي بواسطتها يتم الحكم على لغات البرمجة. و أيضا احتوت الوحدة على ملخص لأهم منهجيات تطبيق لغات البرمجة. ،أخيراً هنالك وصف لبعض أمثلة بيئات البرمجة.

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سوف نتعرف على المدلولات الأساسية للمتغيرات، و من ثم نستعرض طبيعة الأسماء و الكلمات الخاصة المستخدمة في لغات البرمجة. وأيضاً سوف ندرس خصائص المتغيرات من نوع وعنوان و قيمة.بالإضافة إلى الربط، و زمن الربط ثم بعد ذلك سوف نقوم بدراسة عملية التدقيق النوعي والنوع القوي و قوانين التوافق و التطابقة.

إجابات التدريبات

التدريب (1)

عند تنفيذ البرنامج يقوم المترجم Compiler بالخطوات التالية :

1. يقوم محلل الرموز بتجميع الرموز في وحدة الرموز .
2. يقوم محلل الصرف بأخذ وحدة الرموز من محلل الرموز ، ثم يقوم بإنشاء شجرة الهيكل .

يقوم منشئ البرامج بترجمة البرمجيات الوسطية إلى لغة الآلة المكافئة .

مسرد المصطلحات

الذكاء الاصطناعي (AI) Artifital Intellegence

الذكاء الاصطناعي من مجالات الحاسوب التي تستخدم فيها البرمجة الرمزية symbolic

برمجة النظم System Programming

يعرف نظام التشغيل و كل أدوات الدعم الموجودة معه ببرمجيات النظم System Programs و التي تستخدم باستمرار، لذا لا بد أن تتسم بكفاءة التنفيذ بالإضافة إلى مواصفات لغات المستوى الأدنى لتسهيل عملية كتابة برامج الواجهات الخارجية للأجهزة.

اللغات النصية Scripting Languages

تقوم اللغات باستخدام قوائم من الأوامر و التعليمات تسمى نصاً في ملف يتم تنفيذه. من أهم اللغات النصية لغة ksh. أيضاً هنالك لغة awk.

اللغات الخاصة Special Purpose Languages

هنالك مجموعة من اللغات التي تستخدم للأغراض الخاصة مثل لغة PRG التي تستخدم لإنتاج تقرير للأعمال التجارية و لغة APT و التي تستخدم في ماكينات لغات البرمجة و أيضاً لغة GPSS التي تستخدم في مجال نظم المحاكاة.

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
24	المقدمة
24	تمهيد
24	أهداف الوحدة
26	1. لغة الآلة
26	1.1. لغة بلانكالكول Plankalkul
28	2.1. لغة الآلة Pseudo codes
29	3.1. أمثلة على لغات الآلة
30	2. لغة التجميع
31	3. اللغات العليا
32	1.3. لغة الكوبول COBOL
32	2.3. لغة الفورتران FORTRAN
34	3.3. لغة البيسك BASIC LANGAUGE
34	4.3. لغة باسكال PASCAL
35	5.3. لغة سي C
35	6.3. لغة فيجوال بيسك
43	7.3. لغة آدا ADA

44	4. الجيل الرابع للغات
45	1.4. لغة الاستفسار البنوية SQL
45	5. اللغات الطبيعية Natural Language أو لغات الجيل الخامس
46	6. لغات الإنترنت
46	1.6. لغة HTML
47	الخلاصة
47	لمحة مسبقة عن الوحدة التالية
48	المصادر و المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الثانية من المقرر "مفاهيم لغات البرمجة" إن لغات البرمجة مرت بمراحل تطور عدة من لغة الآلة ثم لغة التجميع مروراً باللغات عالية المستوى إلى يومنا هذا.

سنحدث في هذه الوحدة عن مراحل تطور مجموعة من لغات البرمجة ، مستعرضين البيئة التي تم تطويرها فيها ، وسنركز على مشاركة اللغة في تطور البرمجة، بالإضافة إلى الدوافع التي أدت إلى تطوير هذه اللغة ، سنتعرض فقط إلى الخصائص الجديدة التي ظهرت مع كل لغة وليست المواصفات التفصيلية لكل لغة.

هذه الوحدة لا تحتوي على مناقشة عميقة لخصائص أي لغة أو مفاهيمها بل تتعرض بصورة مختصرة إلى هذه الخصائص، حيث تركت المناقشات التفصيلية لهذه اللغات في الوحدات القادمة .

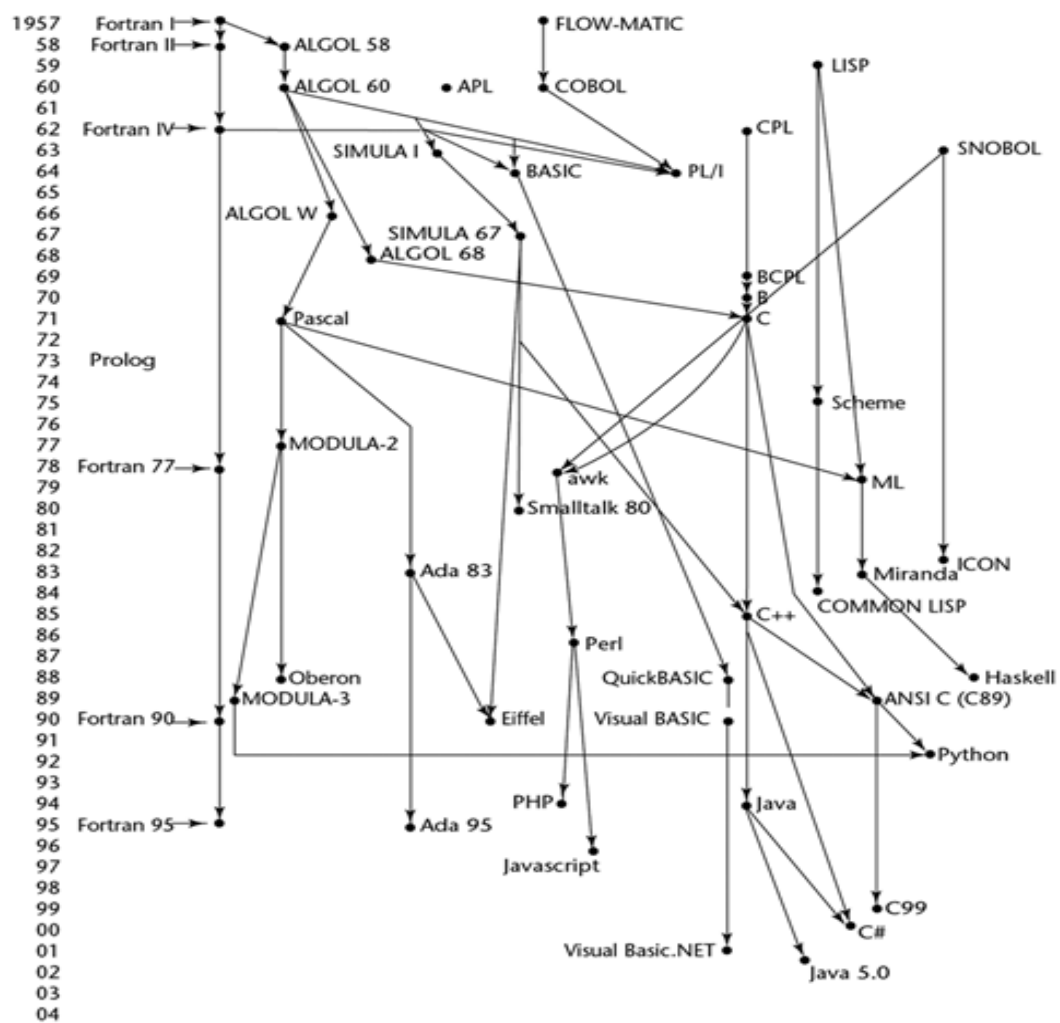
تحتوي هذه الوحدة على مجموعة من الأمثلة من اللغات المختلفة استخدمت لتوضيح ظهور البرامج في هذه اللغات، لذا لم يتم وصف هذه البرامج .

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تعرف لغة الآلة و لغة التجميع.
- تعرف لغات المستوى العالي و تذكر أكثرها استخداماً.
- تعرف لغات الجيلين الرابع و الخامس و أهم ميزاتها.
- تعرف لغات الإنترنت





Genealogy of common high-level programming languages

شكل رقم 1: الخريطة الزمنية لتطور لغات البرمجة

1. لغة الآلة

وتسمى "اللغة الثنائية" حيث إنها تتكون من سلسلة من 0 و 1، وهي اللغة الوحيد التي يفهمها الحاسب الآلي، حيث تحول جميع اللغات إلى لغة الآلة، حتى تتمكن معدات الحاسب الآلي من التفاهم معها، ولأنها تتكون من صفر وواحد، لذا فقد تميزت هذه اللغة بالصعوبة نظراً لما تتطلبه من حفظ ودقة في كتابة سلسلة طويلة من صفر وواحد بترتيب معين، مما ينتج عنه أخطاء كثيرة من الترميز، ويجب أن يحدد المبرمج كل شيء، فكل خطوة ينفذها البرنامج يجب أن ترمز، لذا فالمبرمج يجب أن يكون على علم بتركيب الحاسب الداخلي، والعناوين الرقمية لمواقع التخزين، سواء للبيانات أو التعليمات، كما أن لكل جهاز لغة آلة تختلف عن الجهاز الآخر بحسب النوع والتركيب، مما يعني أنه يجب كتابة البرنامج بشكل كامل مره أخرى عن الرغبة في تنفيذه على جهاز آخر، ونتيجة لهذه الصعوبات فقد ظهرت طرق أخرى لتمثيل الترميز الثنائي، كالنظام الثماني OCTAL والست عشري HEX-ADECIMAL بدلاً من 0، 1 فالنظام الستة عشري يتكون من ستة عشر رمزاً هي :

0، 1، 2، 3، 4، 5، 6، 7، 8، 9، A، B، C، D، E، F.

مما يساعد على سهولة قراءة التعليمات المكتوبة وحفظها بهذه اللغة، فبدلاً من كتابة 16 رقماً في سلسلة يمكن الاستعاضة عنها بأربعة رموز من رموز نظام التشغيل الست عشري.

1.1. لغة بلانكالكول Plankalkul

تعتبر هذه اللغة من اللغات غير الاعتيادية لعدة أسباب :

أولاً : هذه اللغة لم تطبق فقط في مجال الكمبيوتر.

ثانياً : بما أنها طورت منذ العام 1945م إلا أن وصفها لم يتم نشره إلا في العام 1972م، ونسبة لهذا التجاهل بعض إمكانياتها لم يظهر في اللغات الأخرى إلا بعد 15 عاماً من تطويرها .

1.1.1. خلفية تاريخية

بين العامين 1936م و 1945م قام العالم الألماني كونارد سوزو (Konard Zuse) ببناء مجموعة من الحواسيب المعقدة باستخدام ريلات كهروميكانيكية ، ولكن الحرب في العام 1945م دمرت كل النماذج التي قام بصنعها إلا واحدا وهو 24 ثم انتقل إلى قرية بافاريا لمواصلة أبحاثه لوحده بعد أن تفرق مجموعة الباحثين الذين يعملون معه .

عمل العالم تسوزو لوحده لتطوير لغة للتعبير عن الحوسبة إلى أن استطاع أن يطور لغة Plankalkul والتي تعني (حساب البرنامج) في مشروع لنيل درجة الدكتوراة حيث تمكن من تعريف اللغة وكتابة الخوارزميات لمجموعة من المشاكل

2.1.1. خصائص لغة Plankalkul

شاركت لغة Plankalkul بميزاتها المتطورة في حقل هياكل البيانات بصورة كبيرة . إن أبسط نوع من أنواع البيانات في لغة Plankalkul هو وحدة البت ومن نوع البت. تقوم Plankalkul ببناء أنواع عدد للأعداد الصحيحة (Integer) والأعداد الكسرية (Floating Points) . يستخدم نوع Floating Points طريقة البت المخفي والتطبيق الثنائي ، وأيضا تحتوي على أنواع المصفوفات . بما أن السجلات في لغة Plankalkul لا تحتوي على صيغة مباشرة للأمر Go To إلا أنها تحتوي على صيغ تكرار شبيهة ب For في لغة Pascal بالإضافة إلى ذلك تحتوي على الأمر Fin حيث يتحكم في التكرار .

تحتوي لغة Plankalkul على صيغة الاختيار ولكن لا تسمح باستخدام else .

من الخصائص المهمة في لغة Plankalkul احتوائها على تعابير رياضية توضح العلاقة بين المتغيرات في البرنامج ، حيث تنص هذه التعابير على ما هو صحيح خلال تنفيذ البرنامج في لحظة ظهورها في الشفرة.

احتوت وثائق " تسوزو " على برامج معقدة أكثر من التي كتبت قبل العام 1945م . احتوت هذه الوثائق على برامج تتعامل مع ترتيب المصفوفات العددية ، اختبار المخططات ، عمليات الأعداد الصحيحة والأعداد الكسرية ، الجذور التربيعية ، وتحليل الصيغ المنطقية التي تحتوي على أقواس .

وربما أهم ما في الوثائق هو 49 صفحة احتوت على خوارزميات لعبة الشطرنج.

المثال التالي من لغة Plankalkul يوضح صيغة إسناد ، حيث تقوم بإسناد قيمة التعبير $A(4)+1$ إلى $A(5)$

تتميز لغة Plankalkul بطريقة التعبير ، حيث إن كل صيغة تتكون من سطرين أو ثلاثة أسطر من الشفرة ، السطر الأول عبارة عن صيغة عادية كما في اللغات الأخرى ، أما السطر الثاني فهو يوصف مرجعية المصفوفة للسطر الأول . أما السطر الأخير فيحتوي على نوع البيانات للمتغيرات المذكورة في السطر الأول ، انظر المثال التالي الذي يوضح هذا المفهوم مستخدما الصيغة

$A(4)+1$ to $A(5)$

$| A + 1 \rightarrow A$

$V | 4 \rightarrow 5$

$S | I.N \quad I.N$

حيث إن :

السطر V : يرمز إلى المعامل

السطر S : يرمز إلى نوع البيانات ، حيث أن $I.N$ تعني إن هذه العبارة عن N بت من الأعداد الصحيحة (Integer).

2.1. لغة الآلة Pseudo codes

عزيري الدارس، كانت أجهزة الحاسوب في الفترة من أواخر 1940م وفي بدايات 1950م أقل استعمالاً من اليوم ، فبالإضافة إلى كونها بطيئة وغالية ولا تحتوي على ذواكر بأحجام كبيرة كانت هذه الأخيرة في ذلك الزمن صعبة البرمجة، وذلك بسبب عدم توفر برمجيات الدعم .

كانت البرمجة تستخدم لغة الآلة حيث لم يكن هنالك لغات عالية المستوى والتي كانت مضجرة جداً وكثيرة الأخطاء، بالإضافة إلى أنها كانت تستخدم الشفرات العددية لتحديد التعليمات ، مثال ذلك التعليمة ADD (إضافة) كانت تستخدم الشفرة 14 بدلا عن النص ، حتى الحروف المفردة كانت تستخدم نفس الطريقة .

هذه الطريقة جعلت البرامج صعبة القراءة وهناك مشكلة إضافية وهي العنوان المطلقة التي جعلت البرامج صعبة التعديل ، مثال ذلك إذا كان لدينا برنامج بلغة الآلة تم حفظه في الذاكرة ، اغلب التعليمات داخل هذا البرنامج ترتبط مرجعيا بمواقع أخرى داخل البرنامج ، في الغالب عبارة عن مرجع إلى بيانات أو تشير إلى أهداف التعليمات الفرعية . حيث إن إدخال تعليمات في أي موقع داخل البرنامج تقلل من صحة كل التعليمات التي تقع بعد نقطة الإدخال في العنوان لأن العناوين إن تزيد تخلق حيز في الذاكرة للتعليمات الجديدة . ولجعل الزيادة صحيحة كل التعليمات التي ترجع إلى العناوين بعد الزيادة يجب أن توجد وتُعدّل . أيضا هنالك مشكلة مشابهة عند حذف تعليمة .

هذه هي المشاكل القياسية مع كل لغات الآلة، لذا كان دافعا لاختراع المجمعات Assemblers ولغة Assembly language.

3.1. أمثلة على لغات الآلة

في هذا القسم من الوحدة سوف نأخذ بعض الأمثلة لبعض لغات الآلة وهي :

أ/ لغة الشفرة القصيرة Short Code

استخدمت هذه اللغة في حواسيب بايناك (BINAC) بواسطة جون موشلي 1949م ، ثم بعد ذلك انتقلت إلى حواسيب يونيفاك 1 (UNIVAC1) لعدد من السنين .

تحتوي يونيفاك 1 على كلمات من 72 بت . تجمع في 12 فرقة سداسية البايت .

استخدمت الشفرة القصيرة مع مفسرات وسميت البرمجة التلقائية.

ب/ لغة التشفير السريع Speed Coding :

عبارة عن لغة أكثر استخدمت عمليات الأعداد الحقيقية . وطورت بواسطة جون باكاس (John Bakas) للاستخدام مع حواسيب IBM 701 (1954) .

تستخدم الشفرة السريعة مفسرات مكنت من تحويل IBM 701 إلى حاسبات للأعداد الحقيقية تستخدم الطريقة الثلاثية للمصفوفة.

ج/ نظام يونيك للترجمة :

بين العامين 1951م و 1953م تمكن الفريق بقيادة العالم جريس هوير من تطوير مجموعة من أنظمة الترجمة (Compiling) سميت A-O ، A-1 ، A-2 ، حيث تم تطوير لغة آلة بنفس طريقة لغة التجميع وأدخلت تحسينات على لغة الآلة من حيث طول الشفرة وغيرها.

2. لغة التجميع

ظهرت لغة التجميع بوصفها لغة ترميز تستخدم الرموز SYMBOLIC CODE للتعبير عن تعليمات لغة الآلة، وذلك لمواجهة صعوبة الترميز بلغة الآلة، ولغة التجميع لغة قريبة من لغة الآلة التي يفهمها الحاسب الآلي، وتسمى هذه اللغات بلغات المستوى البسيط . ويتم استعمال مختصرات ورموز يسهل حفظها وكتابتها لكل تعليمة من تعليمات لغة الآلة، ولغة التجميع كما في لغة الآلة مصممة للعمل على حاسب معين، مما يوفر قدرة أكبر على استغلال موارد الحاسب الآلي ووحدة المعالجة المركزية بشكل أفضل، ويقوم البرنامج المسمى المجمع ASSEMBLER بترجمة البرنامج المكتوب بلغة المجمع إلى لغة الآلة .

أسئلة التقويم الذاتي



(1) عرف المصطلحات الآتية:

(أ) لغة الآلة.

(ب) لغة التجميع.

(2) اذكر خصائص لغة Plankalkul.

(3) اذكر ثلاثة أمثلة للغة الآلة.

3. اللغات العليا

سميت بهذا الاسم لأنه أصبح بإمكان المبرمج كتابة البرنامج دون معرفة تفاصيل كيفية قيام الحاسب بهذه العمليات، كمواقع التخزين وتفاصيل الجهاز الدقيقة. وتعبيرات لغات المستوى العالي هي تعبيرات شبيهة إلى درجة كبيرة باللغة الطبيعية التي يستخدمها الإنسان في حياته للتواصل. والتخاطب مع الآخرين . ومن مميزات اللغات العليا التي تميزها من لغات المستوى البسيط، بالإضافة إلى ما سبق، أن هذه اللغات غير مرتبطة بجهاز معين . أي يمكننا تنفيذ البرنامج المكتوب بلغة من لغات المستوى العالي، كالفورتران أو الكوبول أو البيسك على أكثر من جهاز، كما يمكن استخدام أكثر من لغة ترجمة على حاسب معين . كذلك، فإن اكتشاف الأخطاء وتصحيحها أصبح أكثر سهولة بسبب سهولة قراءة البرامج وتتبعها وفهمها . تسمى اللغات كالكوبول والفورتران والبيسك باللغات العليا الموجهة نحو إجراءات الحل ، PROCEDURE - ORIENTED LANGUAGE ، وهي اللغات التي يعطي فيها المبرمج التعليمات خطوة خطوة . ويمر البرنامج المكتوب بلغات المستوى العالي بثلاث مراحل قبل أن يكون جاهزاً للتنفيذ .

الترجمة TRANSLATION :

تحويل البرنامج المكتوب بلغة المستوى العالي إلى لغة الآلة .

الربط LINKING :

ربط الروتينات المكتوبة الكائنة بالمكتبة، والتعليمات الضرورية بالبرنامج .

التحميل LOADING :

يقوم البرنامج بتحميل شفرة الهدف والروتينات المكتبة، والتعليمات على الذاكرة الرئيسة بغرض التنفيذ . وكانت اللغات تصنف حسب قوتها إلى لغات أعمال، أو لغة علمية أو لغات مبتدئين وفي وقتنا الحاضر فإن معظم اللغات يمكن استخدامها بكفاءة في تطبيق أنواع التطبيقات كافة، ومن أشهر لغات المستوى العالي :

1.3. لغة الكوبول COBOL

وكلمة COBOL هي اختصار للعبارة الإنجليزية Common Business Oriented Language ، وقد طورت وصممت هذه اللغة من لدن لجنة من مصنعي أنظمة الحاسب الآلي ومطوريهها عام 1960م تسمى لجنة CODASYL اختصاراً لـ Conference of Data System Languages . وقد قام المعهد الوطني الأمريكي للمعايير ANSI بتطوير نسخة معيارية من COBOL ، مفسر كوبول يطابق المواصفات القياسية لهذا المعهد، وكان ذلك في عام 1968م، وسميت "ANSI 1968" وظهر بعدها ANSI 1974 و ANSI 80 .
وينتشر استخدام لغة كوبول على نطاق واسع عالمياً حيث تستخدم في البنوك وفي المنظمات الحكومية، وتستخدم على حاسبات كبيرة أو على حاسبات شخصية . وتتميز لغة كوبول بقدرتها على التعامل مع الملفات؛ لذا اشتهرت بأنها لغة أعمال .

2.3. لغة الفورتران FORTRAN

وهي اختصار عبارة FORMula TRANSlation ، وتعد لغة FORTRAN أقدم اللغات ذات المستوى العالي، وقد ظهرت هذه اللغة في منتصف الستينيات، حيث بدئ عام 1954 بالعمل على تطوير لغة برمجة تقبل برنامجاً مكتوباً بلغة قريبة من لغة الإنسان، وتحوله إلى شفرة قابلة للتنفيذ على الحاسب الآلي، وبعد 3 سنوات أي في عام 1957م . ظهر أول مفسر فورتران. وفي تلك الفترة كان استخدام الحاسب الآلي يكاد يكون حكراً على العلماء والمهندسين والرياضيين، ومن الطبيعي أن تكون هذه اللغة المطورة حديثاً قد جاءت لتواكب احتياجاتهم إذ تتميز لغة FORTRAN بقدراتها على إجراء العمليات الحسابية المعقدة وحل المعادلات الرياضية .

لا شك إن التقدم الذي أحرزته حواسيب IBM 701 في العام 1954م يعتبر مهماً في مجال البرمجة لأن إمكانياتها هي التي أعطت الدافع لتطوير لغة فورتران.

1.2.3. خلفية تاريخية :

أدخلت حواسيب IBM 701 عمليات الأعداد الحقيقية في عالم البرمجيات لأن هذه العمليات تحتاج إلى زمن معالجة طويل مما يزيد من عبء المعالج، وذلك بسبب أن هذه العمليات تحتاج إلى زمن أطول في التفسير والمحاكاة في عملية البرمجة .

اعتبرت لغة فورتران أول لغة من لغات المستوى العالي تحتوي على مترجم (Compiler)، حيث إن المترجم طبق في العام 1952م .

2.2.3. تصميم فورتران

حتى قبل الإعلان عن نظام IBM 701 في مايو 1954م بدأت خطط للغة الفورتران في نوفمبر 1954م. أنتج جون باكاس مع مجموعة تقريراً بعنوان " FORTRAN " اختصاراً لـ Formula TRANslating System .

هذه الوثائق وضعت أول نسخة من الفورتران ، وسميت FORTRANO . كما نصت الوثائق على أن لغة الفورتران سوف تقلل من الأخطاء في الشفرات وعملية معالجة الأخطاء.

بالنظر إلى البيئة التي طورت فيها لغة الفورتران نجد أن :

1/ الحواسيب كانت صغيرة ، بطيئة وغير موثوقة.

2/ الاستخدام للحواسيب كان للحوسبة العلمية.

3/ لم تكن هنالك طريقة كفؤة لبرمجة الحواسيب .

4/ كانت الحواسيب غالية مقارنة مع تكلفة المبرمجين لذا كان الهدف الأساسي من مترجم فورتران الأول هو تسريع شفرة الهدف النتيجة .

3.3. لغة البيسك BASIC LANGAUGE

وهي اختصار للعبارة الإنجليزية Beginner's All Purpose Symbolic Instruction Code أي اللغة المتعددة الأغراض للمبتدئين، وتأتي كلمة BASIC التي تعني الأساس لتحقيق المعنى نفسه، فهذه اللغة ونظراً لبساطة تعليماتها ومحدوديتها فإنها تعد لغة مناسبة للتعلم من قبل المبتدئين في عالم الحاسب الآلي والبرمجة، وتستخدم هذه اللغة في معظم الحاسبات الشخصية، مما يدل على الانتشار الواسع لها، وقد طورت هذه اللغة في كلية Dartmout عام 1963م من لدن John Demuy و Thomas Kurtz . وتستخدم لغة Basic الحديثة في قطاع واسع . إذ تستخدم في مجال الأعمال لقدرتها على التعامل مع الملفات، وكذا في العمليات الرياضية من قبل العلماء والمهندسين لامتلاكها كثيراً من الوظائف للقيام بمثل تلك العمليات المعقدة .

4.3. لغة باسكال PASCAL

عزيري الدارس، وسميت باسم العالم الفرنسي الرياضي الفذ Blaise Pascal ، وصممت هذه اللغة من قبل العالم السويسري Niklaus Wirth ، وطرحت عام 1971م، وقد انتشرت هذه اللغة خصوصاً في الجامعات . إذ تدرس هذه في معظم جامعات العالم لطلاب علوم الحاسب، وتمتاز لغة PASCAL بالسهولة، واختصار الكلمات فيها إلى حد كبير، وبنيتها التركيبية وقوة البرامج الفرعية، واستخدام المؤشرات Pointer ، وقد ظهرت نسخ جديدة من Pascal امتازت بتلافي العيوب في النسخ السابقة، ويعد TurboPascal من شركة Borland من البرامج المستخدمة بكثرة بين المبتدئين والتخصصيين في البرامج، وتتنافس هذه اللغة لغة Basic في كثرة المستخدمين لتوافر كثير من المميزات فيها، وتستخدم هذه اللغة من قبل الطلاب والمهندسين، كما تستخدم في قطاع الأعمال التجارية . تاريخ لغة الباسكال تعود قصة الرياضي والفيلسوف الفرنسي بليز باسكال إلى منتصف القرن التاسع عشر ميلادي عام 164 م حيث أن هذا الشاب والذي كان يبلغ الثامنة عشر ربيعاً والذي كانت بواذر العبقرية والاختراع تبدو جلية على أعماله وحيث أنه كان شديد الإبداع في علم الرياضيات مولعاً بالاكشافات العلمية وتطبيق النظريات الرياضية ففي نفس السنة اخترع أول حاسب نصف آلي وسمي باسمه (حاسب باسكال 1642) ، وكان ذلك لسبب أساسي وهو مساعده أبيه الذي كان يعمل في مؤسسة الضرائب محصلاً للفواتير والذي كان يقضي معظم لياليه مستخدماً العد اليدوي في إحصاء وتدقيق حسابات المبالغ التي حصلها وقد كان يشكل

هذا النوع البطيء من الحساب إرباكاً لعائلته ، وبالتالي يأخذ منه الوقت الكبير .

يتكون حاسب باسكال الميكانيكي من مجموعه متتالية من الإطارات (الأقراص) كل واحد منها مرقمة من الصفر إلى التسعة ، هذه الإطارات مرتبه بحيث تقرأ الأرقام المسجلة عليها من اليسار إلى اليمين ويتم إدارتها يدويا عن طريق الذراع ، فعندما يتم أحد هذه الأقراص دورته من الصفر إلى التسع فإن نتوء الرقم 9 يدفع الطارة المجاورة له رقما واحدا وعند ذلك أي دورة الطارة الأولى تسعة مواقع متتالية (دوره كاملة) فإنها تدفع الطارة التالية لها من اليسار موقعا واحدا... وهكذا حتى يتم تسجيل العدد . وبهذه الطريقة استطاع والد باسكال إجراء عمليات الجمع والطرح أما عمليات الضرب والقسمة فتتم بتكرار عمليات الجمع والطرح مرات متعددة.

5.3. لغة سي C

طورت هذه اللغة في معامل Bell من قبل Dennis Ritchie وهي تطوير لنسخة قديمه تسمى B من لغة BCPL التي ظهرت عام 1969م . لذا سميت هذه النسخة اللاحقة بـ C . وتشتهر لغة C باستخدامها كلغة برمجة نظم system software ، حيث تستخدم لكتابة برامج النظم التشغيلية . إذ إنها تعد لغة قريبة، وتشبه إلى حد كبير لغة التجميع As-sembly وتمتاز بسرعتها الكبيرة، كما تملك مجموعة جيدة من التعليمات، كما أنها لغة قابلة للنقل من جهاز إلى آخر لصغر الجزء الواجب نقله منها، وتعد لغة C من اللغات التركيبية Structured Language . وازداد اهتمام محترفي البرمجة بلغة الـ C ، والنسخ المحسنة منها C++ . إذ إن معظم التطبيقات تكتب بها .

6.3. لغة فيجوال بيسك

عزيري الدارس، تحتل لغة بيسك الصدارة بين باقي اللغات الأخرى من حيث انتشارها وتليبيتها لمطالب المبرمجين المتنوعة، و يعود ذلك إلى سهولة استخدامها و مرونتها الشديدة . وقد واكبت هذه اللغة التطور الهائل الذي حصل مؤخرا على صعيد العتاد Hardware و البرمجيات Software من خلال حلتها الجديدة لغة فيجوال بيسك و التي تلاقي نجاحا عظيما لسهولة استخدامها و تعاملها السلس من النظام Windows وقد وافقت لغة فيجوال بيسك باقي لغات البرمجة الحديثة من حيث أسلوبها المتطور إذ تعد

هذه اللغة من لغات التطوير السريع للتطبيقات Rapid Application Development أو ما يدعى بلغات RAD . تتميز لغة فيجوال بيسك بقدرتها على التعامل مع عالم الوسائط المتعددة Multimedia إضافة إلى سهولة استعمال مكتبات الربط الديناميكية (DLL) و التي تعطي قدرة إضافية للمبرمجين للاستفادة من بعض القوالب البرمجية المسبقة للتطوير مما يوفر وقتا و جهدا كبيرين ، و تتعامل أيضا مع واجهة التطبيقات البرمجية الخاصة بنظام (Windows) أو ما يدعى (Windows API's) ، كما يستطيع المبرمج باستخدام هذه اللغة كتابة برامج ذات الواجهة متعددة الوثائق (MDI) إضافة إلى إنشاء أيقونات متحركة إضافة إلى المتحكمات المرفقة بالصوت إلى تطبيقاته المختلفة ، و لم تهمل لغة فيجوال بيسك موضوع التعامل مع الملفات على اختلاف أنواعها ، و منها ملفات قواعد البيانات (Database) إذ تتعامل مع هذه الملفات المعدة بواسطة مايكروسوفت أكسس أو أحد البرامج الشهيرة في هذا المجال مثل dBase . تعتبر الإصدار (Visual Basic 6) آخر الإصدارات في عائلة فيجوال بيسك . و هي تسمح لك بسرعة و بسهولة بتطوير تطبيقات (Widows) لكمبيوترك الشخصي دون أن تكون متمرسا في لغات البرمجة أو حتى . C++ . و قد أصبحت هذه اللغة من مصاف اللغات كائنية التوجيه (OOP) ابتداء من الإصدار الخامس منها.

يزودك فيجوال بيسك ببيئة رسومية و التي بواسطتها تستطيع أن تصمم نظريا النماذج و التحكمات و التي تصبح هي أساسيات بنائك في تطبيقاتك لكمبيوترك الشخصي . فيجوال بيسك يدعم عددا كبيرا من الأدوات المفيدة ، و التي تساعدك لإنتاج أكثر ، و هذه الأدوات تشتمل على ما يلي : نماذج ، قوالب ، متحكمات متخصصة ، إضافات و مديري قواعد البيانات . يمكنك استخدام هذه الأدوات جميعها لإنشاء تطبيقات كاملة في شهور ، أو أسابيع أو حتى أيام . إن فيجوال بيسك 6 صمم خصيصا ليخدم شبكة الإنترنت و يأتي معه عدد من التحكمات تسمح لك بإنشاء تطبيقات على أساس Web (تسمى ActiveX executables) وهذه تعمل كأنها تطبيق فيجوال بيسك منفرد ، و لكنها يتم توحيدها عبر (Web browser) . باستخدام هذا النمط الجديد في التطبيق يمكن مراجعة تطبيقات فيجوال بيسك الموجودة و توزيعها على الإنترنت . و الجديد في فيجوال بيسك هي مشاريع SAPI و مشروع قوالب Dynamic HTML ، و تزودك هذه القوالب بهيكل عملي لتطوير مكونات جانبية لـ الأجهزة الخادمة في الويب.

1.6.3 Visual Basic.NET

وهي أحدثها (حيث تم إصدارها وتم إعادة تصميمها بالكامل) إن لم نقل أنها لغة برمجة جديدة، حيث أصبحت اللغة الآن لغة برمجة كائنية OOP Language كاملة . فاصبحت تدعم الوراثة Inheritance وإعادة تعريف الدوال وغيرها . وأهم مزاياها أنه يمكنك تطوير برمجيات يمكن الاستفادة منها على منصات غير الويندوز مثل لينكس وماك وغيرها . وتسمى هذه التقنية الجديدة خدمات - عبر- الشبكة . WEB Service .

لغة الفيجوال بيسك ليس كما كانت. لعلك سمعت كثيرا عن لغة الـ BASIC ، وقد تجنب تعلمها بسبب حدودها التي تقصر إمكانياتها كسائر لغات البرمجة . لكن مع الإصدارات الحديثة من Visual Basic فإن الأمر اختلف حيث أصبحت إمكانيات اللغة لا حدود لها ، وقابلية التطوير لا نهائية، أي أنه أصبح ذا نهاية مفتوحة . فعن طريق الإضافات Add-Ins وأدوات التحكم الخارجية ActiveX Control ومكونات COM بصفة عامة ، تستطيع إنجاز كل ما استطعت انجازه باللغات المختلفة . فالإضافة مكون COM جديد لا يتطلب الأمر منك سوى تحديد اسم وملف المكون ومن ثم استخدامه مباشرة. التطبيقات التي تنشئها Visual Basic متوافقة 100%/ مع إصدارات ويندوز المختلفة . فالنواة الداخلية للتطبيقات المنشأة بواسطة فيجوال بيسك هي عبارة عن سيل من إجراءات API التي هي عبارة عن روح نظام ويندوز . أما الدوال الإضافية التي توفرها لغة البرمجة فهي موجودة في مكتبة مستقلة MSVBVM60 وهي المسؤولة عن تشغيل برامجك التي طورتها عن طريق فيجوال بيسك. فيجوال بيسك يوفر لك العديد من الحلول الخاصة بالانترنت. فيمكنك من انشاء ادوات تحكم ActiveX Controls يتم تنفيذها في صفحة ويب . أو تصميم تطبيقات متقدمة كـ ActiveX Documents للعمل على متصفح . Internet Explorer ، فيجوال بيسك يوفر لك بيئة تطوير خاصة لتطوير تطبيقات انترنت سواء أكانت للعميل Client كتطوير تطبيقات من نوع DHML Applications او للخادم Server كتطوير تطبيقات ASP Applications

2.6.3. بنية اللغة

لغة البرمجة BASIC هي الجذر الأصلي للبرمجة بلغة فيجوال بيسك. فمعظم الصيغ العامة Syntax كعبارة If أو حلقة For Next لم تتغير . لكن هنالك بعض الامور التي تغيرت كي تتناسب مع بيئة نظام ويندوز. فلا تتوقع وجود الإجراءات . Locate بصفة عامة الدوال المبنية Built-in Functions قد تغيرت

تغير كلي وتم اضافة مئات الدوال الجديدة التي لا بد لها من التأقلم مع بيئة ويندوز . ابحث في مكتبة ADL عن هذه الدوال . بالاضافة إلى ذلك ، تطورت هذه اللغة تطوراً كبيراً حتى أصبحت OOP تقريباً. فمبدأ الـ Encapsulation والـ Polymorphism مدعومة بشكل ممتاز عن طريق إضافة ملفات مستقلة بالامتداد CLS. لملفات الفئات. أما تعدد الواجهات فيتم عن طريق إضافة الكلمة المحجوزة Implement فقط وتصبح الفئة قابلة لاشتقاق واجهة من أخرى.

3.6.3. الرسائل Messages واجراءات API

لن تحتاج إلى استخدام آلاف الثوابت لقنص رسائل النظام كـ WM_CLICK وغيرها. فمع فيجوال بيسك يكفي أن تضع الأداة على النافذة وتتقر عليها نقراً مزدوجاً لتعرف اجراءً يمثل اسم الاداة والرسالة المراد قنصها. في عالم فيجوال بيسك يعرف هذا النوع من الاجراءات بالاحداث Events حيث تحتوي كل اداة على مجموعة أحداث خاصة بها بمثابة الرسائل الموجودة في لغات البرمجة الأخرى . أما بالنسبة لـ Windows API فتستطيع الوصول لهذه الاجراءات عن طريق تعريف الدالة باستخدام الكلمة المحجوزة Declare مع تضمين المكتبة الديناميكية الموجودة فيها الاجراء كـ User أو. GDI برمجة الكائنات المكونة: COM لا يتطلب منك الامر إلى الدخول في التفاصيل وتعريف واجهات IUnknown وغيرها لبرمجة المكونات . COM حدد المشاريع من نوع ActiveX وبرمج بنفس الطريقة العادية وستنتج مكونات COM حقيقية بسهولة شديدة. الواجهات Interface تتم عن طريق تعريف الفئات Classes فلا يوجد حاجة الآن لاستخدام لغة تعريف الواجهات . IDL أما عن مسارات التنفيذ Threading فتستطيع التحكم وتحديد نوعه فيما إذا كنت تريد مساراً تنفيذياً فردياً Single أو متعدد Multi عن طريق صندوق حوار خصائص المشروع. Project Properties . المترجم : Compiler يوفر لك فيجوال بيسك خيارين للترجمة هما P-Code و Native Code مع خيارات ممتدة للنوع الثاني.

كما يدعم مترجم فيجوال بيسك معالجات Pentium Pro دعماً كاملاً للاستفادة من المعالجات الرياضية بها وغيرها.

4.6.3. مستقبل لغة الفيچوال بيسك

حقق فيجوال بيسك شعبية لا مثيل لها بين مطوري التطبيقات تحت بيئة ويندوز والفرص الوظيفية لمبرمجة فيجوال بيسك هي الأعلى . كذلك، المواقع التي تناقش هذه اللغة في زيادة رأسية! وإعداد المبرمجين المهاجرين لفيجوال بيسك يومي إلى الزيادة . الزيادة أيضاً ،أدوات التطوير الخاصة بفيجوال بيسك كمكونات COM في كل مكان ويكفيك وجود أكثر من ثلاثة مجلات عالمية تناقش هذه اللغة Visual Basic.NET التي بدورها طريقك إلى محاذاة الركب وتطوير تطبيقات انترنت الذكية . ستهبط شعبية اللغة تدريجياً خلال السنوات القادمة لصالح لغة الجافا مالم تقدم تقنية الدوت نت الجديدة جميع مزايا الجافا وتفتح شركات أخرى بقبول هذه التقنية. ورغم أن شركة مايكروسوفت أثبتت قدرتها في هذا المجال إلا ان مستقبل هذه اللغة البعيد لا يبدو ساطعاً مثل لغة الجافا . الخبر السعيد أنه خلال السنوات الخمس القادمة ستظل اللغة قوية وموجودة بشكل واسع.

لغة البرمجة لازمة لكتابة تعليمات للحاسوب من أجل تأدية عمل ما. ويستخدم الإنسان في كتابة هذه التعليمات قواعد معينة حسب طريقة تصميم لغة البرمجة التي يعمل عليها. ثم يتم بعد ذلك نقل هذه التعليمات للجهاز عن طريق مترجم Compiler أو مفسر Interpreter خاص بتلك اللغة وذلك لترجمة هذه التعليمات للآلة. وعملية الترجمة أو التفسير تعني عملية تحويل أسطر البرنامج المكتوب من الشكل الذي يكون قريب الشبه بلغة الإنسان إلى اللغة البدائية البسيطة (1/0) التي تفهمها دوائر الحاسوب والتي تنفذها بعد ذلك. ثم يقوم المترجم أو المفسر بعد ذلك بإخراج النتائج ليس باللغة البدائية البسيطة ولكن بلغة الإنسان الراقية حتى يتسنى له فهمها.

ولقد صمم الحاسوب بحيث يمكنه تنفيذ تعليمات مصاغة فقط بلغة الآلة Machine Language وهذه اللغة مكونة من فئة من شيفرات الدوال OperationCodes التي يمكن لوحدة المعالجة المركزية تنفيذها مباشرة. تتكون التعليمات الواحدة في لغة الآلة من سلسلة من الأعداد تعتمد على التصميم المنطقي للحاسوب وقد تمثل بالنظام الثنائي أو النظام العشري أو النظام الثماني أو النظام السادس عشر أو أي نظام رقمي آخر. وأحد أجزاء التعليمات يقوم بتوجيه الحاسوب عما يجب فعله من عمليات (طرح، جمع، ضرب، قسمة، مقارنة، ...) والأجزاء الأخرى تُعلمه أين يجد البيانات التي سوف ينفذ عليها العمليات الحسابية والمنطقية وأين تخزن النتائج في وحدة الذاكرة.

وتسمى البرامج المكتوبة بلغة الآلة بالبرامج الفعلية. Object Programs. ولقد أدى التطور والنقد في طرق البرمجة إلى تطوير لغات التجميع Assembly Languages والتي صممت لتتناسب المبرمج وليس الآلة. وفي مثل تلك اللغات تكتب التعليمات باستخدام أسماء رمزية Symbolic Names تمثل شيفرات الدوال وعناوين في الذاكرة. هذه الأسماء غالبا مختصرة وسهلة التذكر. فمثلا الأمر ADD اختصار لعملية الجمع. Addition.، والأمر SUB اختصار لعملية الطرح Subtraction ... وهكذا.

وينبغي أن يترجم البرنامج المكتوب بلغة التجميع إلى لغة الآلة قبل تنفيذه. وذلك لأن الآلة تستطيع كما ذكرنا تنفيذ التعليمات المكتوبة فقط بلغة الآلة. وعملية الترجمة تتم بواسطة الحاسوب بمساعدة الـ Assembler وهو برنامج مكتوب بلغة الآلة. وحيث إن لغة التجميع تستخدم أسماء رمزية فإنها سهلت البرمجة و إلى حد ما مكنت من اكتشاف الأخطاء في البرنامج.

نلاحظ أن لغة الآلة ولغة التجميع عبارة عن لغات تعتمد على الحاسوب بمعنى أنها صممت للاستخدام على أجهزة ذات تصميمات محددة. وقد أدت الحاجة إلى تطوير لغات برمجة لا تعتمد على الأجهزة التي تعمل عليها إلى تطوير لغات مستقلة عن الأجهزة Procedure Oriented Languages قريبة الصلة بمفردات العلم والهندسة والرياضيات وكذلك المستخدم. وهي تعكس بوضوح الطريقة Procedure المستخدمة لحل المشكلة وتسمح ببرمجة سهلة وإزالة سريعة للأخطاء. وغالبا ما تسمى لغات الآلة ولغات التجميع باللغات ذات المستويات الدنيا أو البدائية. Low Level Languages أما اللغات المستقلة عن الأجهزة فإنها تسمى باللغات ذات المستويات العليا أو الراقية High Level Languages ومن الأمثلة عليها لغة السي والفجوال بيسك والجافا وغيرها.

تاريخ السي و السي بلس بلس History of C and C++

طُورت الجافا من السي بلس بلس، والسي بلس بلس طُورت من السي. BCPL عرفت عام 1967م عن طريق Martin Richards لكتابة برامج أنظمة التشغيل والمجمعات "compilers". Ken Thompson قسم العديد من مختلف الهيئات في لغته الـ B بعدما ادمج مع الـ BCPL، استخدم الـ B لإنشاء إصدارات مبكرة لنظام التشغيل يونكس في لابات بيل عام 1970م.. أخذت لغة السي من لغة البي عن طريق العالم دينيس ريتشي في معامل بيل وكتبت أساسياتها عام 1972م ومنذ تأسيسها أصبحت ذات استخدام واسع خصوصا في تطوير لغة نظام التشغيل يونكس..

واليوم معظم الشفرات لأنظمة التشغيل العامة (التي توجد في اللابتوبز والديسكتوبز ومحطات العمل والخادمت الصغيرة) تُكتب بالسي والسي بلس بلس.

أستخرجت السي بلس بلس من السي وطورت عن طريق Barney Stroustrup في بدايات 1980م في معامل بيل..

تقريبا السي بلس بلس تشبه السي كثيرا لكن تعتبر السي بلس بلس أثر اهمية .. وتعتمد هذه اللغة على الـ OOP أوبجكت أورينتد بروقرام.

تعتمد السي بلس بلس لغة مولدة أي أنك تستطيع برمجتها أيضا في السي.

سببت السي بلس بلس ثورة في عالم البرامج لأن بناء البرامج أصبح سريعا ، صحيحا ، اقتصاديا خصوصا عندما يكون هناك حاجة لبرامج قوية وجديدة ومرتبطة.

بشكل عام تنقسم الأشكال objects في السي بلس بلس إلى قسمين : الخواص attributes (كالأسماء والألوان والأحجام) والمتصرفات behaviors (كالحسابات والتحريك والتوصيل).

مطوري البرامج اكتشفوا أن استخدام الـ modular و object orinated design وكتابة الاكواد تستطيع جعل جماعات تطوير البرامج أكثر تطورا من تقنية البرمجة العامة (كالبرامجات المركبة) .. الأو أو بي OOP أسهل في الفهم والإستيعاب. ولغة الجافا هي أكثر لغة تستخدم الأوبجكت أورينتد بروقرام ...

تاريخ الجافا History of Java

كانت ثورة الجافا أهم سبب في صناعة وتطوير الكمبيوترات الشخصية ، والتي تعتبر الآن واحدة من مئات الملايين في العالم ، أثرت الكمبيوترات الشخصية بشكل عميق في حياة الأشخاص وطرق إدارة منظماتهم ومشاريعهم.

المايكروبروسسر لها تأثير كبير في استهلاك الأجهزة الإلكترونية الذكية ، لأجل ذلك شركة Sun Microsystems عام 1991م كان لها تأثير في بحث اتحاد داخلي لمشاريع الشفرات والاكواد والتي طورت لتطوير لغة السي بلس بلس.

كان يطلق على الجافا اسم Oak في البدايات لكن تم تغييرها إلى الجافا عندما قام فريق من Sun بزيارة للمقهى المحلي واعتمدوا الاسم في ذلك الوقت.

هذا المشروع استغل وطبق وكان في ذلك الوقت صعباً جداً ، لذلك لم تتطور ساحة تسوق واستهلاك الاجهزة الإلكترونية الذكية في بدايات عام 1990 م بسرعة كما توقع Sun بل كان لابد لها من التأجيل وكاد المشروع أن يواجه مشكلة الإلغاء في أي وقت . ولكن عن طريق الحظ والصدفة الورد وايد ويب فجرت عام 1993م ورأى الأشخاص العاملين في Sun الإمكانية المباشرة لاستخدام الجافا لأضافة الاشكال الديناميكية والتي كان لهادور كبير في التأثير وإنعاش صفحات الويب .. أصدرت شركة Sun لغة الجافا في May 1995 واعتمدتها كلغة رسمية من لغات البرمجة ، ضمنت الجافا اهتمام كبير للجماعات التجارية بسبب ظهور المتعه في الورد وايد ويب ، تستخدم الجافا الان لتطوير تطبيقات المشاريع الكبيرة ، ولتعزير دوال خادم الويب (هو الكمبيوتر الذي يعطي المحتويات التي نراها في مستعرض الويب) ، ولإعطاء التطبيقات في الاجهزة الإستهلاكية (مثل شرائح التلفون والمساعدات الرقمية الشخصية) وغيرها من الاشياء الأخرى.

تاريخ لغات الفورتران و كوبول و آدا و باسكال FORTRAN , COBOL , Pascal and Ada

طورت مئات لغات البرمجة العالية لكن القليل منها اكتسبت قوة وشهرة واسعة مثل الفورتران .. FORTRAN (FORmula TRANslator) و التي طورت عن طريق شركات الأي بي إم في اواسط 1950s م لتستخدم في التطبيقات العملية والهندسية التي تحتاج إلى تطبيقات حسابية معقدة. فورتران تستخدم بشكل واسع في التطبيقات الهندسية.

(OCBOL (Business Oriented Language Common) طورت في نهايات 1950 م عن طريق معامل الكمبيوتر وحكومة الولايات المتحدة ومعاهد استخدام الكمبيوتر ، كوبول تستخدم لغرض التطبيقات التجارية التي تحتاج إلى تسعيرات ومعالجات فعالة لاعداد كبيرة من البيانات ، معظم البرامج التجارية بقيت تُبرمج عن طريق الكوبول.

إلى نهايات عام 1960 م حاول كثير من مطوري البرامج تطوير وتعزيز الخادومات الصعبة وتوزيع البرامج كان عادة ما يتأخر، بالإضافة إلى الأسعار التي تجاوزت الحد المعقول. والمبيعات كانت تنتهي بسرعة، لذلك بدأ الناس يدركون أن تطوير البرامج أكثر تعقيدا مما يتصورون ..

توصلت الابحاث في الـ 1960 م إلى أن تطور البرمجة المركبة تقترب من كتابة البرامج السهلة والمفهومة للإختبار والتجربة، وأسهل في التحديد من مبيعات البرمج الكبيرة مع مختلف التقنيات .. واحدة من العديد من النتائج الحقيقة من هذه الأبحاث كانت تطوير لغة البرمجة Pascal عن طريق البروفيسور Niclaus Wirth وسماها باسم العالم الرياضي والفيلسوف باسكال قبل سبعة عشر قرن .. باسكال صممت لتدريس البرامج المركبة في البيئات الأكاديمية. وبسرعة أصبحت لغة البرمجة المفضلة لدى معظم الكليات.

تفتقر لغة باسكال للعديد من من الهيئات التي تحتاجها لجعلها أكثر استخداما في التطبيقات التجارية والحكومية وفي المعاهد لذلك ليست فهي واسعة الاستخدام في هذه البيئات ..

7.3. لغة آدا ADA

لغة البرمجة Ada طورت تحت كفالة أقسام الولايات المتحدة للحماية (DOD) طوال عام 1970م، وبدايات 1980 م. المئات من اللغات المختلفة بدأت في استخدام مبيعات DOD التي سيطرت بضخامة وتحكمت في انظمة البرامج.. أرادت الـ DOD لغة منفردة تلبي معظم احتياجاتها .. سميت Ada بهذا الاسم نسبة إلى السيدة Ada ابنة الشاعر Lord Byron ، التي بدأت كتابة لأول برنامج كمبيوتر عالمي بدايات 1800م لتحليل تصميم أجهزة الكمبيوتر المحركة عن طريق تشارلز بابوج واحدة من أهم القدرات لـ Ada تسمى multitasking أي تسمح للمبرمجين أن يقرروا ما إذا كان يسمح لهم بكتابة البرامج بتفاعل متماثل أم لا.

أما لغة البرمجة (All-Purpose Symbolic Instruction Code BASIC (Beginners فقد طورت في أواسط الستينيات في كلية Dartmouth كمعنى لكتابة البرامج البسيطة ، كانت البيسك في البدايات سببا لتعود.

أسئلة التقويم الذاتي



- (1) وضح سبب تسمية اللغات العليا.
- (2) ما المراحل التي يمر بها البرنامج ليكون جاهزاً للتنفيذ؟
- (3) ما أشهر لغات المستوى العالي؟
- (4) قم بسرد مختصر للتطور التاريخي للغات المستوى العالي.

4. لغات الجيل الرابع (GL4) Fourth-Generation language

عزيزي الدارس، سميت بهذا الاسم نسبة إلى الجيل التي ظهرت فيه، مثل هذه اللغات تتصف بقلّة التعليمات التي يكتبها المبرمج لتحقيق هدف ما ، فما كان يتطلب مئات الأسطر من لغة بيسك أو الآلاف من لغة كوبول يتم باستخدام عدد بسيط من الأسطر في هذه اللغات .

تمتاز هذه اللغات بعدة مزايا :

1- إنها لغة موجهة للنتائج أي إن المبرمج يهتم بطلب ما يريد من الكمبيوتر، دون أن يوجهه لكيفية القيام بذلك .

2- زادت من الإنتاجية؛ لأن كتابة البرامج وصيانتها أكثر سهولة .

3- سهولة الاستخدام، وتتطلب قليلاً من التدريب على استخدامها، سواء أكان للبرمجة أو غيرها .

4- لم يعد المستخدم بحاجة إلى التفكير في الأجهزة أو هيكلية البرنامج . وقد بدأت لغات الجيل الرابع بالانتشار بين المبرمجين، وبشكل كبير جداً لما توفره لغات البرمجة هذه من سهولة في معاملة الملفات، وربط هذه الملفات بعضها مع بعض بغرض التحديث أو طباعة التقارير .

كما توفر هذه اللغات قوة في المشاركة في الموارد البرمجية وخصوصاً الملفات، ومن أمثال هذه اللغات لغة Dbase من Borland ، ولغة Foxpro من Microsoft ، و Oracle من Ingress، وتستخدم لغة الاستفسار المهيكلية SQL Structured Query Language في عمليات بناء قواعد المعلومات وتحديثها، وتوفير الحماية اللازمة لها.

1.4. لغة الاستفسار البنيوية SQL

SQL هي اختصار للعبارة الإنجليزية Structured Query Language أي لغة الاستفسار البنيوية، وهي عبارة عن مجموعة من التعليمات القريبة من اللغة الطبيعية التي توجه نظام إدارة قواعد البيانات DBMS للقيام بعمليات بناء قواعد البيانات، وتحديثها والبحث فيها، وعمل قواعد الحماية للبيانات . وتمتاز لغة الاستفسار SQL بالآتي :

- 1- أنها لغة قريبة من اللغة الطبيعية .
- 2- تستطيع الحصول على أية بيانات من قاعدة البيانات، إذ يمكن الحصول على بيانات ملف كامل أو بعض الحقول من ملف أو مجموعة ملفات .
- 3- يهتم المستخدم بتعريف احتياجه What you want، ولا يهتم بكيفية الحصول على أو من أين .

5. اللغات الطبيعية Natural Language أو لغات الجيل الخامس

ويقصد بها لغة الإنسان، أي إيجاد لغة مبرمجة نستطيع بها توجيه الحاسب للقيام بما نريد من أعمال، وذلك باستخدام التعبيرات الشائعة "اطبع تقريراً يحوي اسم الطالب والدرجة"، فالمستخدم ما عليه سوى طباعة الأمر أو إملائه للحاسب الذي يلبي الطلب . فالمستخدم يستطيع إملاء النص وكتابته بعبارات متعددة تحمل المعنى نفسه، كأن يقول "اطبع الاسم والدرجة لجميع الطلاب" أو "اطبع اسم الطالب ودرجة الاختبار".

للجميع"، فالطلبان لهما المعنى نفسه، وإن اختلفا في الصياغة، فلغات البرمجة هذه قادرة على فهم تراكيب الجمل المختلفة، وإن اختلفت اللهجات أو كان هناك أخطاء لغوية، وإذا لم يفهم الحاسب المقصود يقوم بتقديم أسئلة لمزيد من الوضوح والوصول إلى الهدف.

والأبحاث في مجال اللغات الطبيعية هو من المجالات التي يبحثها علم الذكاء الاصطناعي. A.I. إذ إن فهم اللغة الطبيعية يحتاج إلى كثير من الخبرات والعلاقات في ربط الجمل وتحليل كل جملة لاستنتاج المعنى الصحيح، ومن ثم تقديم رد الفعل المناسب.

6. لغات الإنترنت

1.6. لغة HTML

وهي إحدى اللغات المستخدمة لإنشاء صفحات الويب www . إن (Hypertext Markup Language) عبارة عن لغة تحوي مجموعة من الأوامر تؤدي إلى تكوين صفحات الويب، إن إنشاء صفحة بواسطة تركيبة Ascii على منسق الكلمات يمكن أن تحول إلى صفحة على الويب بإضافة بعض أوامر Html . إن أوامر Html تمكن المستخدم من تنفيذ عدد من العمليات على صفحات الويب منها :

- تحديد حجم النص وطريقة عرضه .
- إنشاء الروابط مع الوثائق والمستندات الأخرى .
- إنشاء نماذج تفاعلية مع الصفحة .
- توفير الدعم للوسائط المتعددة كالفديو والصوت والصورة .

2.6. لغة جافا Java

وهي من اللغات المستخدمة للإنترنت ولصفحات الويب في WWW وتهدف لغة البرمجة هذه إلى إضافة الحيوية إلى صفحات الويب عبر النصوص المتحركة والرسوم التي تتحرك بشكل تفاعلي، والوسائط المتعددة .

ولغة جافا طورت من قبل شركة Sun Microsystems على غرار لغة ++C . وتقوم الوثائق المنشأة بلغة الترميز النصي Html باستدعاء برامج جافا كما يمكن تشغيلها بصورة منفردة.

أسئلة التقويم الذاتي



- (1) وضح مزايا لغات الجيلين الرابع و الخامس .
- (2) ما لغة الاستفسار البنوية SQL؟
- (3) ما الهدف من استخدام لغة جافا في صفحات الويب؟

الخلاصة

عزيزي الدارس، تناولت الوحدة تطور لغات البرمجة منذ ظهور الجيل الأول للغة الآلة في العام 1950 مع بداية اجيل الحاسبات الالية الأولى. استخدمت لغة الآلة الترميز الثنائي في البرمجة. من ثم ظهرت لغة التجميع و هي من لغات البرمجة التي استخدمت الرموز لكتابة البرامج للحاسب الآلي.

تحدثت الوحدة أيضاً عن لغات المستوى العالي، و التي مكنت من استخدام لغة أقرب الى الطبيعة لكتابة شفرات البرامج للحاسب الآلي و من أشهر هذه اللغات لغة الفورتران و البيسك وباسكال.

بعد ذلك تم التعرض للغات الجيل الرابع للحاسبات الآلية.

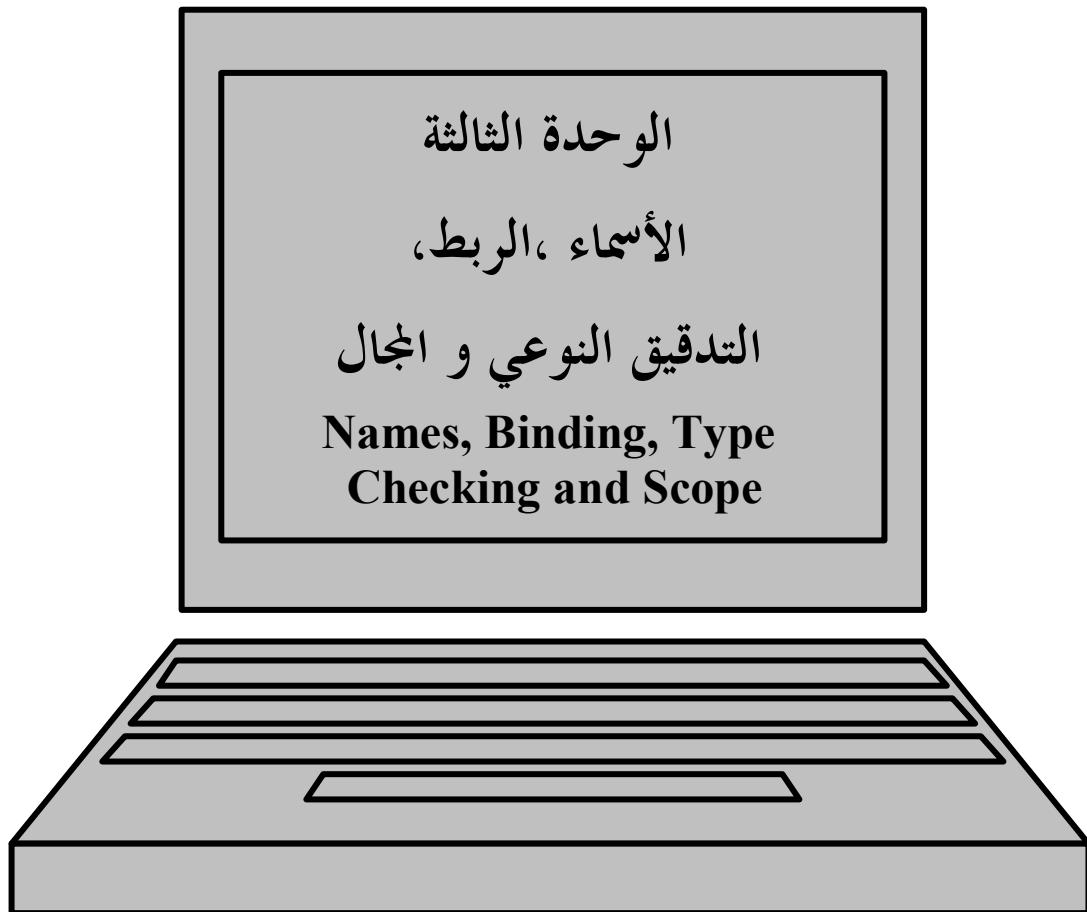
وأخيراً من المتوقع أن تكون لغات البرمجة مستقبلاً عبارة عن تعليمات منطوقة بلغة الإنسان العادية يقوم الحاسب الآلي بترجمتها و تحويلها إلى شفرات منفذة تقوم بعمل البرامج الحالية.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، سنقوم في الوحدة التالية بدراسة المدلولات الأساسية للمتغيرات، لذا سوف نقوم بدراسة طبيعة الأسماء و الكلمات الخاصة المستخدمة في لغات البرمجة.

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
51	المقدمة
51	تمهيد
51	أهداف الوحدة
52	1. الأسماء
52	1.1. متطلبات تصميم الأسماء Design Issues
52	2.1. نماذج الأسماء Name Forms
52	2. الكلمات الخاصة Special Words
53	3. المتغيرات
54	4. الربط
55	1.4. ربط النوع Type Binding
58	5. التحقق من النوع
60	6. المدى
60	1.6. الكتلة Blocks
61	2.6. أنواع المدى
56	الخلاصة
56	لمحة مسبقة عن الوحدة التالية
66	إجابات التدريبات
67	مسرد المصطلحات
68	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الثالثة من المقرر "مفاهيم لغات البرمجة" في هذه الوحدة سوف نقوم بدراسة المدلولات الأساسية للمتغيرات، لذا سوف نقوم بدراسة طبيعة الأسماء و الكلمات الخاصة المستخدمة في لغات البرمجة. وأيضاً سوف ندرس خصائص المتغيرات من نوع وعنوان و قيمة.بالإضافة إلى الربط، و زمن الربط ثم بعد ذلك سوف نقوم بدراسة عملية التدقيق النوعي و اختباره والنوع القوى و قوانين التوافق.

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تعرف الأسماء و متطلبات تصميمها و نماذجها و الكلمات الخاصة بلغات البرمجة.
- تعرف المتغيرات.
- تشرح مفهوم الربط.
- تشرح عملية التأكد من النوع.
- تعرف مدى المتغير.



عزيزي الدارس، بالنسبة للغات البرمجة، يتكون الحاسوب من مكونين أساسيين هما الذاكرة و المعالج، الذاكرة حيث تخزن البيانات و تعليمات البرمجيات. الذاكرة عبارة عن خلايا حيث تخزن المتغيرات Variables.

المتغيرات تعرف بمجموعة من الخصائص مثل نوع البيانات Data Type و مدى المتغير Scope ، بالإضافة إلى التدقيق النوعي Type Checking و القيمة الابتدائية Initialization

1. الأسماء Names

الاسم عبارة عن سلسلة من الحروف تستخدم لتعريف خاصية محددة في البرنامج. في اللغات الأولى استخدمت حروف مفردة كما في الرياضيات لأن الخلفية لتلك اللغات كانت رياضية.

1.1. متطلبات تصميم الأسماء Design Issues

النقاط التالية هي المتطلبات الأولية لتصميم الأسماء في لغات البرمجة:

- ما أقصى طول للاسم؟
 - هل من الممكن استخدام حروف الوصل في الأسماء؟
 - هل الأسماء حساسة لحالة الأحرف؟
 - هل الكلمات الخاصة بكلمات محجوزة أو كلمات مفتاحية؟
- هذه المتطلبات سوف نناقشها في الأقسام الفرعية التالية مع استخدام بعض الأمثلة لخيارات التصميم.

2.1. نماذج الأسماء Name Forms

كما ذكرنا سابقاً الأسماء عبارة عن سلسلة من الحروف تستخدم لتعريف خاصية محددة في البرنامج.

في اللغات الأولى استخدمت حروف مفردة كما في الرياضيات لأن الخلفية لتلك اللغات كانت رياضية.

لغة FORTRAN I و FORTRAN 77 سمحت باستخدام أسماء حتى 6 حروف . ثم أتت بعد ذلك FORTRAN 99 و لغة C و لغة C++ لم تحدد أطوال محددة للأسماء.

لذا فنموذج الاسم المقبول هو سلسلة حروف بطول معقول مع حروف وصل مسموح باستخدامها مثل الشرطة النحتية (_) Under-score. في بعض اللغات مثل C و C++ و JAVA تعتبر حساسة لحالة الأحرف مثلاً Rose ليست هي rose.

2. الكلمات الخاصة Special Words

الكلمات الخاصة في لغات البرمجة تستخدم لجعل البرامج أسهل قراءةً بواسطة تسمية الأفعال التي يقوم بها البرنامج. وأيضاً تستخدم للتفريق بين الوحدات في البرنامج هجائياً.

تسمى الكلمات الخاصة في أغلب اللغات كلمات محجوزة Reserved Words ، و في البعض الآخر تسمى كلمات مفتاحية Keywords كما في لغة FORTRAN.

3. المتغيرات Variable

يعتبر المتغير في البرنامج تجريباً لخلية أو مجموعة خلايا في الذاكرة، كما ينظر أغلب المبرمجين للمتغيرات على أنها أماكن في الذاكرة ولكن في الحقيقة هنالك الكثير من المتغيرات أكثر من أنها مجرد أماكن في الذاكرة لتخزين البيانات ففي مرحلة التحول من لغة الأكثر إلى لغة التجمع ثم تبديل العناوين الرقمية المجردة للذاكرة بالأسماء، مما يجعل البرامج سهلة القراءة والكتابة والإصلاح . كما تم التخلص نهائياً من مشكلة العناوين الرقمية المجردة للذاكرة.

يميز المتغير بستة صفات هي الاسم ، العنوان ، القيمة ، النوع ، دورة حياة المتغير و المدى . في القسم التالي سوف نناقش صفات المتغيرات ببعض التفصيل :

أ/ اسم المتغير Variable Name

أسماء المتغيرات هي الأكثر استخداماً في البرنامج ، ويجب أن تبدأ بحرف. وتكون خليطاً من الحروف والأرقام والعلامات مثل (_) ويجب أن يكون لها طول محدود.

ب/ العنوان Address :

عنوان المتغير هو العنوان في الذاكرة الذي يرتبط بالمتغير . من الممكن أن يكون للاسم موقع واحد أو أكثر من موقع في الذاكرة في نفس الوقت، مثال ذلك إذا كان لدينا برنامج فيه برنامجان فرعيان sub 1 و sub 2 كل منهما يستخدم نفس الاسم لتعريف متغير مثلاً sum لذا لأن كل برنامج فرعي مستقل عن الآخر فالرجوع لـ sub1 غير الرجوع لـ sub2 .

جـ / نوع المتغير Data Type

يحدد نوع المتغير القيم التي من الممكن أن يحملها المتغير بالإضافة إلى مجموعة العمليات التي يمكن إجراؤها على هذه القيم . مثال ذلك في لغة FORTRAN النوع INTECGER في بعض الأحيان . يحدد القيم من 32,758 - إلى 32,767 مع عمليات الجمع والطرح والضرب والقسمة وبعض عمليات المقارنة المكتوبة مثل القيمة المطلقة .

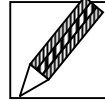
د/ القيمة Value :

قيمة المتغير هي محتوى خلية الذاكرة التي ترتبط بالمتغير . يمكن النظر إلى خلية الذاكرة كخلية مجردة. ثم تعرف خلية مجردة عمل الحجم المراد بواسطة المتغير . مثلاً المتغير من نوع Floating point يحتاج إلى 4 بايتات في الذاكرة.

تسمى قيمة المتغير بالقيمة اليمين R .Value لأنها تكون في الجهة اليمنى صيغة الإسناد في البرنامج .

تدريب (1)

أي العبارات التالية مقبولة كأسماء في لغات البرمجة :
Ahmed Omer , Student – of – open – university , 123 –
Max , Max 123 , Computer , AND



4. مفهوم الربط Concept of Binding

في الصورة العامة الربط عبارة عن رابطة بين خاصية وفئة أو بين عملية ورمز والزمن الذي يتم فيه هذا الربط يسمى زمن الربط Binding time ، لذا فإن الربط وزمن الربط من المفاهيم المهمة في مدلولات لغات البرمجة.

يمكن أن تتم عملية الربط في زمن تصميم اللغة Language design time ، زمن تطبيق اللغة Language Implantation time ، زمن الترجمة Compile Time ، زمن التحميل load time ، أو زمن التنفيذ Run time ، مثلاً رمز النجمة (*) Asterisk يتم ربطه مع عملية الضرب في زمن تصميم اللغة، أما نوع البيانات Integer في لغة FORTRAN يربط مع القيم المحتملة في زمن تطبيق اللغة.

وبالنظر إلى الوحدة البرمجية التالية من لغة C حيث يتم إسناد قيمة إلى المتغير count

```
Int count;  
Count = count +5;
```

فالذي يحدث كالآتي بالنسبة لعملية الربط وزمن الربط :

المجموعة المحتملة من النوع للمتغير count : تربط في زمن التصميم.

- نوع count : تربط في وقت الترجمة.

- مجموعة القيم المحتملة لـ count : تربط في زمن تصميم المترجم.

- قيمة count : تربط في زمن التنفيذ مع الكود.

- مجموعة المعاني للعلامة(+) : تربط في زمن تعريف اللغة.

- معنى العلامة(+) : تربط في زمن الترجمة.

من الضروري فهم متى يتم ربط خواص الوحدات الموجودة في البرنامج لكي نتمكن من فهم مدلولات لغات البرمجة مثال ذلك لفهم عمل دالة (function) في برنامج معين لابد

من فهم كيف يتم ربط البارامترات الحقيقية عند استدعاء الدالة بالبارامترات في تعريف الدالة .

وأيضاً لتحديد القيمة الحالية لمتغير تحتاج أن تعلم متى يتم ربط هذا المتغير بمكان التخزين.؟

1.4. ربط النوع Type Binding

قبل استخدام متغير معين في برنامج، يجب أن يربط إلى نوع بيانات (data type) .
المهم في الأمر هو أمران الأول هو كيف نحدد نوع البيانات والثاني هو متى يتم الربط.
يمكن أن يحدد النوع في السكون عن طريق تصريح مباشر أو غير مباشر.
أ- الربط الساكن static binding :

يمكن التصريح عن المتغيرات بإحدى طريقتين :

1- التصريح المباشر Explicit Dedaration

هو عبارة عن صيغة في البرنامج يذكر فيها أسماء المتغيرات ونوع بياناتها .

2- التصريح غير المباشر Implicit Dedication

هو عبارة عن رابطة بين المتغيرات والنوع عبر العلاقة الافتراضية في الصيغة، في هذه الحالة أول ظهور لاسم المتغير يعتبر هو التصريح غير المباشر لهذا المتغير .
وفي الحالتين ينشأ نوع من الربط الساكن بين المتغير والنوع. أغلب اللغات التي تم تصنيعها قبل منتصف الستينيات من القرن الماضي تحتاج إلى تصريح مباشر للمتغيرات ، أما اللغات التي تم تطويرها بعد ذلك تحتاج إلى تصريح غير مباشر مثال ذلك Fortran و Basic.

ب-الربط غير الساكن أو الديناميكي Dynamic Binding

في هذا النوع من الربط نجد أن نوع البيانات لا يحدد بواسطة صيغة تصريح بل بربط المتغير بنوع البيانات عندما تسند له قيمة في صيغة الإسناد وعندما يتم تنفيذ صيغة الإسناد تتم عملية ربط المتغيرات في صيغة الإسناد لقيمة النوع ، أو متغير، أو تغير في الجهة اليمنى من صيغة الإسناد.

الفائدة المهمة للربط غير الساكن أن هنالك مرونة في البرمجة مثال ذلك البرنامج الذي يكتب بلغة تتعامل مع الربط غير الساكن يمكن استخدامه مع أي نوع بيانات - و السبب في ذلك لأن المتغير يتم ربطه مع أي نوع بيانات أثناء التنفيذ.

والعكس صحيح في اللغات التي تتطلب التصريح الساكن لابد من تحديد نوع البيانات مسبقاً مثل لغة C و Pascal ولكن هنالك مساوئ أيضاً للتصريح غير الساكن وهي:

- لا يمكن تتبع الأخطاء في التصريح غير الساكن، لأن نوعي البيانات يمكن أن يظهر في الجانبين من صيغة الإسناد ، النوع الخاطئ في يمين صيغة الإسناد لا يمكن اكتشافه كخطأ، بل الجانب الأيسر يتغير إلى النوع الخاطئ .
- التكلفة هي أيضاً من مساوئ التصريح غير الساكن - لأن من الضروري استخدام أوعية تخزين متغيرة الحجم لعملية التحقق من النوع في زمن التشغيل لأن كل نوع يتطلب حجم تخزين مختلف، بالإضافة إلى ذلك فإن اللغات التي تستخدم التصريح غير الساكن تستخدم متغيرات Interpreter وليس مترجمات Compilers لأن عملية ربط النوع للمتغيرات لا يمكن القيام بها في لغة الأكثر.

ج- ربط التخزين ودورة حياة المتغير

من الخصائص المهمة للغات البرمجة هو تصميم ربط التخزين للمتغيرات، لذا من المهم فهم هذا الربط.

في العادة يتم تخصيص خلية أو يتم حجز مساحة من الذاكرة المتوفرة للمتغير فيما يسمى عملية حجز الذاكرة (allocation) عند عملية الربط ، ثم يتم تحرير هذه الخلية بعد فك ربط المتغير (Deallocation) وإعادتها إلى الذاكرة المتوفرة.

دورة حياة المتغير هي عبارة عن زمن ربط المتغير بخلية محددة في الذاكرة والتي تبدأ مع بداية ربط المتغير مع خلية الذاكرة وتنتهي بنهاية عملية ربط المتغير مع تلك الخلية .

تنقسم المتغيرات حسب ربط التخزين إلى أربعة أنواع وهي:

- 1- المتغيرات الساكنة Static Variables
- 2- المتغيرات ديناميكية المكس stack dynamic
- 3- المتغيرات ديناميكية التكوين المباشرة Explicit heap dynamic
- 4- المتغيرات ديناميكية التكوين غير المباشرة Implicit heap dynamic

أولاً: المتغيرات الساكنة:

هي المتغيرات التي يتم ربطها بالذاكرة قبل بداية تنفيذ البرنامج حتى أثناء تنفيذ البرنامج.

فوائد استخدام المتغيرات الساكنة :

من فوائد استخدام المتغيرات الساكنة التالي:

- [1] الفائدة الأولى للمتغيرات الساكنة عند استخدام المتغيرات العامة في البرنامج أنها تحتاج إلى توافرها طوال فترة تنفيذ البرنامج، لذا من الضروري ربطها بنفس التخزين طوال فترة التنفيذ.

[2] الفائدة الأخرى للمتغيرات الساكنة هي الكفاءة لأن كل العناوين للمتغيرات الساكنة هي مباشرة ، مما يقلل زمن التشغيل عند حجز مساحة لها في الذاكرة المتوفرة.

مساوئ المتغيرات الساكنة :

هنالك بعض المساوئ ألا وهي قلة المرونة لأنها لا تدعم البرمجة الفرعية المتتالية. وأيضاً التخزين لا يمكن أن يشارك بين المتغيرات ، مثلاً إذا كان لدينا برنامج به برنامجان فرعيان يتطلب كل واحد منهما مصفوفة كبيرة غير مرتبطة، فإذا كانت ساكنة لا يمكن أن يستخدم التخزين .

في لغة Fortran 1,11,17 كل المتغيرات ساكنة أما في لغة C ، + C و Java يمكن أن نستخدم كلمة static لتعريف المتغيرات الساكنة، أما pascal لا يوجد بها متغيرات ساكنة .

ثانياً: المتغيرات ديناميكية المكس:

هي المتغيرات التي ينشأ ربط التخزين لها أثناء نشر صيغة التصريح ولكن مع ربط ساكن للنوع.

حيث يتم النشر أثناء زمن التشغيل

في لغة pascal يتكون البرنامج الفرعي من قسم التصريح (Dedaration) وقسم الشفرة (code)، حيث يتم نشر قسم التصريح قبل بداية تنفيذ الشفرة مباشرة وحجز مساحة للمتغيرات طوال فترة التنفيذ ثم يتم تحرير المساحة بعد أن يُرجع البرنامج الفرعي (الإجراء) التحكم إلى منادي الإجراء.

ثالثاً: المتغيرات ديناميكية التكوين المباشرة

هي عبارة عن خلايا في الذاكرة بدون اسم يتم حجزها وتحريرها بواسطة تعليمات تشغيل مباشرة من البرنامج .

تستخدم المؤشرات "Pointers" والمراجع "Reenences" للتعامل مع هذه الخلايا.

تنشأ هذه الخلايا بواسطة معامل كما في + C و Java أو بواسطة استدعاء إلى إجراء في النظام كما في لغة C.

مثال ذلك في لغة ++C:

في هذا المثال لقد تم إنشاء متغير من نوع ديناميكي التكوين بواسطة المعامل new ثم استخدمنا المؤشر int note كمرجع للمتغير ثم أخيراً تم تحرير الخلية باستخدام المعامل delete

في لغة Java ليس هنالك طريقة مباشرة لهدم أو تحرير الخلية لذلك تستخدم طريقة جمع القمامة garbage collection من مساوئ المتغير ديناميكي التكوين هو أنه من الصعوبة

استخدام المؤشرات والمراجع بالإضافة إلى تكلفة استخدام المراجع إلى المتغيرات ، حجز الخلية وتحرير الخلية .

ثالثاً: المتغيرات ديناميكية التكوين غير المباشرة

هي التي يتم ربطها مع التكوين عند استخدام قيمة لها . بمعنى أنها أسماء تتأقلم مع أي عمل يطلب منها القيام به . وفائدتها أن لها أكمل درجة من المرونة . ومن مساوئها أنها تحتاج إلى زمن طويل لإصلاحها. كما أن هنالك بعض الصعوبة في تحديد أخطائها بواسطة المترجم .

أسئلة التقويم الذاتي



1) اذكر متطلبات تصميم الأسماء و نماذجها.

2) عرف الآتي

أ) الكلمات الخاصة.

ب) المتغيرات.

3) ما نعني بمفهوم الربط؟

5. التحقق من النوع Type Checking

عزيزي الدارس ، عملية التأكد من أن نوع المؤثر لكل معامل متطابق (Compatible).

نوع متطابق:

تعني أنه شرعي لهذا المعامل، أو أنه حسب قواعد اللغة المعينة يمكن أن يحول إلى نوع شرعي لهذا المعامل بواسطة المترجم تلقائياً.

خطأ النوع Type Error

هو علمية تطبيق معامل على Oprands من نوع غير مناسب

تتم عملية التحقق من النوع كالتالي :

- إذا كانت عملية المتغير للنوع ساكنة إذاً فإن عملية التحقق تكون ساكنة أما إذا كانت عملية الربط غير ساكنة أو ديناميكية إذا يتم التحقق أثناء زمن التشغيل ويسمى التحقق الديناميكي.
- التحقق الساكن أفضل من التحقق الديناميكي وذلك لأنه من الأفضل والأسهل اكتشاف الأخطاء في مرحلة زمن الترجمة من اكتشافها في مرحلة زمن التشغيل .

ملحوظة:

هنالك بعض اللغات مثل APL و SNOBOL4 تسمح فقط بالتحقق الديناميكي وذلك لأن ربط النوع فيها ديناميكي . يمكن أن يكون التحقق من النوع معقد جداً في حالة اللغات التي تسمح لخلايا الذاكرة بتخزين قيم مختلفة في أزمان مختلفة أثناء زمن التنفيذ كما في Pascal ، Ada ، + C و C في هذه الحالة يجب أن يكون التحقق من النوع ديناميكي كما يتطلب من النظام المحافظة على نوع الخلية الحالي.

ب- تطابقية النوع Type Compatibility :-

هي عبارة عن قواعد وضعت للاستفادة من تطابق متغيرين لاستخدامه مكان الآخر . لذا المعرفة التامة بهذه القواعد للغة المحددة تساعد في القدرة على كتابة وقراءة البرامج في هذه اللغة .

يوجد نوعان من تطابق النوع وهما:

أ- تطابقية اسم النوع

ب- تطابقية بنية النوع

أولاً: تطابقية اسم النوع

القاعدة :

كل متغيرين متطابقين في النوع إذا كانا في نفس التصريح أو كانا في تصريحين يستخدمان نفس اسم النوع.

مثال ذلك كما في الشفرة البرمجية التالية في لغة Pascal

Type index type = 1... 100

Var

Count : integer

Index: index type

المتغيران Count و index ليسا متطابقين لذا لا يمكن إسناد أحدهما للآخر .

ثانياً: تطابقية بنية النوع

القاعدة:

يكون المتغيران متطابقين إذا كان نوعهما لهما نفس البنية،

مثال ذلك كما في الوشفرة البرمجية التالية :

Type degree = real,

Grade =real,

لذا فإن المتغيرين degree و grade متطابقان من حيث البنية

6. مدى المتغير Variable Scope

مدى المتغير هو عبارة عن مجال الصيغ التي يمكن أن يرى فيها المتغير.

القاعدة:

يمكن أن يكون المتغير في صيغة معينة إذا كان من الممكن استخدام هذه الصيغة.

قواعد المجال: للغة تحدد كيف أن الحدوث أو الظهور الخاص للاسم مرتبط مع المتغير . وبصورة خاصة يحدد كيف أن استخدام متغير التصريح عنه خارج الإجراء أو أن البرنامج الفرعي الذي يتم تنفيذه حالياً مرتب مع التصريح عنه وكذلك مع خصائصه .

المثال التالي يوضح ذلك:

```
Begin {sub1}
```

```
... × ...
```

```
end sub1
```

```
.....
```

```
End , {main}
```

المرجع لـ × في sub1 هو × المصرح عنه في sub1 ، لذا فإن × في المصرح عنه في main مخفي عن البرنامج الفرعي sub1 وبصورة عامة التصريح عن متغير مخفي بصورة تامة أي تصريح آخر لمتغير بنفس الاسم في الشفرة البرمجية الأكبر .

ملحوظة:

في لغتي C و C++ لا يوجد تداخل لبرامج فرعية ولكن يوجد ما يسمى بالمتغيرات العامة والتي يصرح عنها خارج البرنامج الفرعي وفي المقابل توجد متغيرات محلية تخفي هذه المتغيرات العامة. حيث يمكن استخدامها كما في C++ باستخدام معامل المدى (::) scope operator.

1.6 الكتلة Blocks

أغلب اللغات تسمح بتعريف مدى ساكن جديد في وسط برنامج . هذا المفهوم القوي تم استحداثه في لغة Algol 60 ، حيث تسمح لقسم من البرنامج بامتلاك المتغيرات المحلية الخاصة بها بأقصر مدى هذه المتغيرات هي ديناميكية المكس، لذا تتصف بتخزين يحجز في حالة الدخول إلى قسم البرنامج وتحرر في حالة الخروج من قسم البرنامج - ويسمى هذا القسم الكتلة Block .

2.6. أنواع المدى

هنالك أنواع مختلفة من المدى هي:

1- المدى الساكن Static scope

2- الكتلة Block

3- المدى الديناميكي Dynamic scope

أولاً : المدى الساكن:

وهو المدى الذي فيه يحدد المتغير قبل التنفيذ لذا سمي ساكناً
لغة ALGOL 60 هي اللغة الأولى التي استخدمت المدى الساكن ثم بعد ذلك تبعتها اللغات الأخرى.

في أغلب اللغات يرتبط كل مدى ساكن مع وحدة التعريف في البرنامج، حيث تقوم البرامج الفرعية بإنشاء المدى خاصتها ، حيث يمكن عمل تداخل من البرامج الفرعية مع بعضها البعض ما عدا في C و C++ و Fartran و Java مكونة مدى هيكلي في البرنامج.

القاعدة:

في اللغات ساكنة المدى عندما يجد المترجم مرجعاً لمتغير يقوم بتحديد صفات المتغير بواسطة البحث في صيغة التصريح عن المتغير . في اللغات ساكنة المدى ذات البرامج الفرعية المتداخلة يقوم المترجم بعمل الآتي:

بافتراض أن المرجع للمتغير \times عمل في البرنامج الفرعي sub1 التصريح الصحيح يتم بواسطة البحث أولاً في البرنامج الفرعي sub1 فإذا لم يوجد البحث يستمر في البرنامج الفرعي الذي صرح فيه ، عن sub1 ، فإذا لم يوجد يستمر البحث إلى الوحدة الأكبر التالية وهكذا إلى أن يوجد التصريح عن \times ، وإلا سوف يكتشف خطأ عدم تصريح عن متغير.
مثال ذلك كما في الوحدة التالية عن لغة Pascal:-

```
Procedure big ;  
Var  $\times$  : integer;  
Procedure sub1 ;  
Begin D{ sub1} ;  
...  $\times$  .....  
End ;{sub1}  
Procedure sub2  
Var  $\times$  : integer ;  
Begin {sub2}
```

```

.....
End {sub2}
Begin {big}
.....
end ; {big}

```

استخدم المتغير \times في sub1 لذا نبحث عنه في sub1 ولكنه غير موجود ، لذا سيبحث عنه في big وهو موجود هنالك.

ملحوظة:

اللغات التي تستخدم المدى الساكن بعض تصريح المتغيرات تكون مخفية عن بعض البرامج الفرعية كما في المثال التالي من لغة Pascal:

```

program main ;
Vara ; integer ;
Procedure sub1;
Vara: integer;

```

في لغة Ada يتم تحديد الكتلة بواسطة declare كما في المثال التالي :

```

.....
Declare TEMP : integer ;
begin
TEMP := FIRST ;
FIRST := SECOND ;
SECOND := TEMP ;
end ;
.....

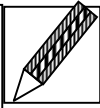
```

ثانياً: الكتلة هي الأصل في تسمية اللغات هيكلية الكتلة مثل لغة pascal .

اللغات C ، C + + و Java تسمح لأي صيغة مركبة compound statement لتعريف مدى جديد ، لذا فإن هذه الصيغ المركبة تعتبر مربعاً.

تدريب (2)

الوحدة البرمجية التالية مكتوبة بلغة C++ ، أي المتغيرات متطابقة ؟



```
in clued < iostream . h >
int x , y , z
float t , m
int add x y ( int , int , floalt multiply ( float , float ) , int main c )
{
add x y
multy tm ,
{
int add x y ( x , y )
int z ,
z = x + y ,
return z ,
float multitm ( t , m )
{
float l ,
{
l = t + m
retun ( l ) ,
{
```

الصيغة المركبة :compound statement

عبارة عن مجموعة من الصيغ داخل { } .
مثالاً لذلك التالي:

```
If (list [i] < list [j] )
{
Int temp
temp = list [i];
list [i] = list [j];
list [j]; = temp ;
}
```

ثالثاً: المدى المتحرك Dynamic Scope :

المدى المتحرك يعتمد على تتابع الاستدعاءات (callims) للبرامج الفرعية لذا لا يمكن تحديد المدى إلا في زمن التشغيل .

انظر إلى المثال التالي ثانية :

```
Procedure pig ;  
Var x : integer ;  
Procedure sub1 ;  
begin {sub1}  
....x.....  
end ; {sub1}  
Procedure sub2 ;  
Var x : integer ;  
begin {sub2}  
....  
end ; {sub2}  
begin {big}  
....  
end ; {big}
```

بافتراض أن قاعدة المدى المتحرك طبقت على المتغيرات غير المحلية أي أن معنى x في sub1 لا يمكن تحديده في زمن الترجمة .

لتحديد معنى x في زمن التشغيل الآن نبحث في البرنامج الفرعي الذي قام باستدعاء sub1 ثم إلى البرنامج الأعلى إلى أن نجد التصريح عن x ، وإلا سوف ينتج عن ذلك خطأ زمن التشغيل .

لذا في حالة تم استدعاء sub1 من sub2 سوف يوجد التصريح عن x في sub2 وإلا فسوف يوجد التصريح عن x في sub2 ، أما في حالة تم استدعاء sub1 من big يوجد التصريح عن x في big ... وهكذا .

أسئلة التقويم الذاتي



- (1) وضح باختصار ما نعني بالتحقق من النوع.
- (2) عرف مدى المتغير.
- (3) اذكر أنواع المدى المختلفة.

الخلاصة

عزيزي الدارس، في هذه الوحدة تمت مناقشة المدلولات الأساسية للمتغيرات، حيث بينا طبيعة الأسماء و الكلمات الخاصة المستخدمة في لغات البرمجة. وأيضاً تمت دراسة خصائص المتغيرات من نوع وعنوان و قيمة. بالإضافة إلى الربط، و زمن الربط ثم بعد ذلك قمنا بدراسة عملية التدقيق النوعي و اختباره والنوع القوى و قوانين التوافق.

لمحة مسبقة عن الوحدة التالية

سوف تناقش الوحدة التالية مفهوم أنواع البيانات والخصائص العامة لأنواع البيانات الأولية. ثم بعد ذلك تناقش تصميم الترميز والأنواع الفرعية . ثم بعدها سوف نناقش أنواع البيانات الهيكلية، خاصة المصفوفات ، السجلات ، ومجموعة البيانات ، وأخيراً المؤشرات.

إجابات التدريبات

تدريب-1:

الأسماء المقبولة كتسمية في لغات البرمجة :

1. Student – of – open – university لأنها تبدأ بحرف وتستخدم علامة وصل

مصرح بها (-) .

2. Max 123 : تبدأ بحرف وعبارة عن خليط من الحروف والأرقام .

3. Computer : تبدأ بحرف وعبارة عن مجموعة من الحروف.

الأسماء غير المقبولة :

1. Ahmed Omer : تتكون من كلمتين .

2. 123 – Max : تبدأ برقم .

3. AND : عبارة عن كلمة محجوزة للحاسوب .

تدريب-2:

1/ المتغيرات x , y , z متطابقة في النوع ، لأنها من int .

2/ المتغيرات t , m , l : متطابقة أيضا في النوع لأنها من نوع float .

مسرد المصطلحات

الأسماء: Names

الاسم عبارة عن سلسلة من الحروف تستخدم لتعريف خاصية محددة في البرنامج.

الكلمات الخاصة Special Words.

الكلمات الخاصة في لغات البرمجة تستخدم لجعل البرامج أسهل قراءةً بواسطة تسمية الأفعال التي يقوم بها البرنامج. وأيضاً تستخدم للتفريق بين الوحدات في البرنامج هجائياً.

المتغيرات Variable :

يعتبر المتغير في البرنامج تجزئاً لخلية أو مجموعة خلايا في الذاكرة

مفهوم الربط Concept of Biding :

في الصورة العامة الربط عبارة عن رابطة بين خاصية وفئة أو بين عملية ورمز والزمن الذي يتم فيه هذا الربط يسمى زمن الربط Brinding time لذا فإن الربط وزمن الربط من المفاهيم المهمة في مدلولات لغات البرمجة .

التحقق من النوع

هو عملية التأكد من أن نوع المؤثر لكل معامل متطابق (Compatible).

الصيغة المركبة compound statement

عبارة عن مجموعة من الصيغ داخل { } .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
71	المقدمة
71	تمهيد
71	أهداف الوحدة
72	1. أنواع البيانات الأولية
72	1.1. أنواع البيانات الرقمية Numeric Types
72	2.1. أنواع البيانات الحقيقية Floating point Data Type
73	3.1. الأعداد الكسرية Decimal
73	4.1. نوع البيانات الحرفية Character Type
73	5.1. نوع البيانات منطقي Boolean Type
73	6.1. نوع البيانات السلاسل الحرفية Character String Type
74	2. الأنواع العادية المعرفة
74	1.2. الترميز Enumeration
75	2.2. نوع المدى الفرعي
75	3. المصفوفات
76	1.3. ملاحظات حول المصفوفات
76	2.3. كيفية إسناد القيم للمصفوفة
76	3.3. العمليات على المصفوفات
77	4.3. شريحة المصفوفة Slice of Array
78	4. نوع السجلات
78	1.4. ملحوظات حول السجلات
78	2.4. أمثلة على السجلات
79	3.4. عملية الرجوع إلى السجل
79	5. المؤشرات
80	1.5. ملاحظات حول المؤشرات
80	2.5. أمثلة على المؤشرات
81	الخلاصة
81	لمحة مسبقة عن الوحدة التالية
82	إجابات التدريبات
83	مسرد المصطلحات
84	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الرابعة من مقرر "مفاهيم لغات البرمجة" سوف نناقش في هذه الوحدة أولاً مقدمة عن مفهوم أنواع البيانات والخصائص العامة لأنواع البيانات الأولية . ثم بعد ذلك نناقش تصميم enumerator's والأنواع الفرعية . ثم بعدها سوف نناقش أنواع البيانات الهيكلية خاصة المصفوفات ، السجلات ، والوحدة والنوع ومجموعة البيانات ، وأخيراً المؤشرات.

في كل نوع من أنواع البيانات سوف نناقش قضايا تصميم نوع البيانات، المعين وخيارات التصميم لكل لغة .

عملية تطبيق أنواع البيانات لها تأثير على تصميم نوع البيانات لذا سوف نناقش عملية التطبيق خاصة بالنسبة للمصفوفات.

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تتعرف على أنواع البيانات الأولية.
- تتعرف على أنواع البيانات المعرفة.
- تعرف المصفوفات و العمليات عليها وشريحة المصفوفة.
- تعرف السجلات و كيفية التعامل معها.
- تعرف المؤشرات.



تقوم الحواسيب باستخراج النتائج من إجراء العمليات على البيانات، لذا من الضروري أن تدعم اللغة أنواعاً مختلفة من البيانات حتى نستطيع أن نطابق هذه البيانات مع معطيات المشكلات الموجودة على الطبيعة .

في القسم التالي من الوحدة سوف نناقش كل أنواع البيانات العامة موضحين قضايا التصميم لكل نوع بيانات والعمليات المتوفرة للمتغيرات من كل نوع وكيف نحددها .

1. أنواع البيانات الأولية Primitive Data Type

عزيزي الدارس، أنواع البيانات الأولية هي أنواع البيانات التي لا تعرف بالنسبة لبيانات أخرى.

كل لغات البرمجة تحتوي على مجموعة من أنواع البيانات الأولية، وتستخدم هذه الأنواع الأولية مع نوع أو أكثر من منشئ النوع لتكوين أنواع البيانات الهيكلية، وتنقسم إلى: رقمي integer ، وصحيحة floating point وحرفية character ومنطقية Boolean

1.1. أنواع البيانات الرقمية Numeric Types

أكثر الأنواع شيوعاً هو نوع البيانات الرقمي integer. تدعم لغات البرمجة أحجام مختلفة من نوع البيانات الرقمي integer مثلاً في short integer ، long integer ، unsigned short integer ، unsigned long integer ، يمثل قيمة نوع البيانات " integer " في الحاسوب بسلسلة من البت مع حفظ آخر بت في اليسار لتمثيل الإشارة (+ أو -) لتمثيل الأرقام السالبة تستخدم صيغة كمية الإشارة بمعنى أن بت الإشارة تستخدم لتمثيل (-) وبت في مجموعة البت تستخدم لتمثيل القيمة المطلقة للرقم السالب .

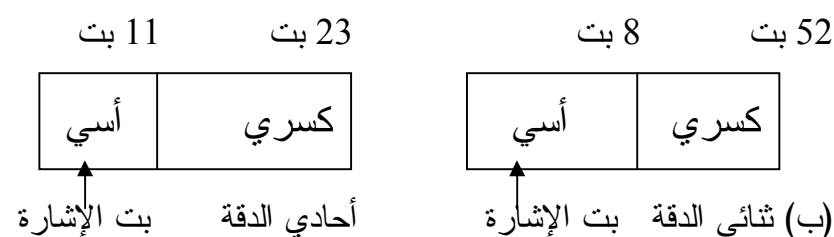
2.1. أنواع البيانات الحقيقية Floating point Data Type

تمثل الأعداد الصحيحة . ما التمثيل إلا تقريب لأغلب الأعداد الحقيقية. تخزن الأعداد الحقيقية في أغلب الحواسيب في شكل ثنائي هنالك بعض المشاكل التي تظهر عند استخدام تمثيل الأعداد الحقيقية وهي :

- عدم الدقة في العمليات الحسابية
- يتم تمثيل الأعداد الحقيقية كجزء كسري ، جزء أسي كما في التمثيل العلمي للأعداد

أغلب الحواسيب الحالية تستخدم تنسيق IEEE القياسي 754 كما في (3.1) يوجد نوعان من الأعداد الحقيقية - أحادي الدقة وثنائي الدقة وتخزن في 4 وحدات بايت من الذاكرة .

النوع ثنائي الدقة يستخدم في حالة إذا كان العدد لديه جزء كسري كبير



شكل (3.1) : تنسيق IEEE 754 القياسي

3.1. الأعداد الكسرية Decimal

تستخدم الأعداد Decimal لتخزين عدد أرقام عشرية ثابتة، مع الشرطة العشرية في مكان ثابت في القيمة، وتستخدم عادة في مجال الأعمال التجارية. تخزن الأعداد كما في السلاسل الحرفية باستخدام الشفرة الثنائية للأرقام فيما يسمى بالتمثيل المشفر وتحتاج إلى مساحة تخزين أكبر من التمثيل الثنائي.

4.1. نوع البيانات الحرفية Character Type

تخزن بيانات الحروف في الحاسوب في شكل شفرات أرقام أشهر طريقة للشفرة هي شفرة ASCII (American Standard Code for Information Interchange) والتي تستخدم الأرقام 0...127 لتشفير 128 حرفاً مختلفاً ثم أتت بعد ذلك طريقة جديدة سميت شفرة الكود الموحد (Unicode)، وهي تحتوي على الحروف من جميع اللغات الطبيعية. مثال ذلك أنها تحتوي على حروف السيرليك من العرب وأرقام التاي.

5.1. نوع البيانات منطقي Boolean Type

عزيزي الدارس، ان نوع البيانات منطقي هو أبسط أنواع البيانات لأنها تحتوي على قيمتين فقط هما عنصر للخطأ وعنصر آخر للصواب. حيث أن كل القيم غير الصفرية تعتبر صواباً (True) والصفر تعتبر خطأ (false) وأيضاً تستخدم نوع البيانات منطقي لتمثيل التبديل في البرامج يمثل النوع منطقي بوحدة بت مفردة .

6.1. نوع البيانات السلاسل الحرفية Character String Type

نوع السلاسل الحرفية هي التي تتكون من قيمة متتالية من الحروف، وتستخدم ثوابت السلاسل لتمثيل الإدخال والإخراج وكل أنواع البيانات . ويستفاد منها في عمليات معالجة الحروف .

عمليات السلاسل الحرفية

إذا لم تحفظ السلسلة في نوع بيانات أولي في الغالب تحفظ في مصفوفة من حروف. من العمليات التي تجرى على السلاسل هي عملية الإلصاق Concatenation كما في المثال التالي من لغة Ada.

في لغة Ada يستخدم الرمز (cf) للإلصاق السلاسل. المثال التالي يقوم بالإلصاق Name إلى : Name 2

Name 2 := Name1 + Name 2

إذا كان Name يساوي "Mohamed" , Name 2 يساوي Ahmed

إذا Name := Mohamed Ahmed

هناك عمليات أخرى تجرى على السلاسل في اللغات C , ++ C مثل :

- strcpy : نسخ السلاسل

- strcat : للإلصاق

- Strlen : توجد طول السلسلة

- Strncmp : تقوم بإجراء مقارنة بين الحروف في سلسلتين.

- في لغات Fortran و 90 fortran و Basic تعامل السلاسل كنوع بيانات أولى،
وتقوم بعمليات العلاقات ومعاملات المقارنة والإلصاق .

قضايا التصميم لأنواع البيانات Design Issues

أهم قضايا التصميم هي:

- هل يجب أن تكون السلسلة نوعاً خاصاً من مصفوفة حروف أو أنواع بيانات أولى.
- هل يجب أن يكون الطول ثابتاً أو غير ثابت

2. أنواع البيانات المعرفة Defined Ordinal Type

الأنواع المعرفة من البيانات هي التي يمكن فيها أن نربط القيم المحتملة مع الأعداد الصحيحة الموجبة.

في لغتي باسكال وآدا الأنواع المعرفة هي integer ، char ، Boolean .
هناك نوعان من البيانات المعرفة:

- النوع الأول : الترميز : Enumeration Type.

- النوع الثاني : المدى الفرعي subrange Type.

1.2. الترميز Enumeration

الترميز هو نوع البيانات الذي فيه كل القيم المحتملة (التي تصبح ثوابت رمزية) ويمكن تحويلها إلى أرقام في التعريف.

مثال : في لغة Ada:

Type Days is (Mon , Tue , Wed , Thu , Fri , Sat , Sun);

ملاحظات حول Enumeration :

- نوع Enum توفر فوائده في حالتي القراءة والكتابة بالنسبة للبرنامج .
- ينصح باستخدام نوع enum كما في Ada × لتسهيل القراءة

2.2. نوع المدى الفرعي

المدى الفرعي عبارة عن متوالية من نوع البيانات العادي مثال ذلك :

– 12....14 مدى فرعين من نوع integer

في لغة باسكال يتم التصريح عن مدى الفرعي . كما في المثال التالي :

```
Type  
upper case A ..Z ;  
index = 1..100;
```

حيث أن :

Upper case : هو مدى فرعي من الحروف

Index : هو مدى فرعي من الأرقام

ملاحظات على نوع المدى الفرعي

– في لغة Ada يوجد المدى الفرعي في فئة تسمى النوع الفرعي sub Type

مثال ذلك : sub type week day is day range man Mon.. Fri;

Sub type index is integer range 1..100 ;

حيث إن:

Week day : مدى فرعي للأيام من الاثنين إلى الجمعة

Index : مدى فرعي للأرقام من 1 إلى 100

ب- يحسن النوع الفرعي طريقة قراءة البرنامج لأنها تحدد مدى قيم المتغير

3. أنواع البيانات مصفوفة Array Type

تعريف:

عزيزي الدارس، المصفوفة: عبارة عن مجموعة من العناصر من نوع واحد (مثلاً

integer) من حيث إن كل عنصر يحدد بموقعه داخل المجموعة مقارنة بالعنصر الأول .

لكل عنصر في المصفوفة في البرنامج مرجع في البرنامج يسمى الفهرس عبارة عن رقم

يوضح موقع العنصر داخل المصفوفة وهو يحسب أثناء زمن التشغيل لتحديد موقع العنصر

في الذاكرة .

مثلاً:

```
Int array_Name [index]
```


حيث إن:

Array_Name : متغير عبارة عن مجموعة من نوع int
[index] : الفهرس وهو عبارة عن ثابت.

1.3. ملاحظات حول المصفوفات

- أغلب برامج الكمبيوتر تحتاج إلى برمجة مجموعات من البيانات من نفس النوع وتجري عليها نفس العمليات، لذا تستخدم المصفوفات.
- يمكن أن تكون المصفوفة من بعد واحد أو متعددة الأبعاد .
- الصيغة المستخدمة لتمثيل المصفوفات تقريباً واحدة في كل لغات الكمبيوتر ألا وهو اسم المصفوفة متبوعاً بالفهرس داخل قوس مربع أو قوس دائري في بعض اللغات مثل Ada و Fortran .

2.3. كيفية إسناد القيم للمصفوفة

1- في لغة Fortan يتم إسناد القيم للمصفوفة كما في المثال التالي:

Integ list (3) .

Data list /0,1,2/

يتم إسناد قيم ابتدائية للمصفوفة list من القيم المحصورة بين / /

/2 في لغتي C و ++C يتم إسناد القيم الابتدائية للمصفوفة بالطريقة التالية:

Int list [] = { 4,5,7,12}

لغتي pascal و Ada لا تسمح بإسناد قيمة ابتدائية للمصفوفة

3.3. العمليات على المصفوفات

عزيزي الدارس، تتم عمليات الجمع (+) والطرح (-) والضرب (×) والإلصاق بالإضافة إلى عمليات الإسناد وتتم هذه العمليات على جميع المصفوفة على أنها وحدة واحدة. ويكون الناتج عن عبارة مصفوفة أيضاً أي وحدة واحدة .

ومثال ذلك،: في لغة Ada تستخدم (+) لعملية جمع مصفوفتين ويكون الناتج مصفوفة مثلاً :

A+B

4.3 شريحة المصفوفة Slice of Array

شريحة المصفوفة عبارة عن بنية جزئية من المصفوفة، مثلاً الصف الأول من المصفوفة A كما في الشكل (3.2) التالي:

A

شكل 3.2 شريحة من المصفوفة (A)

ويرمز لها برمجياً كالتالي :

A (1:3)

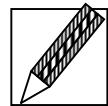
أما إذا كانت رأسية تكون كما في الشكل (3.3) التالي :

شكل 3.3 شريحة عمودية من المصفوفة (A)

ويرمز لها برمجياً كالتالي: A (1:3,2)

تدريب (1)

أنشئ مصفوفة باسم student - index مكونة من 12 عنصراً من نوع float



أسئلة التقويم الذاتي



1) ماذا نعني بالآتي:

أ) أنواع البيانات الأولية؟

ب) الترميز؟

ج) أنواع البيانات المصفوفة؟

2) عرف شريحة المصفوفة.

4. نوع البيانات سجل Record Type

عزيزي الدارس، نوع البيانات سجل هو عبارة عن مجموعة عناصر مختلفة من البيانات، حيث أن كل عنصر يعرف بإسم مختلف ويسمى الحقل .

مثال لذلك :

لتمثيل المعلومات حول طالب جامعي يتطلب ذلك وجود اسم الطالب ، رقم الطالب، مجموعة الطالب وهكذا.

حيث يمكن استخدام نوع البيانات حرف لتمثيل اسم الطالب ونوع البيانات رقم لتمثيل رقم الطالب والمجموع وهكذا.

1.4. ملحوظات حول السجلات

- 1- توجد السجلات في كل لغات البرمجة ما عدا النسخة الأولى من Fortran 90
- 2- في لغات البرمجة غرضية التوجه - نجد أن الفئات تدعم السجلات.
- 3- تختلف السجلات عن المصفوفات في أنها ليس من الضروري أن تحتوي على بيانات من نوع واحد. بالإضافة إلى أنها لا تمثل بواسطة الفهرس.
- 4 - تستخدم السجلات في حالة أن تكون هنالك مجموعة من البيانات من نوع مختلف.

2.4. أمثلة على السجلات

المثال الأول عن لغة COBOL:

```
01 Employee – Record
02 Employee – Name
05 First          Picture is × (20)
05 MIDDLE         Picture is × (10)
05 LAST           Picture is × (20)
20 HOURLY – RATE Picture is 99799.
```

حيث أن :

- 1- السجل Employee – Name يتكون من الحقلين Employee – Name و Hourly – Rate
- 2- الأرقام 05,02,10 تسمى أرقام المستوى، وهي تكون كل سطر في التعريف، وهي توضح موقع المتغير في هيكل السجل .

3- Picture : توضح تنسيق الحقل في السجل، مثلاً (20) × تعني الحقل مكون من 20 حقل، و 99799 تعني أن الحقل رقمي مكون من أربعة أرقام مع رقم بعد الفاصلة العشرية.

المثال التالي من لغة Ada وهو :

```
Employee Record :  
record :  
Employee Name :  
First : string (1..20) ;  
Middle : string (1..10);  
Last : string (1.....20);  
end record ;  
Hourly – rate : Float ;  
end Record ;
```

حيث إن:

– السجل : Employee مكون من سجلين فرعيين هما Employee Name الذي يتكون من: – الحقول : First ، Middle ، Last و Hourly rate من نوع Float.

3.4. عملية الرجوع إلى السجل

التعامل مع فعل في السجل نستخدم معامل النقطة(.) للرجوع لذلك الحقل فمثلاً Middle في المثال السابق نستخدم الجملة التالية:

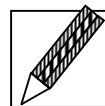
Employee – Record – Employee – Name. Middle

5. نوع المؤشرات Pointers Type

نوع المؤشرات عبارة عن متغير يحمل قيمة لعناوين في الذاكرة مع قيمة خاصة تسمى : nil

تدريب (2)

أنشئ سجلاً باسم stndint يقوم بحفظ بيانات الطلاب من اسم، ورقم، ونتيجة



عزيزي الدارس، تستخدم المؤشرات للأغراض التالية :

- تستخدم في العنونة غير المباشرة للمتغيرات كما في لغة أسمبلي
- تستخدم في إدارة الذاكرة الديناميكية .

1.5. ملاحظات حول المؤشرات

- أ- استخدام المؤشرات قد يؤدي إلى أخطاء برمجية كثيرة وخطيرة لذا فإن بعض اللغات مثل C++ و Java استحدثت المراجع (Refences) كبديل للمؤشران.
- ب- المؤشر المتوحش dangles : يقوم بتدمير الذاكرة ، وهو مؤشر يسند الى ذاكرة تم تحريرها.

2.5. أمثلة على المؤشرات

- 1- في لغة pascal كلمة new لإنشاء المؤشرات، وتستخدم dispose لتحرير الذاكرة من المؤشر .
 - 2- في لغة C و C++ :
 - أ- تستخدم علامة (*) لإنشاء المؤشر .
 - ب- تستخدم العلامة (&) لإنشاء العنوان في الذاكرة.
- كما في المثال التالي:

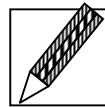
```
int * ptr ;  
int count , init ;  
.....  
ptr = & init ;  
count = * ptr ;
```

حيث إن :

- ptr* : هو مؤشر من نوع int يشير إلى عنوان
- count , Int : متغيران من نوع int
- &int : استخدمت لاسترجاع العنوان للمؤشر ptr

تدريب (3)

أنشئ مؤشرين من نوع float مع بيان عملية استرجاع العنوان للمؤشر .



أسئلة التقويم الذاتي

- (1) متى تستخدم السجلات؟
- (2) ما المواشرات؟ و فيم تستخدم؟



الخلاصة

عزيزي الدارس، هذه الوحدة بدأت أولاً بمقدمة عن مفهوم أنواع البيانات والخصائص العامة لأنواع البيانات الأولية، ثم بعد ذلك تم نقاش تصميم الترميز والأنواع الفرعية . ثم بعدها ناقشت الوحدة أنواع البيانات الهيكلية . خاصة المصفوفات ، السجلات ، والوحدة والنوع ومجموعة البيانات ، وأخيراً المؤشرات.

في كل نوع من أنواع البيانات قمنا بعرض قضايا تصميم نوع البيانات المعين وخيارات التصميم في كل لغة .

كما تبين لنا أن عملية تطبيق أنواع البيانات لها تأثير على تصميم نوع البيانات .

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس، البرامج الفرعية هي حجر الأساس في بناء البرامج، ولذا يعتبر تصميمها من أهم مفاهيم لغات البرمجة.

الوحدة التالية سوف تناقش تصميم البرامج الفرعية خاصة البارامترات ، طريقة التمرير ، بيانات المرجعية وغير المحلية ، التحميل الزائد للبرامج الفرعية ، الترجمة المستقلة والمشاكل التي تظهر مع البرامج الفرعية .

ثم بعد ذلك سوف نناقش طرق تطبيق البرامج الفرعية .

إجابات التدريبات

تدريب-1:

يمكن تعريف المصفوفة في لغة C++ بالصيغة التالية :

float studeot – index [11]

حيث إن [11] عبارة عن فهرس المصفوفة فالعنصر [0] هو العنصر الأول و [11] العنصر الثاني عشر في المصفوفة .

ثم بعد ذلك نقوم بإسناد القيم بالصيغة التالية :

studeot – index = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

حيث إن :

studeot – index [0] = 1

.

.

studeot – index [11] = 12

تدريب-2:

نقوم بإنشاء السجل في لغة Add كالآتي :

Student Record :

record :

student No : Number (1....10) , end record ,

student – Name : String (1 20) , end Record ,

Grade : float , end Record ,

تدريب-3:

float * ptr ,

float count , init ,

.....

p tr = & init ,

count = * ptr ,

مسرد المصطلحات

أنواع البيانات الأولية Primitive Data Type

أنواع البيانات الأولية هي أنواع البيانات التي لا تعرف بالنسبة لبيانات أخرى.

أنواع البيانات الحقيقية Floating point Data Type

تمثل الأعداد الصحيحة.

الأعداد الكسرية Decimal

تستخدم الأعداد Decimal لتخزين عدد أرقام عشرية ثابتة.

نوع البيانات الحرفية Character Type

تخزن بيانات الحروف في الحاسوب في شكل شفرات أرقام.

نوع البيانات منطقي Boolean Type

أبسط أنواع البيانات لأنها تحتوي على قيمتين فقط هما عنصر للخطأ وعنصر آخر للصح.

نوع البيانات السلاسل الحرفية Character String Type

نوع السلاسل الحرفية هي التي تتكون من قيمة متتالية من الحروف وتستخدم ثوابت السلاسل لتمثيل الإدخال والإخراج وكل أنواع البيانات . ويستفاد منها في عمليات معالجة الحروف .

أنواع البيانات المعرفة Defrnid Ordinal Type

الأنواع المعرفة من البيانات هي التي فيها يمكن أن نربط القيم المحتملة مع الأعداد الصحيحة الموجبة.

الترميز Enumeration

الترميز هو نوع البيانات الذي فيه كل القيم المحتملة (التي تصبح ثوابت رمزية) يمكن تحويلها

إلى أرقام في التعريف.

أنواع البيانات مصفوفة Array Type

المصفوفة: عبارة عن مجموعة من العناصر من نوع واحد (مثلاً integer) من حيث أن كل عنصر يحدد بموقعه داخل المجموعة مقارنة بالعنصر الأول .

نوع المؤشرات pointers Type

نوع المؤشرات عبارة عن متغير يحمل قيمة لعناوين في الذاكرة مع قيمة خاصة تسمى :
nil

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
87	المقدمة
78	تمهيد
78	أهداف الوحدة
88	1. أساسيات البرامج الفرعية
88	1.1. مواصفات البرامج الفرعية
88	2.1. التعريفات الأساسية
89	3.1. البارامترات Parameters
90	4.1. الإجراءات والدوال Procedure and Function
91	2. تصميم البرامج الفرعية
91	1.2. قضايا التصميم
91	2.2. المرجعية المحلية
93	3. طرق التمرير
93	1.3. نماذج مدلولات تمرير البارامترات
94	2.3. تطبيقات طرق التمرير
97	4. التحميل الزائد للبرامج الفرعية over loading sum program
98	الخلاصة
98	لمحة مسبقة عن الوحدة التالية
99	إجابات التدريبات
100	مسرد المصطلحات
102	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الخامسة من مقرر "مفاهيم لغات البرمجة" تعتبر البرامج الفرعية هي حجر الأساس في بناء البرامج، ولذا يعتبر تصميمها من أهم مفاهيم لغات البرمجة.

في هذه الوحدة سوف نقوم بمناقشة تصميم البرامج الفرعية، خاصة: الباراميترات ، طريقة التمرير ، بيئات المرجعية وغير المحلية ، التحميل الزائد للبرامج الفرعية ، الترجمة المستقلة والمشاكل التي تظهر مع البرامج الفرعية . ثم بعد ذلك سوف نناقش طرق تطبيق البرامج الفرعية .

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تتعرف على أساسيات البرامج الفرعية.
- توضح ما المقصود بالباراميترات.
- تشرح مفهومي الإجراءات و الدوال.
- تعرف طرق تمرير الباراميترات.
- تعرف التحميل الزائد للبرامج الفرعية.



1. أساسيات البرامج الفرعية

1.1. مواصفات البرامج الفرعية

عزيزي الدارس، كل البرامج الفرعية لها المواصفات التالية :

- 1- كل برنامج له نقطة دخول واحدة
- 2- يتم الخروج من البرنامج الرئيسي في حالة استدعاء البرنامج الفرعي، مما يعني أن هنالك برنامجاً فرعياً واحداً في وقت تنفيذ محدد
- 3- بعد انتهاء تنفيذ البرنامج الفرعي يرجع التحكم إلى المكان الذي تم فيه استدعاء البرنامج الفرعي .

2.1. التعريفات الأساسية

البرنامج الفرعي

يقوم بوصف واجهة البرنامج الفرعي وتجريد الأعمال التي يقوم بها البرنامج الفرعي .

استدعاء البرنامج الفرعي `subprogram call`

هو إصدار أمر إلى البرنامج بعملية تنفيذ البرنامج الفرعي .

رأس البرنامج الفرعي `Subprogram Header`

هي عبارة عن الأسطر الأولى من التعريف، وتقوم بالتالي :

أ- تحدد بأن الوحدة التي تليها عبارة عن تعريف للبرنامج الفرعي .

ب- تعطي أسماء للبرنامج الفرعي

ج- توضح البارامترات

مثال للرأس : في لغة باسكال

`Sub routine Adder (parameters) ;`

حيث إن :

- `subroutine` : كلمة خاصة يبدأ بها تعريف البرنامج الفرعي

- `Adder` : اسم البرنامج الفرعي

- `Parameter` : البارامترات في البرنامج الفرعي

أما في لغة C فلا توجد كلمة خاصة، حيث يتم التعرف على البرنامج الفرعي من الصياغ ، ويسمى دالة `Function` ، ومثال لذلك:

`.void adder (parameters)`

حيث إن :

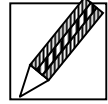
void : تعني أن الدالة لا ترجع قيمة.

adder : اسم الدالة.

Parameters الباراميترات للدالة .

تدريب (1)

اكتب رأس البرنامج الفرعي Header للبرنامج الفرعي MAX والتي تأخذ
ثلاثة متغيرات x , y , z من فرع float



3.1. الباراميترات (المعاملات) Parameters

عزيزي الدارس، هنالك طريقتان يستخدمهما البرنامج الفرعي للدخول إلى البيانات التي يتعامل معها وهما :

أ- الدخول المباشر إلى المتغيرات غير المحلية (يصرح عنها في مكان آخر).

ب- بواسطة تمرير الباراميترات.

البيانات التي تمرر بواسطة الباراميترات يمكن الدخول إليها عبر الأسماء التي تكون محلية بالنسبة للبرنامج الفرعي .

يوجد نوعان من الباراميترات :

1- الباراميترات الرسمية Formal Parameters

وهي الباراميترات التي توجد في رأس البرنامج الفرعي

2- الباراميترات الحقيقية Actual Parameters :

وهي الباراميترات التي تكون في صيغة الاستدعاء للبرنامج الفرعي

ملاحظة حول الباراميترات

- تتم عملية ربط الباراميترات الحقيقية مع الباراميترات الرسمية عبر عملية ربط بسيطة، حيث يتم ربط الباراميتر الحقيقي الأول مع الباراميتر الرسمي الأول، و الباراميتر الثاني مع الثاني وهكذا . وتسمى هذه الباراميترات الموضعية .

- في لغات ++ C و FORTRAN 90 و Ada الباراميترات الرسمية يمكن أن تحتوي على قيم افتراضية Default Vslues

القيمة الافتراضية

هي قيمة تستخدم في حالة أنه لم يتم تمرير باراميترات حقيقية إلى الباراميترات الافتراضية. و لتوضيح ذلك انظر المثال التالي، وهو مأخوذ من دالة في لغة Ada.

Function *compute - pay* (income : Float ;

EXEMPTION : integer : = 1;

Tax – RATE : Float) return float ;

حيث إن :

Compute – pay : هي دالة

Mac rate , income : هي بارمترات

البارمتر Exemption أعطى قيمة افتراضية وهي 1 في حالة استدعاء الدالة *compute*

pay – ولم يتم تمرير قيمة الدالة مكان Exemption فإن القيمة تستخدم ويتم الاستدعاء

كما في المثال التالي :

Pay : = compute –pay (2000.0,Tax – Rate = > 0.15 ;

ملاحظة:

– في لغة C++ القيم الافتراضية لابد أن تأتي في آخر قائمة البارمترات

4.1. الإجراءات والدوال Procedure and Function

تنقسم البرامج الفرعية إلى قسمين أساسيين هما : الإجراءات، والدوال.

أ. الإجراءات

الإجراءات عبارة عن تجميع لصيغ تعرف حوسبة معينة وتقوم الإجراءات بالآتي:

أ– يتم تنفيذها بواسطة صيغة استدعاء.

ب– تقوم بتعريف صيغ جديدة.

ج– تنتج نتائج في وحدة البرامج التي يتم منها الاستدعاء.

د– تقوم بتغيير المتغيرات في البرنامج أو في الإجراء.

ب. الدوال

الدوال هي شبيهة بالإجراءات ولكنها صممت بدلالة الدوال الرياضية، وهي لا تقوم بالآتي:

أ– لا تجري أي تعديل في المتغيرات التي تمرر عليها

ب– يتم استدعاؤها عند ظهور اسم الدالة في أي تغيير مع البارمترات

ج– القيمة المسترجعة من تنفيذ الدالة ترجع إلى البرنامج الذي تم الاستدعاء منه

مثال ذلك الدالة *power* التالية من لغة C:

Float power (float base, float exp);

ويتم استدعاء *power* كالآتي :

Result = 3.4 * *power* (10.0.x);

وتعطي النتيجة التالية:

Result = 3.4*10.0**X;

2. تصميم البرامج الفرعية

1.2. قضايا التصميم

تعتبر البرامج الفرعية من التراكيب المعقدة في لغات البرمجة، لذا فإن هنالك قائمة طويلة من قضايا التصميم ، إحدى هذه القضايا هو اختبار طريقة تمرير البارميترات أو الطرق التي يجب استخدامها . وقضية أخرى ذات صلة وهي هل من الضروري أن يتم اختيار نوع البارميترات الحقيقية بالنسبة لنوع البارميترات الرسمية المقابلة ؟ إن طبيعة البيئة المحلية للبرامج الفرعية تشابه إلى حد ما طبيعة البرامج الفرعية، لذا السؤال المهم هنا هو هل المتغيرات هي ساكنة أو أتوماتيكية ؟. بعض اللغات تسمح لأسماء البرامج الفرعية أن تمرر كبارميترات . هل من الممكن السماح بذلك في اللغة؟.

هل البرنامج الفرعي قابل للتحميل الزائد ؟
التحميل الزائد يعني أن هنالك أكثر من برنامج فرعي بنفس الاسم في نفس البيئة:
أخيراً في حالة تسمح اللغة بترجمة جزء من البرنامج الفرعي، ما هي الطريقة التي تستخدمها لذلك ؟

2.2. المرجعية المحلية

تسمح للبرنامج الفرعي بتعريف المتغيرات الخاصة به ، لذا تعرف بيئة مرجعية محلية.

المتغيرات التي تعرف داخل البرنامج الفرعي تسمى بالمتغيرات المحلية لأن التعامل معها مفيد فقط مع البرنامج الفرعي الذي عرفت فيه المتغيرات المحلية تكون نوعين

- أ- النوع الأول ساكنة.
- ب- النوع الثاني متحركة المكس.

ومواصفات كل منها كالآتي:

أ/ المتغيرات المحلية الساكنة :

- 1- ذات كفاءة عالية.
- 2- أسرع لأن الدخول إليها يتم مباشرة.
- 3- ليس هنالك مشكلة في زمن التشغيل في حال حجز وتحرير الذاكرة.
- 4- تسمح للبرامج الفرعية أن تكون حساسة للتاريخ.
- 5- من مساوئها عدم قدرتها على دعم التكرار.

ب/ المتغيرات المحلية ديناميكية المكس :

- 1- يتم فك ربطها مع التخزين عند بداية تنفيذ البرنامج الفرعي.
- 2- مرنة في التعامل.
- 3- تدعم التكرار في البرامج الفرعية.
- 4- تدعم مشاركة التخزين.

ومن مساوئها :

- أ- تحتاج إلى زمن أكبر في حجز وتحرير الذاكرة.
 - ب- الدخول إليها غير مباشر.
 - ج- لا تسمح للبرامج الفرعية أن تكون حساسة للتاريخ.
- في لغة 60 ALGOL المتغيرات المحلية تكون افتراضياً ديناميكية المكس في لغتي C و C++ تكون المتغيرات المحلية ديناميكية المكس إلا إذا صرّفنا عنها بواسطة الكلمة static فهي ساكنة .
- المثال التالي مأخوذ من لغة C و C++ يوضح أن المتغير sum ساكن والمتغير count ديناميكي المكس في الدالة adder .

```
int add ( int list [ ] int listlen)
{
    Static int sum = 0 ;
    int count ;
    for (count = 0 ; count < listlen , count ++ )
        sum += list ( count ++ )
    return sum ;
}
```

حيث إن :

Adder : دالة ترجع قيمة من نوع int

Sum: متغير محلي ساكن (static) من نوع int

Count: متغير محلي ديناميكي المكس افتراضياً (لم يذكر static)

في لغة pascal، Modulaz، Ada و java لا توجد غير المتغيرات المحلية ديناميكية المكس

أسئلة التقويم الذاتي



- 1) اذكر مواصفات البرامج الفرعية
- 2) اذكر طريقتين يستخدمهما البرنامج الفرعي للدخول إلى البيانات التي يتعامل معها.
- 3) عرف الإجراءات و الدوال.

3. طرق تمرير البارميترات

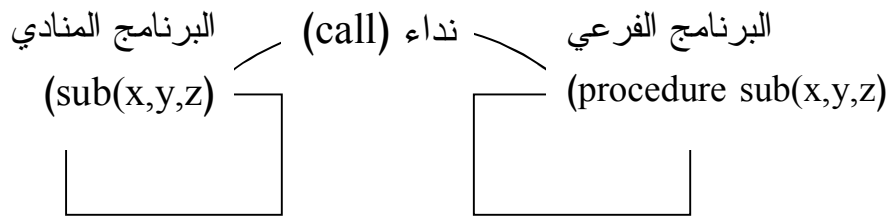
طرق تمرير البارميترات هي عملية انتقال البارامتر من أو إلى البرنامج الذي يقوم باستدعاء البرنامج الفرعي .

هنا أولاً سوف نناقش نماذج دلالات طرق تمرير البارميترات، ثم بعد ذلك الطرق المختلفة المستخدمة لتطبيق هذه النماذج، وأخيراً نأخذ بعض الأمثلة على تطبيق هذه النماذج.

1.3. نماذج مدلولات تمرير البارميترات

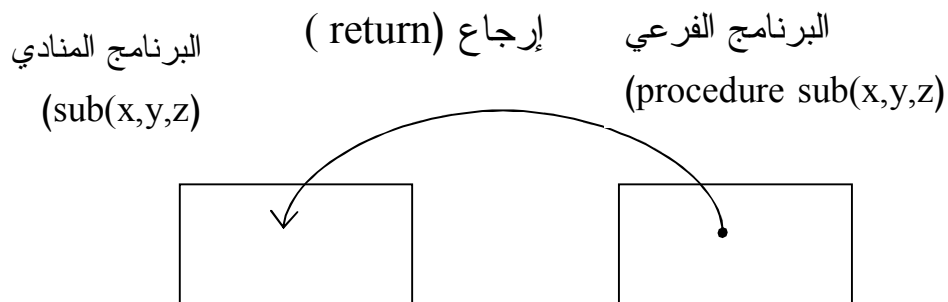
هنالك ثلاثة نماذج توصف طرق تمرير البارميترات هي كالآتي:

- 1- البارميترات الرسمية يمكن أن تستقبل البيانات من البارميترات المقابلة لها وهي كما في الشكل التالي : وتسمى الطريقة الداخلة :



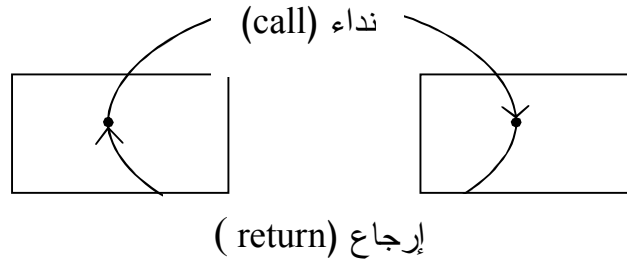
شكل (901) - الطريقة الداخلة

- 2- البارميترات الرسمية يمكنها أن تنقل البيانات إلى البلامترات الحقيقية كما في الشكل التالي : وتسمى الطريقة الخارجة :



شكل (902) الطريقة الداخلة

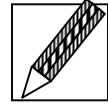
4- البارامترات الرسمية يمكن أن تنقل البيانات إلى البارامترات الحقيقية أو تستقبل البيانات من البارامترات الحقيقية . كما في الشكل التالي : وتسمى الطريقة الداخلة الخارجة.



شكل (903) الطريقة الداخلة الخارجة

تدريب (2)

أنشئ برنامجاً فرعياً في C أو C++ يوضح استخدام المتغيرات المحلية الساكنة، والمحلية ديناميكي المكوس .



2.3. تطبيقات طرق التمرير

عزيزي الدارس، هنالك نماذج عديدة طورت بواسطة مصممي اللغات المختلفة لتطبيق النماذج الثلاثة الأساسية . وهنا سوف نذكر بعضاً منها .

أ- التمرير بواسطة القيمة Pass-By-Value

عندما يمرر البارامتر بواسطة القيمة : يعني ذلك أن قيمة البارامترات الرسمية الحقيقية تستخدم لإعطاء قيمة ابتدائية للبارامترات الرسمية المقابلة ، التي تعمل بعد ذلك كمتغيرات عملية في البرنامج الفرعي .

ب- فوائد التمرير بواسطة القيمة

أ. ذات كفاءة عالية، لأن البيانات التي يتم تمريرها هي الحقيقة .

ب. لا يمكن فيها تعديل البيانات التي يتم تمريرها .

مساوئ التمرير بواسطة القيمة

أ. تحتاج البارامترات الرسمية إلى تخزين إضافي .

ب. تكلفة عالية في حالة كانت البارامترات كبيرة الحجم (المصفوفات).

ج- التمرير بواسطة النتيجة : Pass-By-Result:

عندما يمرر البارامتر بواسطة النتيجة يعني أنه لا يتم إنتقال القيم، حيث يعمل البارامتر الرسمي كمتغير محلي ويتم إرسال قيمة إلى البارامتر الحقيقي المقابل.

فوائد التمرير بواسطة النتيجة

لا يتم انتقال لقيم البارامترات الحقيقية .

مساوئ التمرير بواسطة النتيجة

1- في حالة استخدام أكثر من بارامتر يمكن أن يحدث تصادم بينها (sub (p₁,p₂).

2- هنالك صعوبة أن تكون البرامج انتقالية لوجود تعارض في أزمان التطبيق.

د- التمرير بواسطة القيمة المنتجة Pass-By-Value-Result

التمرير بواسطة القيمة المنتجة تعني أن قيمة البارامتر تستخدم لإعطاء قيمة ابتدائية للبارامتر الرسمي المقابل والذي يعمل لمتغير محلي، وعند إنتهاء البرنامج الفرعي يتم إرسال قيمة البارامتر الرسمي إلى البارامتر الحقيقي .

فوائد التمرير بواسطة القيمة المنتجة

يتم نسخ البارامتر الحقيقي إلى البارامتر الرسمي ثم يتم إعادة نسخها إلى البارامتر الحقيقي .

مساوئ التمرير بواسطة القيمة المنتجة

- تحتاج إلى تخزين كبير وزمن إضافي .

- مشكلة تعارض البارامترات (sub (p₁,p₂)

هـ- التمرير بواسطة المرجع sub. By. Ryarence

التمرير بواسطة الكتلة يعني : أن يتم إرسال عنوان البارامتر الحقيقي إلى البرنامج الفرعي مما يمكن البارامتر الرسمي من الدخول إلى البارامتر الحقيقي في البرنامج المنادي، مما ينتج عنه عملية مشاركة في البارامتر الحقيقي .

فوائد التمرير بواسطة القيمة المنتجة

- كفاءة عالية في عملية التمرير

مساوئ التمرير بواسطة القيمة المنتجة

1. عملية الدخول إلى البارامتر الرسمي بطيئة .

2. يحدث تعديل في البارامتر الحقيقي.

3. من الممكن حدوث تصادم بين البارامترات الحقيقية في حالة ارسال نفس البارامترات

مرتين مثال (fun (ftotal , total) .

6- أمثلة على طرق تمرير البارامترات

مثال (1) :

المثال التالي يوضح طريقة التمرير بالقيمة في برنامج من لغة C

أ. البرنامج الفرعي (الدالة) بالاسم swap1 كما يلي :

Void swap1 (inta , intb) حيث إن :

void swap1 { Int temp = a ;

a = b ;

a , b : بارامترات رسمية من نوع int

temp : متغير من نوع int ليستخدم لتبديل قيمة a بقيمة b

ب. يمكن تمرير البارامتر بالقيمة بواسطة صيغة المناداة التالية

swap (c , d) ;

حيث إن:

c , d : عبارة عن بارامترات حقيقية للدالة swap.

ج . ما يحدث بعد التمرير هو الآتي:

1- a = c : حرك قيمة البارامتر الأول

2- b = d : حرك قيمة البارامتر الثاني.

3- temp = q : ضع قيمة a في temp.

4- a = b : ضع قيمة b في a.

5- b = temp : ضع قيمة temp في b.

6- يتم تبديل q مع b.

7- البارامترات d و c لا تتغير.

مثال (2) :

المثال يوضح طريقة التمرير بواسطة المرجع في برنامج C حيث تتعامل مع المؤشرات

(pointers) كما يلي :

void swap 2(o int * a , int * b)

{ int temp = * a ;

* a = * b ;

* b = temp ;

حيث إن :

Swap 2 : عبارة عن دالة من نوع void

* a , * b : مؤشرات من نوع int وهي بارامترات رسمية للدالة swap 2
يتم استدعاء الدالة swap2 كما في الصيغة التالية :

Swap 2 (& C ,&d) ;

حيث إن :

&d , &C : عبارة عن بارامترات حقيقية تمرر بواسطة العنوان

يحدث التالي :

a = &c حرك البارامتر الأول إلى

b = &d : حرك البارامتر الثاني إلى

يتم تبديل قيمتي c , d

4. التحميل الزائد للبرامج الفرعية over loading sum program

عزيزي الدارس، التحميل الزائد عبارة عن أكثر من برنامج فرعي بنفس الاسم في نفس البيئة المرجعية، ولكن تختلف في التوقيع ، بمعنى أنها تختلف في عدد ، أو نوع ، أو ترتيب البارامترات أو في النوع الذي ترجعه

مثال :

المثال التالي يوضح كيف تصرح عن البرامج الفرعية زائدة التحميل في لغة C++

Void func(load a , float a) ;

Int func (int a , int b) ;

الدالة الأولى fun ترجع القيمة void.

الدالة الثانية : fun ترجع القيمة من نوع int.

لذا فهما دالتان بنفس الاسم ولكن تختلفان في القيمة المرجعية.

أسئلة التقويم الذاتي



(1) ما نماذج تمرير البارامترات؟

(2) عرف التحميل الزائد للبرامج الفرعية.

الخلاصة

عزيزي الدارس، في هذه الوحدة قمنا بمناقشة البرامج الفرعية و أثرها في بناء البرامج، حيث تعرضت الوحدة إلى تصميم البرامج الفرعية في لغات البرمجة. ثم بعد ذلك تمت مناقشة تصميم البرامج الفرعية، خاصة البارامترات ، طريقة التمرير ، بيئات المرجعية غير المحلية ، التحميل الزائد للبرامج الفرعية ، الترجمة المستقلة والمشاكل التي تظهر مع البرامج الفرعية . و أخيراً طرق تطبيق البرامج الفرعية .

لحة مسبقة عن الوحدة التالية

في الوحدة التالية سنتحدث عن التعبيرات و هي المعنى الأساسي لتعيين الحسابات في لغات البرمجة. من الصعب على المبرمج فهم كل من الصيغة اللغوية syntax والمعاني semantic للتعبير.

لذا من المهم فهم التقويم في التعبير من الضروري أن تكون ملماً بتقييم وترتيب العوامل والمؤثرات .

ترتيب معامل التقويم يكون تبعاً لقوانين العلاقة و الأسبقية للغة، وكذلك قيمة تعبير أحياناً اعتمد عليها .

ترتيب المعاملات في التعبير عادة غير محدد بواسطة مصمم اللغة ، الحالة التي تسمح لبرامج بإنتاج نتائج مختلفة لتطبيقاتها .

إجابات التدريبات

تدريب-1:

أولا : يكتب رأس البرنامج في برنامج باسكال كالآتي :

```
Subroutine Max ( float x , float y , float z ),
```

ثانيا في لغة :

```
void MAX ( float x , float y , float z ),
```

تدريب-2:

```
int add x ( int x , int y , int z )  
{  
    static int z = 0 ,  
    z = x + y ,  
    Return ( z ) .  
}
```


مسرد المصطلحات

البرنامج الفرعي

يقوم بوصف واجهة البرنامج الفرعي وتجريد الأعمال التي يقوم بها البرنامج الفرعي .

استدعاء البرنامج الفرعي subprogram call

هو إصدار أمر الى البرنامج بعملية تنفيذ البرنامج الفرعي .

البارامترات الرسمية Formal Parameters

وهي البارامترات التي توجد في رأس البرنامج الفرعي.

البارامترات الحقيقية Actual Parametrs

وهي البارامترات التي تكون في صيغة الاستدعاء للبرنامج الفرعي.

القيمة الافتراضية :

هي قيمة تستخدم في حالة أنه لم يتم تمرير بارامترات حقيقية إلى البارامترات الافتراضية.

الإجراءات :

الاجراءات عبارة عن تجميع لصيغ تعرف حوسبة معينة.

الدوال :

الدوال هي شبيهة بالإجراءات ولكنها صممت بدلالة الدوال الرياضية

المرجعية المحلية

تسمح للبرنامج الفرعي بتعريف المتغيرات الخاصة به ، لذا تعرف بيئة مرجعية محلية .

طرق تمرير البارامترات

طرق تمرير البارامترات هي عملية انتقال البارامتر من أو إلى البرنامج الذي يقوم باستدعاء البرنامج الفرعي .

التمرير بواسطة القيمة Pass-By-Value

عندما يمرر البارامتر بواسطة القيمة : يعني ذلك أن قيمة البارامترات الرسمية الحقيقية تستخدم لإعطاء قيمة ابتدائية للبارامترات الرسمية المقابلة ، التي تعمل بعد ذلك كمتغيرات عملية في البرنامج الفرعي .

التمرير بواسطة النتيجة : Pass-By-Result

عندما يمرر البارامتر بواسطة النتيجة يعني أنه لا يتم انتقال القيم، حيث يعمل البارامتر الرسمي

كمتغير محلي ويتم إرسال قيمة إلى البارامتر الحقيقي المقابل.

التمرير بواسطة القيمة المنتجة Pass-By-Value-Result

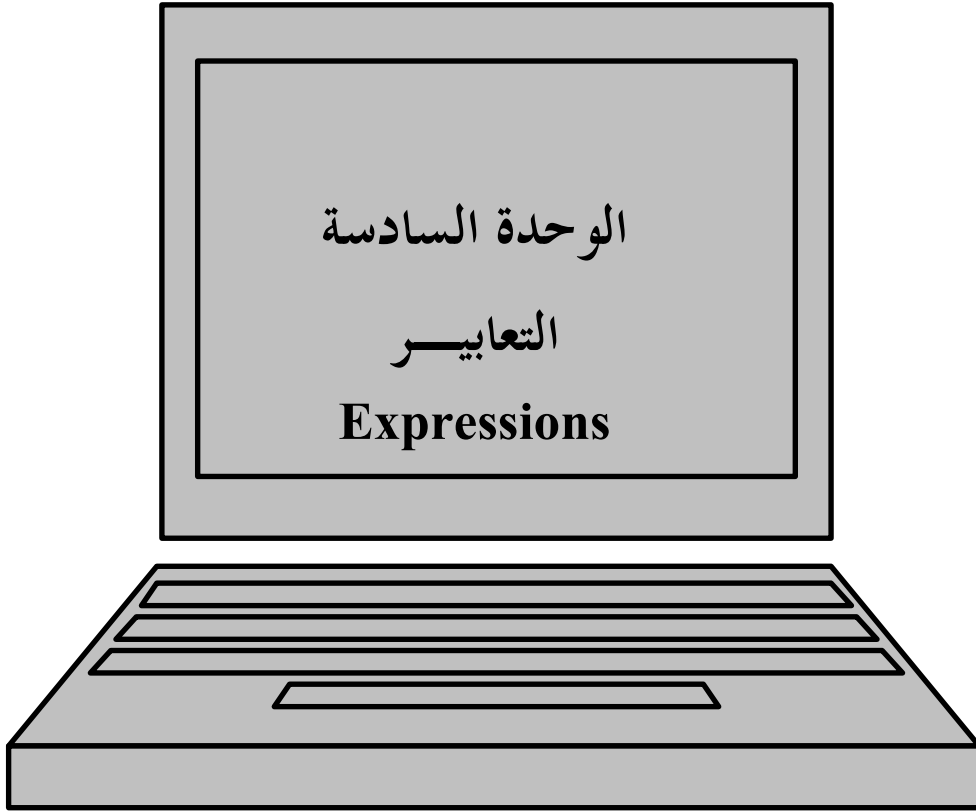
التمرير بواسطة القيمة المنتجة تعني أن قيمة البارامتر تستخدم لإعطاء قيمة ابتدائية للبارامتر الرسمي المقابل والذي يعمل كمتغير محلي، وعند إنتهاء البرنامج الفرعي يتم إرسال قيمة البارامتر الرسمي إلى البارامتر الحقيقي .

التمرير بواسطة المرجع sub. By .Ryarence

التمرير بواسطة الكتلة يعني : أنه يتم إرسال عنوان البارامتر الحقيقي إلى البرنامج الفرعي مما يمكن البارامتر الرسمي من الدخول إلى البارامتر الحقيقي في البرنامج المنادي، مما ينتج عنه عملية مشاركة في البارامتر الحقيقي .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
105	المقدمة
105	تمهيد
105	أهداف الوحدة
105	1. التعبيرات الرياضية
106	2. ترتيب تقويم المعاملات
106	1.2. الأسبقية Precedence
107	2.2. التشاركية Associatively
108	3. الأقواس Parentheses
109	4. التعبيرات الشرطية Conditional Expression
109	5. ترتيب تقييم المعاملات Operator Evaluation Order
110	6. التأثيرات الجانبية Side effects
110	7. التحميل الزائد للمعاملات Over loaded operators
110	1.7. ملاحظات حول التحميل الزائد للمعاملات
112	8. تحويل النوع Type Conversion
112	9. عملية التصحيح في التعبيرات
113	1.9. التحويل المباشر
114	10. الأخطاء في التعبيرات Errors in Expressions
115	الخلاصة
115	لمحة مسبقة عن الوحدة التالية
116	إجابات التدريبات
116	مسرد المصطلحات
117	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة السادسة من مقرر "مفاهيم لغات البرمجة".
التعابير هي المعنى الأساسي لتعيين الحسابات في لغات البرمجة. ومن الصعب على المبرمج فهم كل من الصيغة اللغوية syntax والمعاني semantic للتعابير .
لفهم التقويم في التعبير من الضروري أن تكون ملم بتقييم وترتيب العوامل والمؤثرات.
ترتيب معامل التقويم يكون تبعاً لقوانين العلاقة و الأسبقية للغة، وكذلك قيمة تعبير أحياناً اعتمد عليها .

ترتيب المعاملات في التعبير عادة غير محدد بواسطة مصممي اللغة ، الحالة التي تسمح لبرامج بإنتاج نتائج مختلفة لتطبيقاتها .
هدف آخر لمعنى التعبير هو عدم تجانس الأنواع .

جملة الإسناد يمكن ببساطة أن تتسبب في نسخ قيمة من خلية الذاكرة إلى أخرى، ولكن في حالات كثيرة جمل الإسناد تحتوي على تعبيرات مع مؤثرات المعاملات التي تتسبب في نسخ القيم إلى المعالج وتتم عمليات عليها والنتيجة تنسخ إلى الذاكرة مرة أخرى .

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تعرف التعابير الرياضية.
- تشرح أهمية ترتيب معمل التقويم.
- تعرف مفهوم التشاركية و قوانينها في لغات البرمجة.
- تشرح استخدام الاقواس و التعابير الشرطية وترتيب تقييم المعاملات و التأثيرات الجانبية للدالة.
- تعرف التحميل الزائد للمعاملات.
- تعرف عملية تحويل النوع.
- تدرك عملية التصحيح في التعابير بالإضافة للأخطاء في التعابير.



1. التعبيرات الرياضية Arithmetic Exp

- 1- أحد الأهداف الأساسية للغات عالية المستوى هو تقييم التعبيرات الحسابية كما في الرياضيات فمعظم خواص التعبيرات الحسابية في لغات البرمجة مشتقة من القواعد الرياضية المتعارف عليها مثل المعاملات، الأقواس، استدعاء الدوال.
- 2- المعاملات يمكن أن تكون أحادية، بمعنى أنها تحتوي على عامل واحد ، أو ثنائية بمعنى أنها تحتوي على عاملين.
- 3- اللغات C++ و C و Java تحتوي على معاملات ثلاثية. في معظم لغات البرمجة المعاصرة المعاملات الأحادية تكون في الوسط، بمعنى أنها تظهر بين عواملها. الاستثناء الوحيد هو لغة perl التي لها بعض المعاملات تكون في البداية بمعنى أنها تتقدم معاملها. مثال ذلك $+(a,b)$.
- 4- غاية التعبيرات الحسابية هو تعيين arithmetic coup تطبيق مثل هذه الـ cooptation يجب أن تسبب في حدثين : البحث عن العوامل (عادة من الذاكرة) وتنفيذ العملية الحسابية عليها.

2. ترتيب تقويم المعاملات Operator Evaluation Order

1.2. الأسبقية Precedence

عزيزي الدارس، قيمة التعبير تعتمد على الأول جزئياً ، على ترتيب معامل التقويم في التعبير، مثلاً اعتبر $A+B+C$ تختلف القيمة الناتجة على حسب اتجاه العمليات من اليمين للشمال أو العكس. وبدلاً من تقييم التعبيرات بهذه الطريقة الرياضية طوروا مفهوم وضع المؤثرات بتسلسل بتقسيم حسب الأولوية وأسسوا ترتيب التقييم للتعبيرات جزئياً على هذا التسلسل، مثلاً عملية الضرب لها أولوية على عملية الجمع.

1.1.2. ملاحظات حول الأسبقية

- 1- ثوابت أسبقية المعاملات للغات المعاصرة العامة تقريباً متشابهة لأنها كلها مبنية على الرياضيات . في هذه اللغات أعلى أسبقية للأس ثم يلي ذلك الضرب والقسمة في نفس المستوى، ثم يلي ذلك الجمع والطرح الثنائي في نفس المستوى .
- 2- معظم اللغات أيضاً تحتوي نسخة أحادية لعمليات الجمع والطرح . الجمع الأحادي يسمى معامل الهوية لأنه عادة ليس لديه عملية مصاحبة، وبذلك ليس له تأثير على عوامله .

3- في لغة Java الزائدة فعلياً له تأثير عندما يكون العامل short , byte char يتسبب في تحويل كامل للعامل إلى نوع int، بالطبع الطرح الأحادي دائماً يغير العامل ، ويغير علامة قيمة العامل .

1- في كل اللغات المعاصرة العامة مؤثر الطرح الأحادي يمكن أن يظهر في التعبير أما في البداية أو في أي مكان داخل التعبير طالما أنهى داخل أقواس لمنعه من مجاورة معامل آخر، مثلاً $A+(-B)*C$ ولكن $A+-B*C$ غير صحيحة

2.2. التشاركية Associatively

قوانين الأسبقية لا تقول شيئاً عن ترتيب تقييم المؤثرات في التعبيرات التي تحتوي على مؤثرات من نفس المستوى أسبقية مثل $A-B+C-D$.
قوانين التشاركية للغة هي التي تبين كيفية حساب مثل هذا التعبير فالمؤثر يمكن أن يكون لديه التشاركية من اليمين أو من الشمال، بمعنى أنه يتم تقييم الحدوث باعتبار أقصى اليسار أولاً أو أقصى اليمين.

1.2.2. ملاحظات حول التشاركية

1- في لغات البرمجة المعاصرة المعروفة التشاركية هي من اليسار لليمين ما عدا العامل الأساسي (إن وجد) يكون من اليمين إلى اليسار . في FORTRAN التعبير $A*B**C$ يتم تقييمه من اليمين إلى اليسار، ولكن في الـ Ada العامل الأساسي هو غير متشارك لذلك مثل التعبير الساري يعتبر غير صحيح وإنما يجب وضع أقواس لتدل على الترتيب المراد كالاتي $(A**B)**C$
2- المعامل الأحادي أسبقية غالباً غير مهمة. في لغة FORTRAN عمليات الطرح الاحادية والثنائية لهما نفس المستوى في الأسبقية، ولكن في Ada (ومعظم اللغات المعروفة) الطرح الاحادي له أسبقية على الطرح الثنائي ولأن الـ Ada تعطي الأسبقية للطرح الاحادي هذا التعبير يكافئ $B+(-A)$ في كل اللغتين السابقتين.

$$(1) A/B \quad (2) -A*B \quad (3) -A**B$$

في الحالتين (1) و(2) الأسبقية للطرح والمؤثر الثنائي ليس له علاقة ، قيمة التعبير لا تتأثر بتنفيذ أيهم أولاً . ولكن في حالة الأخيرة تفرق ، في كل لغات البرمجة المعروفة، ففي FORTRAN و Ada لديهم مؤثر أسي . وفي كلا الحالتين له أعلى أسبقية على الطرح الأحادي لذا $-A**B$ مكافئ لـ $-(A**B)$

3- عندما يظهر المؤثر العادي في مواضع غير النهاية اليسرى للتعبير يجب أن يكون بين قوسين في كلا اللغتين لذا في تلك الحالة المؤثرات تجبرنا ان يكون لها الأسبقية العليا.

2.2.2. قوانين التشاركية في لغات البرمجة Language Associatively rule

- أ- في APL كل المؤثرات لديها نفس الأسبقية، لذلك التقييم يعتمد على قانون associate وهو من اليمين إلى اليسار لكل المؤثرات .
- ب- معظم المترجمات تتغير من حقيقة أن بعض المؤثرات الحسابية تكون مرتبطة رياضياً. إن قوانين التشاركية ليس لديها أثر على قيمة التعبير الذي يحتوي على تلك المؤثرات فقط، مثلاً $A+B+C$ لا تعتمد على ترتيب تقييم المعامل
- ج- اذا كانت عمليات الـ floating point للعمليات الرياضية المرتبطة أيضاً متشاركة. المترجم يمكن أن يستخدم هذه الحقيقة لتأدية بعض simple optimization خاصة إذا كان مسموح للمترجم بإعادة ترتيب تقييم المعامل، يمكن أن يكون لديها المقدرة لإنتاج شفرات سريعاً لتقييم التعابير .
- د- في بعض العمليات يمكن أن يحدث overflow نسبة لمحدودية عمليات الحاسوب الحسابية. في هذه الحالة يمكن للحاسوب أن يغير ترتيب العمليات بحيث أن النتائج لا تسبب في overflow ولكن يؤدي إلى تغيير قيمة التعبير، وهذه مشكلة لحلها يمكن أن يستخدم المبرمجون الأقواس في التعبير حتى يصنفوا أن فقط الترتيب الصحيح للتقييم هو الممكن.

3. الأقواس Parentheses

المبرمجون يمكن أن يبدلوا قوانين الترابط والأسبقية بوضع أقواس في التعبير. الجزء الذي داخل أقواس في التعبير له الأسبقية على كل الأجزاء خارج الأقواس المجاورة له .

اللغات التي تسمح بالأقواس في التعبيرات الحسابية يمكن أن تعفى من كل قوانين الأسبقية وببساطة مشاركة كل المؤثرات من اليسار لليمين أو العكس .

المبرمج يقوم بتعيين الترتيب المراد للتقييم بواسطة الأقواس لذلك ليس على قارئ البرنامج تذكر أي من قوانين الأسبقية أو المشاركة.

من مساوي هذه الطريقة أنها تجعل كتابة التعابير أكثر ملاءمة.

4. التعبيرات الشرطية Conditional Expression

المؤثر : هو جزء من C++ , C و Java يستخدم لتشكيل التعبيرات الشرطية أحياناً جعله if the n – else تستخدم لتأدية عبارة إسناد شرطية مثلاً:

If (count = o)

Then average : = o

else average : = sum / count

في لغات Java , C++ , C, مثل هذا التعيين يمكن أن يتم باستخدام التعبير الشرطي

Exp1 ? exp2 : exp3

حيث إن :

exp1 هو عبارة Boolean إذا كان true فقيمة كل العبارة هو قيمة exp2 عدا ذلك تكون قيمة exp3, مثلاً if – then – else السابق يمكن أن يتحقق بالجملة الإسنادية التالية باستخدام التعبير الشرطي .

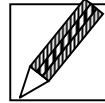
Average = (count == o) ? o : sum / count ;

بمعنى أن علامة الاستفهام تشير إلى بداية then cluse وعلامة (:) هي بداية Else clues

تدريب (1)

ما الفرق بين التعبيرين التاليين؟

$x * y - (x - y)$
 $x * y - x - y = 19$



أسئلة التقويم الذاتي

(1) ماذا نعني بالاسبقية؟

(2) ما مساوي استخدام الأقواس في كتابة البرامج؟



5. ترتيب تقييم المعاملات Operator Evaluation Order

عزيزي الدارس، المتغيرات في التعبير يتم تعيينها عن طريق البحث عن قيمتها في الذاكرة ، كذلك الثوابت أحياناً تقيم بنفس الطريقة، في حالات الثوابت يمكن أن تكون جزء من تعليمات لغة الآلة ولا تحتاج لبحث في الذاكرة إذا كان العامل في أقواس في التعبير عندئذ كل المؤثرات التي تحتويها يجب أن تقيم قبل أن يتم استخدام قيمتها كمعامل. و إذا لم يكن لأي من مؤثرات العامل تأثير جانبي عندئذ يكون ترتيب التقييم غير مهم .

6. التأثيرات الجانبية Side effects

التأثير الجانبي لدالة يحدث عندما تغير الدالة أما أحد معاملاتها parameters أو متغير عام global variable، مثلاً اعتبر التعبير $A + \text{Fun}(A)$ إذا كان Fun ليس لديها تأثير جانبي من تغيير (A) عندئذ ترتيب التقييم للعاملان A و $\text{Fun}(A)$ ليس لديه تأثير على قيمة التعبير . إذا Fun غيرته، عندئذ يوجد تأثير .

اعتبر الحالة التالية : Fun ترجع القيمة بعد القسمة على (2) وتغير المعامل بأن يكون له القيمة 20 : $B := A + \text{Fun}(A)$, $A := 10$

عند البحث عن قيمة A أولاً نجد لها 10 وقيمة التعبير 15 ، ولكن تم تقييم المؤثر التالي :
أولاً تكون قيمة المؤثر الأول وقيمة التعبير 25
* يوجد حلان للمشكلة :

أولاً : عدم السماح للتأثير الجانبي الوظيفي بالبحث عن قيم التعابير في الذاكرة، وذلك بواسطة مصمم اللغة ، ولكن ذلك يقلل من المرونة للمبرمج .
ثانياً : بوضع حالة تقييم العوامل في التعابير بترتيب محدد في تعريف اللغة ومطابقة المطبق بضمان ذلك الترتيب ، ولكن ذلك لا يسمح للمترجم باستخدام طرق الأداء الأمثل.

7. التحميل الزائد للمعاملات Over loaded operators

عزيزي الدارس، المعاملات الحسابية عادة تستخدم لعدة أغراض، مثلاً يستخدم المعامل أحياناً لجمع أي عوامل من نوع رقمي ، بعض اللغات مثل Java أيضاً تستخدمها لإلصاق السلاسل الحرفية. هذا التعدد في استخدام المعامل يسمى التحميل الزائد للمعامل over operator loading وهو مقبول بصورة عامة ما دامت قابلية القراءة و الاعتمادية لم تتأثر .

1.7. ملاحظات حول التحميل الزائد للمعاملات

1- في اللغات مثل APL و SNOBOL نفس المعامل يستخدم للعمليات الأحادية والثنائية .

2- كمثال للتحميل الزائد للمعاملات اعتبر استخدام المعامل (&) ampersand في لغة C كمؤثر ثنائي التعيين، وفي هذه الحال ampersand تسمى عنوان الـ operator.

مثلاً : $x = \&y$ x تتسبب في وضع عنوان y في x .

3- هنالك مشكلتان مع هذا التعدد في استخدام ampersand

أ- أولاً : استخدام نفس الرمز لعمليتين لا يوجد علاقة بينهما مؤدٍ لقابلية القراءة.

ب-ثانياً الـ `keying error` لترك العامل الاول لعملية `bitwise AND` يستمر بدون أن يكتشفه الـ `Compiler` لأنه يفسر على أنه عنوان الـ `operator` حقيقة كل لغات البرمجة لها أقل خطورة لنفس المشكلة والتي تحدث للتحميل الزائد `minus operator` ، المشكلة فقط أن الحاسوب لا يمكن أن يحدد ما إذا كان المعنى هو مؤثر أحادي أم ثنائي، على الرغم من أن معنى العمليات الاحادي والثنائي يكون مرتبطاً ولو بصورة بسيطة لن يؤثر على قابلية القراءة.

4- الرموز المختلفة للمعامل ليس فقط تزيد قابلية القراءة ولكن تكون مناسبة أو ملائمة للعمليات، ولكن المشكلة قد تنتج من أخطاء الكتابة غير مكتشفة.

5- التحميل الزائد للمؤثرات المعرفة بواسطة المستخدم، مثلاً إذا `+` * تم من تحميلهم لبيانات مجردة للمصفوفات أفرض `A,B,C,D` متغيرات من هذا النوع إذن `A+B+C+D` يمكن أن تستخدم بدلاً عن `Matrix add (Matrix , Mult (A,B)` `Matrix Mult (C,D)` على الجانب الآخر يمكن أن يكون التحميل مضراً بقابلية القراءة وعلى القارئ معرفة كلاً من أنواع المؤثرات وتعريف المعامل ليتمكن من معرفة المعنى ، أي من أو كل هذه التعريفات يمكن أن تكون في ملفات أخرى .
`C++` لديها بعض المؤثرات لا يمكن تحميلها مثل `(.)` `class member operator` والـ `(::)` `scope resolution operator` والتحميل الزائد واحدة من خصائص الـ `C++` .

أسئلة التقويم الذاتي



- 1) ماذا نعني بترتيب تقييم المعاملات؟
- 2) أشرح متى يحدث التأثير الجانبي.
- 3) عرف التحميل الزائد للمعاملات.

8. تحويل النوع Type Conversion

يكون واحداً من أثنتين هما:

أ) narrowing = يحول قيمة إلى النوع، لا يمكن تخزينه ولو بالتقريب كل قيم النوع الأصلي (الاول) مثلاً تحويل double إلى float في C. أو

ب) widening : يحول قيمة أي نوع يمكن احتواؤها على الاقل بالتقريب لكل قيم النوع الأصلي مثلاً تحويل int إلى float في الـ C.

الـ Widening Conv تقريباً دائماً آمنة وإنما الـ narrowing ليست كذلك

كمثال للمشكلة التي يمكن أن تحدث مع widening conv اعتبر الحالة التالية : في كثير من تطبيقات اللغة، تحويل int إلى float يعتبر widening conv بعض الدقة قد تفقد مثلاً في بعض التطبيقات يخزن integer في 32 bits وهي أيضاً على الأول تسعة خانات عشرية من الدقة، لكن في كثير من الحالات قيم الـ float point تخزن أيضاً في 32 bits بحوالي سبعة خانات من الدقة لذا تحويل integer إلى float - point يمكن أن ينتج عنه فقدان خانتان دقة.

9. عملية التصحيح في التعابير

عزيزي الدارس، واحد من قرارات التصميم باعتبار التعابير الحسابية هو ما إذا كان مثلث للمعامل أن يكون له أنواع مختلفة. اللغات لا تسمح بمثل هذه التعبيرات التي تسمى التعابير الهجين لأن الكمبيوتر عادة لا يحتوي على عمليات ثنائية لأنواع مختلفة من تصحيح المؤثرات. أنه تحويل صنفى لأنواع مستعملة بواسطة الحاسوب.

تشير إلى أن تحويل الأنواع صراحه المطلوبة بواسطة المبرمج كتحويل مباشر أو casts وليس تصحيحاً.

مع أن بعض رموز المعاملات يمكن أن يتحمل زائدياً ، نحن نفترض أن نظام الكمبيوتر إما في المعدات الصلبة أو في بعض مستوى محاكاة للرمجيات له عملية لكل أنواع المؤثرات و المعاملات معرف في اللغة للتحميل الزائد للمؤثرات في لغة التي تستخدم نوع ربط ساكن ، الحاسوب يختار النوع الصحيح للعملية بناء على انواع المؤثر.

* عندما يكون العاملان للمؤثر ليسا من نفس النوع- وهو قانوني في لغة- يقوم المترجم باختيار واحد من منهم ليكون التصحيح. وتجهيز الشفرة للتصحيح. بما أن مصممو اللغات ليسو على اتفاق على موضوع التصحيح في التعابير الحسابية .

هؤلاء الضد جزء كبير من المصححين مهتمون بمشاكل الاعتمادية التي يمكن أن تحدث من التصحيح لأنها تبدد فوائد اختبار الأنواع ، هؤلاء الذين إلى حد ما يحتون كل تلك التصحيحات أكثر اهتماماً بفقدان المرونة التي تنتج من القيود .

كمثال لمخاطر وتكلفة التصحيح الزائد ، اعتبر جهود الـ PLII لاكتساب المرونة في التعابير PLII متغير سلسلة حرفي يمكن أن يدمج مع integer في عبارة run time ، السلسلة يتم مسحها إلى قيمة عددية . إذا كان القيمة تحتوي على خانة عشرية نفترض أن القيمة هي من نوع float point المؤثر الآخر يكون جبرياً float point والنتيجة العملية هي floating point هذه السياسة للتصحيح مكلفة جداً لأن كلاً من اختبار النوع والتحويل يجب أن يحدث في زمن التشغيل، و ذلك يؤدي إلى فقدان احتمال اكتشاف أخطاء المبرمج في التعابير لأن المعامل الأحادي يمكن أن يدمج مؤثراً من أي نوع مع مؤثر فعلي من نوع آخر . لأن اكتشاف الأخطاء يزداد عند السماح بالتعابير الهجين اللغتان Ada mode 20 تسمح بقليل من المؤثرات في تعابير من نوع الهجين ولا واحد يسمح بدمج العوامل integer و float point في تعبير ، يوجد استثناء واحد في Ada العامل الأسّي (**) يمكن أن يكون من نوع integer أو float point للمؤثر الأول، ونوع integer للمؤثر الثاني كل من اللغتين يسمح بقليل من أنواع type mixed وعادة يكون مرتبطاً بأنواع فرعية.

في معظم اللغات المعروفة لا يوجد قيود على النوع الهجين. حيث إن C++ والـ Java لديها أنواع integer أصغر من النوع int في C++ يوجد shorting char في Java هو byte ، short وأي operator يطبق عليهم . لذلك أثناء ما البيانات تتخزن في متغيرات من هذه الأنواع لا يمكن معالجتها قبل التحويل إلى نوع أكبر مثلاً

```
byte a , b , c ;
```

```
.....
```

```
a = b + c
```

قيم الـ b و C تحول إلى int وعملية الجمع لـ int تنفذ ثم الناتج يحول إلى byte ويوضع في a.

1.9 التحويل المباشر

معظم اللغات تقدم بعض القدرات لعمل التحويل الصريح بكلاً نوعيه widening and narrowing في بعض الحالات رسائل التحذير تنتج explicit narrowing في تغير كذا دلالة في قيم الكائن الذي تم تحويله.

كل من Ada و Modula 2 تقدم عمليات explicit con التي لها syntax استدعاء الدالة.

مثلاً في Ada : لدينا $AVG := \text{Float}(\text{sum}) / \text{Float}(\text{count})$

حيث : AVG هو floating point و sum count يمكن أن يكونا من أي نوع رقمي في اللغات العينية CL. التحويل المباشر يسمى صيغة caste . cast ليس مثل استدعاء الدالة ، النوع المراد يوضع بين أقواس قبل التعبير المراد تحويله كما في angle (int). واحد من الأسباب لوجود الأقواس في تحويلات لغة C هو أن لغة C لها عدة أسماء two word type مثل long int.

10. الأخطاء في التعبيرات Errors in Expressions

عزيزي الدارس، عدة أخطاء يمكن أن تقع عند تقييم التعبير، إذا كانت اللغة تتطلب اختباراً للأنواع وعندئذ أخطاء نوع المؤثر لا يمكن أن تحدث . أنواع أخرى من الأخطاء (غير الأخطاء التي يمكن أن تحدث بسبب تصحيح المؤثر في التعبيرات أيضاً مقابل محدودية المعاملات الحسابية في الحاسوب. والمحدودية المشتقة (متوازنة) من الحسابية. إن أكثر الأخطاء شيوعاً ينشأ عندما تكون نتيجة عملية لا يمكن تمثيلها في خلية للذاكرة حيث يجب تخزينها هذا يسمى over flow أو under flow اعتماداً على ما إذا كانت النتيجة أكبر بكثير أم أصغر بكثير أحد محدودية الحسابية هو عدم السماح بالقسمة على صفر على الرغم من انه رياضياً غير مسموح بها إلا أن ذلك لا يمنع برنامجاً من محاولة القيام بذلك. floating point over flow and . under flow , division by zero هي أمثلة لأخطاء زمن التشغيل وهي أحياناً تسمى الاستثناءات التي تعين أي حلقة ستستمر. كلاً من break , exit يقدم خروج متعدد من الحلقة. والتي هي جزء من تقليل قابلية القراءة. الشروط الاعتيادية التي تتطلب نهاية الحلقة معروفة جداً لذلك مثل هذا construct وجودها مبرر أكثر من ذلك قابلية القراءة لم تضر بشكل كبير لأن الهدف ، خرج مثل هذه الحلقة هو الجملة الاولى بعد الحلقة واكثر من فقط أي مكان في البرنامج break في لغة Java هي استثناء لان الهدف يمكن أن يكون أي صيغة فك تشفير مركبة.

أسئلة التقويم الذاتي

(1) يتم تحويل النوع بطريقتين اذكرهما.

(2) ما التحويل المباشر؟

(3) ما أكثر أخطاء التعبيرات شيوعاً؟

الخلاصة

عزيزي الدارس، في هذه الوحدة شرحنا مفهوم التعابير في لغات البرمجة و هي المعنى الأساسي لتعيين الحسابات، و فهمنا أن خواص التعابير الحسابية مشتقة من القواعد الرياضية المتعارف عليها، كما قسمنا ترتيب تقويم المعاملات إلى الأسبقية، أي التسلسل حسب الأولوية و التشاركية وهي التي تبين كيفية حساب مثل هذا التعبير ووضحنا أن التشاركية لها عدة قوانين، و قلنا إن المبرمجين يمكن أن يبدلوا قوانين الترابط و الأسبقية بوضع اقواس في التعبير أو وضع التعابير الشرطية. قلنا أن المعاملات الحسابية تستخدم عادة لعدة أغراض و سمينا تعدد استخدام المعامل بالتحميل الزائد للمعامل.

قسمنا تحويل النوع إلى narrowing و widening و قلنا أن الثانية أكثر أمانا من الأولى. وشرحنا عملية التصحيح في التعابير و الأخطاء التي يمكن أن تحدث فيها.

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سوف نناقش مفهوم البيانات المجردة في لغات البرمجة المختلفة بداية بلغة 67 SIMULA إلى اللغات الحالية .

تبدأ الوحدة بمناقشة عامة عن تجريد البيانات ، ثم بعد ذلك نتحدث عن عملية الكبسلة في لغات البرمجة . و تقوم الوحدة بتعريف تجريد البيانات ويتم توضيح ذلك بمثال .

بعد ذلك نتحدث الوحدة عن الدعم الموجود في لغات البرمجة لتجريد البيانات مثل Ada و ++C.

إجابات التدريبات

تدريب-1:

بافتراض أن قيمة $x = 5$ و $y = 6$

فإن التعبير الأول :

$$\begin{aligned} & x * y - x - y \\ & = 5 * 6 - 5 - 6 = 19 \end{aligned}$$

التعبير الثاني :

$$\begin{aligned} & x * y - (x - y) \\ & = 5 * 6 - (5 - 6) \end{aligned}$$

أولا نحسب ما داخل الأقواس :

$$\begin{aligned} & = 5 * 6 - (-1) \\ & = 30 - (-1) = 31 \end{aligned}$$

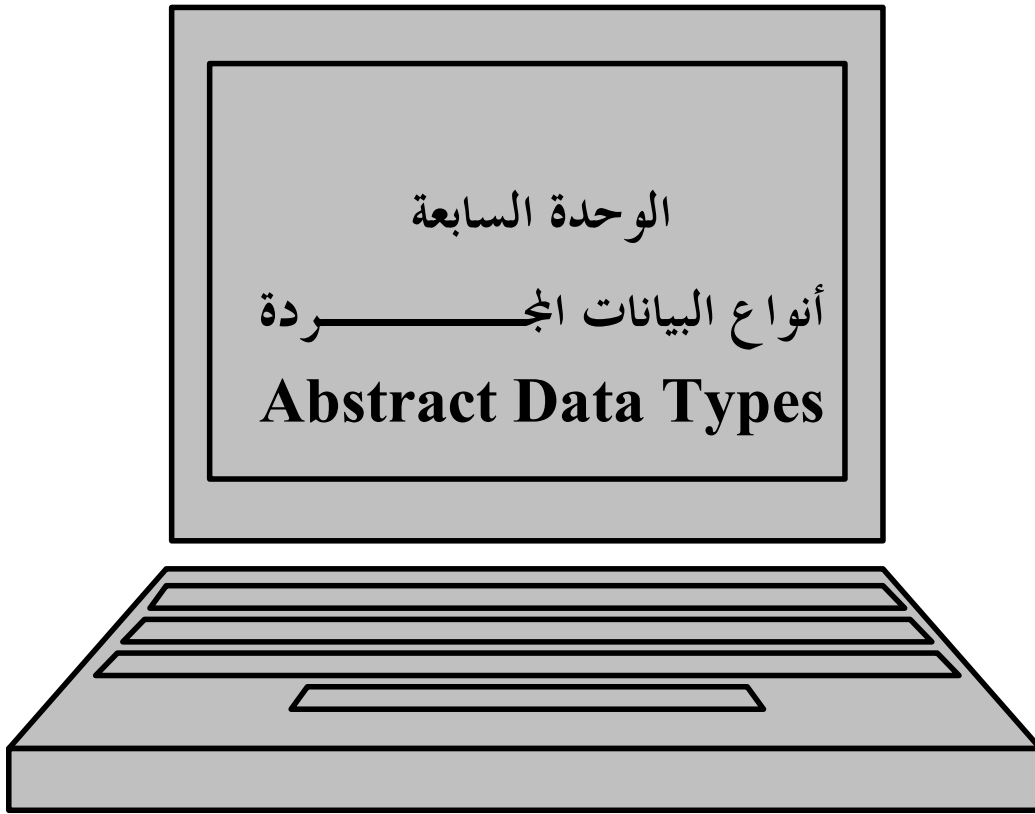
مسرد المصطلحات

التعابير:

التعابير هي المعنى الأساسي لتعيين الحسابات في لغات البرمجة و من الصعب على المبرمج فهم كل من الصيغة اللغوية syntax والمعاني semantic للتعابير .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
121	المقدمة
121	تمهيد
121	أهداف الوحدة
122	1. مفهوم تجريد البيانات
122	1.1. تجريد العمليات Process Abstraction
122	2. الكبسلة
123	3. مقدمة تجريد البيانات
123	1.3. الأنواع القياسية Built in Data Type
123	2.3. البيانات المجردة المعرفة بواسطة المستخدم user defined Type
123	4. عملية إخفاء المعلومات Information Hiding
124	5. قضايا التصميم Design Issues
126	6. أمثلة اللغات التي تستخدم أنواع البيانات المجردة Language Examples
126	1.6. تجريد البيانات في لغة Ada
126	2.6. تجريد البيانات في لغة ++ C
129	الخلاصة
129	لمحة مسبقة عن الوحدة التالية
130	إجابات التدريبات
131	مسرد المصطلحات
132	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة السابعة من المقرر "مفاهيم لغات البرمجة" في هذه الوحدة سوف نناقش مفهوم البيانات المجردة في لغات البرمجة المختلفة بداية بلغة SIMULA 67 إلى اللغات الحالية.

تبدأ الوحدة بمناقشة عامة عن تجريد البيانات ، ثم بعد ذلك سوف نتحدث عن عملية الكبسلة في لغات البرمجة . بعد ذلك سوف نعرف تجريد البيانات ونوضح ذلك بمثال سوف نتحدث عن الدعم الموجود في لغات البرمجة لتجريد البيانات مثل Ada و ++C

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تعرف مفهومي تجريد البيانات و تجريد العمليات.
- تعرف الكبسلة.
- تتعرف على أنواع البيانات المجردة و عملية إخفاء المعلومات.
- تشرح قضايا التصميم.



1. مفهوم تجريد البيانات

عزيزي الدارس، التجريد عبارة عن نظرة أو تمثيل لفئة تحتوي على خصائص في سياق معين.

التجريد يمكننا من جمع خصائص مفردة للفئة في صفة واحدة عامة تؤخذ في الاعتبار. داخل المجموعة، الخصائص التي تعرف العناصر مفردة هي التي تؤخذ في الاعتبار، مما يؤدي إلى تبسيط مهم جداً للعناصر داخل المجموعة. يستخدم التجريد لتبسيط تعقيدات عمليات البرمجة. هنالك نوعان من التجريد، هما: تجريد العمليات، وتجريد البيانات

1.1. تجريد العمليات Process Abstraction

هي طريقة لتحديد أن هنالك يجب أن تتم بدون تحديد تفاصيل دقيقة عن كيفية عملها كما هو مستخدم في البرنامج الفرعي.

مثال ذلك إذا كان لدينا الاستدعاء التالي للدالة swap
Swap (int a , int b) ;

هذا الاستدعاء عبارة عن تجريد لعملية التبديل، حيث إن المستخدم لا يهتم أن يعلم تفاصيل الخوارزمية التي تقوم بالتبديل، بل كل ما يهتم هو الخصائص التالية:
1- اسم الدالة ، البارامترات التي يجب أن تبدل وأنواعها، والنتيجة من عمليات التبديل .

عملية التجريد تساعدنا في تكوين وقراءة وفهم البرامج خاصة في حالة البرامج الكبيرة مع مئات الأسطر من الشفرات البرمجية .

2. الكبسلة Encapsulation

الكبسلة هي عملية تجميع البرامج الفرعية والبيانات التي تتعامل معها .
1- لماذا الكبسلة ؟

من الضروري تقسيم البرامج الكبيرة التي تتكون من آلاف الأسطر إلى وحدات برمجية صغيرة تسمى (modules) لتسهيل علمية إدراتها وفي حالة التعديل عليها إعادة ترجمتها لكي نجد طريقة لعدم إعادة الوحدات التي لا تتأثر بالتعديلات .
لذا فإن الكبسلة تعالج مشكلتي البرامج الكبيرة وإعادة الترجمة لهذه البرامج.

2- المقدمة إلى تجريد البيانات

نوع البيانات المجرد : هو عبارة عن تجميع للبيانات مع البرامج الفرعية التي تعمل على هذه البيانات (كبسلة). عند التعامل مع هذه البيانات ،التفاصيل غير المهمة من هذه البيانات تكون مخفية عن الوحدات خارج الكبسلة التي تستخدم مع هذه البيانات، ويمكن للوحدات التي تستخدم البيانات المجردة التصريح حتى لو كانت مخفية عنها. تقليل تعقيدات عميلة البرمجة وذلك بتسهيل عملية إدارة البرامج الكبيرة أو المعقدة .

أسئلة التقويم الذاتي



- 1) عرف مفهومي تجريد البيانات و تجريد العمليات.
- 2) لماذا الكبسلة؟

3. أنواع البيانات المجردة

هنالك نوعان من البيانات المجردة

1.3. الأنواع القياسية Built in Data Type

وهي عبارة عن أنواع البيانات التي تأتي مع لغات البرمجة مثل نوع البيانات الحقيقية (floating point) حيث إن كل اللغات تأتي معها وتوفر طريقة لإنشاء متغيرات من نوع البيانات الحقيقية بالإضافة إلى مجموعة العمليات الرياضية

2.3. البيانات المجردة المعرفة بواسطة المستخدم user defined

Type

هي أنواع بيانات مجردة يقوم بتعريفها المستخدم وتحتوي على التالي :

- أ- الوصف أو التعريف للنوع مع العمليات التي تعمل على هذه البيانات
- ب- وصف الكائن (Object) من هذا النوع مخفي عن الوحدات البرمجية التي تستخدم نوع البيانات المجردة .

4. عملية إخفاء المعلومات Information Hiding

عزيزي الدارس، يقصد بعملية إخفاء المعلومات أن الوحدة البرمجية التي تستخدم أنواع البيانات المجردة ليس من المهم أن تعلم تفاصيل نوع البيانات، وتستفاد منها في الآتي:

1- الوحدات التي تستخدم نوع البيانات لا يمكنها أن ترى تفاصيل نوع البيانات لذا فإن وحدتها البرمجية مستقلة تماماً.

2- يمكن التغيير في تفاصيل نوع البيانات في أي وقت من دون الرجوع إلى الوحدة التي تستخدم .

3- الوحدات البرمجية التي تستخدم نوع البيانات لا يمكن أن تغير في تفاصيل البيانات، لذا فهي عالية الموثوقية.

مثال :

يوضح المثال التالي نوع البيانات المجردة وعملية التعامل معها .

افترض أن لدينا مكس (stack) من نوع بيانات مجردة لديه العمليات المجردة التالية

Create (stack) : لإنشاء كاش مكس.

destroy (stack) : يحرر التخزين من المكس.

empty (stack) : دالة منطقية للتأكد من المكس خالي.

push (stack, element) : تصنيف عنصر للمكس.

pop (stack) : تزيح العنصر الأول في المكس.

Top (stack) : تعطي العنصر الأول في المكس.

الوحدة البرمجية التي تستخدم المكس تكون كالتالي :

```
Create (stk1) ;
push (stk 1 ,color 1) ;
push (stk1 , color1 ) ;
if (not empty (stk1) ;
then temp = top (stk1) ;
.....
.....
```

5. قضايا التصميم Design Issues

1/ لتدعم لغة البرمجة صفات تجريد البيانات لابد أن تحتوي على الصيغ التي تدعم الكبسلة في البيانات المجردة، بمعنى أنها تستطيع تغليف البيانات والبرامج الفرعية التي تعمل على تلك البيانات .

2/ لابد من اجعل اسم النوع وتعريف البرنامج مرئياً للبرنامج الذي يستخدم النوع المجرد.

3/ بعض العمليات القياسية يجب أن تضمن في اللغة.

ملاحظة

- تدعم لغتي ++C و Java كبسلة أنواع البيانات المجردة بطريقة مباشرة



- (1) وضح أنواع البيانات المجردة .
- (2) ما المقصود بعملية إخفاء المعلومات ؟

6. أمثلة اللغات Language Examples

عزيزي الدارس، في هذا القسم من الوحدة سوف نتحدث عن أمثلة من اللغات التي تدعم تجريد البيانات وهي : ++C و Ada و نعرف كيف يتم تطبيق تجريد البيانات في هذه اللغات.

1.6. تجريد البيانات في لغة Ada

تمكن لغة Ada استخدام الكبسلة لنموذج البيانات المجردة بالإضافة إلى إخفاء المعلومات

الكبسلة في لغة Ada:-

- يتم تطبيق الكبسلة في Ada عن طريق الحزمة (packages)
- تتكون الحزمة من جزئين :
- أ- وصف الحزمة : تعطي الوصف لكبسلة.
- ب- حسم الكبسلة : تعطي التطبيق للوصف.

إخفاء المعلومات في لغة Ada

تستخدم لغة Ada كلمة private (خاص) لإخفاء المعلومات عن البرنامج المستخدم للبيانات المجردة كما في المثال التالي:

Type NODE – TYPE Private

المثال التالي يوضح كيفية إنشاء وتطبيق حزمة المكسد في لغة Ada

مثال

- يتم إنشاء وصف الحزمة في الخطوة الأولى كما يأتي:

Package stack ck is

- هذا الجزء مرئي

Type STACK TYPE is limited private ;

MAX – SIZE : constant := 100;

Function EMPTY (stk: in STACK TYPE) return Boolean ;

Procedure PUSH (stk : in out STACK TYPE)

.....

.....

هذا الجزء مخفي

....

```
Private
Type list – type is array (1..MAX- SIZE) of INTEGER ;
Record
List : list – type ;
End record ;
end STACK PACK ;
```

- يتم تطبيق جسم الحزمة في الخطوة التالية كما يأتي:

```
Package body STACK PACK
Function EMTY (stk : in STACK TYPE ) return Boolean is begin
return stk TOPSUB = 0 ;
end EMTY ;

end STACK PACK ;
```

2.6. تجريد البيانات في لغة ++C

أ- يتم تطبيق تجريد البيانات في لغة ++C باستخدام الفئات (classes) وهي عبارة عن نوع يعرف بواسطة مكون من جزئين:

- 1- أعضاء البيانات Data members : يتم فيه تعريف البيانات
- 2- الدالات الأعضاء member function

يتم فيه تعريف الدالات التي تستخدم البيانات

ب- يتم تطبيق إخفاء البيانات في لغة ++C باستخدام الكلمة private (خاص)
ملحوظات على فئات ++C :

- 1- إذا لم تكن البيانات مخفية تستخدم كلمة public (عام)
 - 2- تستخدم لغة ++C دالة في تعريف الفئة تسمى دالة الإنشاء (construction)، وهي تستخدم لإعطاء قيم ابتدائية للكائنات ولها نفس اسم الفئة.
 - 3- تستخدم ++C دالة الهدم (destructor) وهي تستخدم لأنها لتحرير الذاكرة عن الكائن وتحمل اسم الفئة مع العلامة (~)
- المثال التالي يستخدم كمكس لتوضيح لبقية إنشاء الفئة في لغة ++C وكيفية عملها
- مثال :

- الجزء الأول عبارة عن تعريف الفئة في ++C:

```

include <io stream - h>
class stack {
private : // هذه البيانات مخفية
int * stack ptr ;
int max - leng
public : // هذه البيانات غير مخفية
stack 1 // دلالة الإنشاء
{ stack - ptr = new int [100] ;
    max - len = 99 ;
    top - ptr = -1 ;
}

~ stack ( ) {delete} [ ] stack ptr ; } ; // دالة الهدم

void push ( int number )
{ if top - ptr == max - len )
Cout << " Error - stack is full in " ;
Else stack - ptr [ ++ top - ptr ] = number ;

void pop ( ) {
if (top - ptr == -1 )
cout << "Error - stack id empty in " ;
els top - ptr - - ;
}

int top ( ) {return stack - ptr [ top - ptr ] ;
int empty ( ) {return top - ptr == -1);}

}; // نهاية تعريف الفئة

```

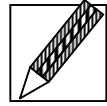
- الجزء الثاني عبارة عن تطبيق الفئة في ++ C حيث يتم إنشاء كائن باسم stk كما يلي :

```

void main ( ) {
int top - one ;
stack stk ; // إنشاء كائن stk
stk . push (42) ;
stk . push (17) ;
top - one = stk . top ( ) ;
stk . pop ( ) ;
.....
}

```

تدريب (1)



أنشئ فئة في C++، ثم بين كيفية تعريف أعضاء البيانات Data memba والدالات الأعضاء Member functions .
اكتب الشفرة التي تقوم بتطبيق الفئة المعرفة.

نشاط



ناقش مع مشرفك و زملائك أمثلة للغات تدعم تجريد البيانات.

الخلاصة

عزيزي الدارس، في هذه الوحدة تم نقاش مفهوم البيانات المجردة في لغات البرمجة المختلفة بداية بلغة SIMULA 67 إلى اللغات الحالية.

بدأت الوحدة بمناقشة عامة عن تجريد البيانات، ثم بعد ذلك تحدثت عن عملية الكبسلة في لغات البرمجة . بعد ذلك تم تعريف تجريد البيانات حيث تم توضيح ذلك بمثال . كما تعرضنا للدعم الموجود في لغات البرمجة لتجريد البيانات مثل Ada و ++C.

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سنتعرف على الاستثناءات والتي هي عبارة عن الأحداث غير الاعتيادية التي تحدث نتيجة عن خطأ أو غيره، والتي تكشف بواسطة البرنامج أو المعدات الصلبة في الحاسوب وتحتاج إلى معالجة خاصة.

إجابات التدريبات

تدريب-1:

الجزء الأول : تعريف الفئة يتم كالآتي :

```
include < iostream . h >
class student
{
private :
int st No ,
char st Name ,
public :
int get No ( int st No )
{ Return ( st No ) , }
char get Name ( char st Name )
{ Return ( st Name ) , }
} ,
```

نهاية تعريف الفئة

الجزء الثاني : تطبيق الفئة

```
void main ( )
student st Obj ,
```

إنشاء كائن stobj

```
stobj : get No ( 20 ) .
stobj : get Name ( Ahmed ) .
}
```

مسرد المصطلحات

تجريد البيانات

التجريد عبارة عن نظرة أو تمثيل لفئة تحتوي على خصائص في سياق معين.

process Abstraction تجريد العمليات

هي طريقة لتحديد أن هنالك عمليات يجب أن تتم بدون تحديد تفاصيل دقيقة عن كيفية عملها كما

هو مستخدم في البرنامج الفرعي.

الكبسلة Encapsulation

الكبسلة هي عملية تجميع البرامج الفرعية والبيانات التي تتعامل معها .

عملية إخفاء المعلومات Information Hiding

يقصد بعملية إخفاء المعلومات أن الوحدة البرمجية التي تستخدم أنواع البيانات المجردة ليس من المهم أن تعلم تفاصيل نوع البيانات.

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
135	المقدمة
135	تمهيد
135	أهداف الوحدة
136	1. البرمجة الموجهة للكائنات
137	2. عملية البرمجة في لغات البرمجة الموجهة بالكائنات
137	3. أمثلة على لغات البرمجة الموجهة بالكائنات
137	1.3. لغة small talk
143	2.3. لغة ++C
145	3.3. البرمجة الموجهة في لغة جافا
146	الخلاصة
146	لمحة مسبقة عن الوحدة التالية
147	مسرد المصطلحات
148	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة الثامنة من المقرر "مفاهيم لغات البرمجة" في هذه الوحدة سوف نناقش مفهوم البرمجة الموجهة بالكائنات ثم يتبع ذلك نقاش قصير عن الوراثة والربط الديناميكي ، بعد ذلك سوف نتحدث عن اللغات التي تدعم البرمجة الموجهة بالكائنات مثل لغات small talk ، Java ولغة Eiffel .

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً على أن:

- تعرف مفهوم البرمجة الموجهة للكائنات.
- تشرح علاقة التوريث بالبرمجة الموجهة للكائنات.
- تتعرف على لغة small talk و مواصفاتها و مكوناتها.
- تتعرف على مكونات و فئات التوريث في لغة C++ و لغة جافا.



عزيزي الدارس، مجموعة من اللغات الحالية تدعم أسلوب البرمجة الموجهة بالكائنات مثل C++ , Ada 95 , Eiffel لغة + C و Ada 95 تدعم البرمجة الهيكلية بالإضافة إلى البرمجة الموجهة بالكائنات، في البرمجة الموجهة للكائنات يتم تجميع عام لصفات البيانات المجردة وتوضع في نوع جديد. الأعضاء في هذه المجموعة تورث صفاتها العامة من ذلك النوع الجديد فيما يسمى بالوراثة وهو القطب الأساسي للبرمجة الموجهة بالكائنات .

1. البرمجة الموجهة بالكائنات

أسلوب البرمجة بالكائنات يركز في البيانات المجردة (تم نقاشها في الوحدة السابقة) حيث تتم الحوسبة في كائن (object) بيانات بواسطة استدعاء برنامج فرعي مرتبط بهذا الكائن . مثال ذلك إذا أردنا إجراء عملية ترتيب لعناصر كائن مصفوفة ، يتم تعريف الترتيب في نوع البيانات المجردة مصفوفة (Array) ويتم إجراء الترتيب باستدعاء هذه العملية في كائن مصفوفة محدد.

هنالك أربعة أقطاب رئيسية في البرمجة بالكائنات وهي (1) الكبسلة (2) التوريث (3) إعادة الاستخدام (4) تعدد الصور.

(1) إعادة الاستخدام software reuse

يعتبر أسلوب إعادة استخدام البرمجيات من الأسباب الرئيسية التي ساعدت في تطور البرمجيات في السنين الأخيرة .

حيث يتم إعادة استخدام البيانات المجردة مع كبسلتها وعمليات الدخول إليها ولكن هنالك صعوبة في حالة التعديل لذا استخدم التوريث

(2) التوريث : عبارة عن عملية إنشاء نوع البيانات مجردة جديدة من بيانات موجودة أصلاً حيث يتم توريث البيانات والوظائف من النوع القديم إلى النوع الجديد مع إمكانية التعديل في بعض الفئات الموروثة .

(3) الكبسلة Encapsulation

هي عملية تجميع البيانات المجردة. والوظائف التي تعمل على هذه البيانات ينتج عنه فئة (class)

(4) تعدد الأشكال والصور polymorphism

يقصد به تعدد أشكال الوظائف التي تنشأ في حالة التوريث حيث يتم إنشاء وظيفة جديدة في الفئة الجديدة على أساس الوظيفة في الفئة القديمة مكوناً شكلاً هيكلياً من الوظائف . من فوائدها سهولة تمديد البرمجيات في حالتي التطوير والصيانة .

2. عملية البرمجة في لغات البرمجة الموجهة بالكائنات

عزيزي الدارس، في كل لغات البرمجة الموجهة بالكائنات تتم البرمجة بواسطة إجراء التقنية التالية :

- يتم إرسال رسالة إلى كائن (object) لاستدعاء إحدى وظائفه.
 - يقوم الكائن بإرسال رد على الرسالة عن قيمة الحوسبة للوظيفة.
- الهدف الرئيسي للبرمجة الموجهة بالكائنات هو الحل مشكلات البرمجة بواسطة تحديد الكائنات الحقيقية للمشكلة والعمليات التي تجري على هذه الكائنات ثم بعد ذلك عمل محاكاة لهذه الكائنات ، عمليات وكيفية التعامل فيما بينها .

أسئلة التقويم الذاتي



- (1) اشرح مفهوم البرمجة الموجهة للكائنات.
- (2) ما الهدف الرئيسي للبرمجة الموجهة للكائنات؟

3. أمثلة على لغات البرمجة الموجهة بالكائنات

في هذا القسم من الوحدة سوف نتعرض لبعض اللغات التي تدعم البرمجة الموجهة بالكائنات مثل لغة small talk التي تعتبر بالكامل موجهة بالكائنات ولغة ++C و Ada.

1.3 لغة small talk

أولاً: بيئة small talk

نظام اللغة small talk يتكامل فيه محرر البرنامج ، والمترجم ، والمواصفات العادية لنظام التشغيل والماكنة الظاهرية، و الواجهة للبرنامج هي صورية ثم كتابة لغة small talk بالكامل بلغة small talk حيث يمكن للمستخدم تعديلها لتلائم استخدامه .

ثانياً: المواصفات العامة للغة small talk :

تتصف لغة small talk بالآتي :

أ- تكون لغة small talk بالكامل من الكائنات ، أي شيء في small talk بالكامل من الكائنات ، أي شيء في small talk هو كائن (object) الثابت 2 يعتبر كائن – ملفات النظام أيضاً هو كائن

ب- كل كائن يقوم بالتالي :

- له ذاكرة محلية.
- يمكن أن يورث العمليات.
- له إمكانية بالتخاطب مع الكائنات الأخرى
- الرسائل بين الكائن ترسل في شكل وكذلك في الردود على تلك الرسائل التي تستخدم لإرجاع طلب الرسائل أو التي تؤكد انتهاء المهمة .
- يتم حجز الذاكرة للكائنات في small talk ديناميكياً، حيث يتم الدخول بطريقة غير مباشرة إلى الكائنات وأيضاً تحرير الذاكرة يتم بطريقة غير مباشرة عن طريق عملية جمع القمامة
- ح- لإنشاء لغة small talk باقي اللغات الهجين مثل C++ و Ada التي تدعم البرمجة الموجهة والبرمجة الإجرائية لأنها صممت فقط من أجل البرمجة الموجهة بالكائنات.

ثالثاً: مكونات لغة small talk

في هذا الجزء سوف نستعرض مكونات لغة small talk

تتكون لغة small talk من الآتي :

1- التعبيرات Expressions

- تبنى الدالات في small talk من التعبيرات.
- توصف التعبيرات الكائنات
- هنالك أربعة أنواع من التعبيرات وهي : الثوابت ، والمتغيرات ، ومجموعة التعبيرات ، والرسائل .
- وهنا سوف نناقشها ببعض التفصيل.

2- الثوابت : literals

- تتكون من الأرقام والسلاسل ، والكلمات المفتاحية وهي عبارة عن كائنات تعرف بالاتفاق المرسل والنتيجة بعد إرجاع الرسالة.
- تعرف في تعريف الفئة مع الفئات الموروثة .

3- المتغيرات Variables

- هنالك نوعان من المتغيرات:
- أ- خاصة : هي محلية بالنسبة للكائن.
- ب- مشاركة shared : غير محلية.
- كل المتغيرات في small talk هي مراجع للكائن أو الفئة . وليس لها نوع.

4- الرسائل : messages

- توفر طريقة للتخاطب بين الكائنات
- لها جزءان هما : مواصفات الكائن المرسله له والرسالة نفسها.
- هنالك ثلاثة انواع من الرسائل، هي: أحادية، وثنائية، ومفتاحية :
- أ- رسالة أحادية : تتكون من جزئين ، الجزء الأول يحدد الكائن المستقبل والجزء الثاني يحدد الدالة (الطريقة) التي يجب تنفيذها بواسطة الكائن مثال ذلك :

First Angle

ترسل رسالة بلا بارامتر إلى الدالة sin من الكائن first angle
ب- رسالة ثنائية : مثال ذلك العمليات الحسابية
مثلاً $21 + 2$

و

sum / count

في حالة الرسالة الأولى 21+2 يتم ارسال البارامتر (كائن) إلى وحدات (+) في كائن 21
رسالة مفاتيحية : مثال ذلك

first Array at : 1 put 5

هنا يتم ارسال الكائنات 1 و 5 إلى الدالات at و put من الكائن first Array .
- يمكن للرسالة أن تكون خليطاً من الثلاثة السابقة مثلاً:

first Array at : mdex -1 put : 77.

في هذه الحالة تنفذ الأحادية أولاً ثم الثنائية ثم المفتاحية حسب الأسبقية من اليسار إلى اليمين .
ومعناها : أرسل 1 إلى دالة (-) من الكائن indea النتيجة من هذه العملية مع 77 يتم ارسالهما إلى دالتي : at و put من الكائن first Array .

الدالات (الطرق) : Methods -

تتكون الصيغة العامة للدالة في لغة small talk كالتالي :

Message – pattern - [1 temporary variables 1]statements

حيث أن :-

message – pattern : هو عبارة عن رأس الدالة ويمثل اسم الدالة

[] : ما داخله هو اختياري

Statements : الصيغ

أ- القيمة التي ترجعها الدالة تكون مسبقة بعلامة (8)

مثال ذلك :

Current Total

^(old Total + new Total)

هذه الدالة تسمى current Total وترجع قيمة التعبير old Total + new Total

صيغة الإسناد : Assignment Statement

- تستخدم العلامة ← كمعامل للإسناد

مثال ذلك :

total < - z z

sum < - total

حيث إن :

total : هو اسم لمتغير يشير إلى الكائن يمين العلامة - <

z z : هو كائن

Sum : هو متغير آخر يشير إلى الكائن

ويمكن استخدام عملية الاسناد لحفظ الرسالة المرسله المرجعية، بواسطة الدالات نضع صيغة

الرسالة في الجانب الأيمن من صيغة الاسناد كما في المثال التالي:

Index < - index +1

4: dependent :350.0 gross pay deducts net pay

حيث أنه : في الصيغة الأولى الرسالة +1 أرسلت إلى الكائن المشار إليه بواسطة index

المتغير index متغير ليحمل الكائن الجديد بعد بتنفيذ الدالة

في الصيغة الثانية نرسل الرسالة المفتاحية :

4: dependent :350.0 Grosspay

إلى الدالة deducts grosspay dependent من الكائن deducts

ويتم إسناد الكائن المرجع بواسطة deducts إلى المتغير netpay

5- مجموعات التعابير : Blocks

يتم تجميع أكثر من تعبير في small talk داخل أقواس مربعة [] بينها علامة (.)

المثال التالي :

[index < - index + 1 . sum < - sum + index]

تنفيذ المجموعة تستخدم كلمة value وترسل كرسالة للمجموعة كما في المثال التالي :

[sum < - sum + index] value

حيث يتم إرجاع قيمة آخر تعبير في المجموعة .

يمكن أن تسند المجموعة إلى متغير ثم بعد ذلك ننفذها بإرسال value كما في المثال التالي:

أ - إسناد المجموعة إلى متغير :

index إلى sum:

تنفيذ المجموعة حيث يتم إضافة
add index value

ب -

6- الحلقات loops :

يوجد في لغة small talk ثلاثة أنواع من الحلقات

أ - طريقة while true :

تقوم while true بإرسال مجموعة تعابير إلى مجموعة أخرى تحتوي على شرط التكرار .
وهي ترسل قيمة true (صحيح) أو false (خطأ) إلى الكائن الذي يراد تنفيذه كما في المثال
التالي :

count < - 10

sum < - 0.

مجموعة الشرط " [count <= 2]

While True : [sum < - sum + count .

count < - count + 1] "ال الحلقة"

يتم تنفيذ مجموعة التعابير كالآتي:

1- يتم إرسال المجموعة التي تحتوي على "sum + count" , "count + 1" كبرامتر إلى
الطريقة while true.

2- ترسل while true قيمة value صحيح أو خطأ إلى المجموعة الشرط [count <=]
[2].

3- إذا كانت القيمة صحيح (true) يتم تنفيذ مجموعة التعابير .

4- تكرر العملية حتى ترسل القيمة خطأ "false"

ب - طريقة Times Repeat

وفيها يتم إرسال Times Repeat مع رقم مجموعة تعابير كبرامتر حيث يتم تنفيذ مجموعة
التعابير عدد مرات مساوياً للرقم المرسل كما في المثال التالي :

X cube < - 1.

3 times Repeat : [x cube < - x cube * x]

تقوم بحساب × ثلاث مرات

ج - طريقة to: do و to: by :

هي شبيهة بحلقة for في اللغات الأخرى

حيث إن to : عبارة عن تعبير رقمي عن نهاية التكرار .

do : عبارة عن مجموعة تعابير يتم تنفيذها بواسطة الدالة الرقمية، وطريقة عملها كما في المثال التالي :

[to : 5 do : [sum < - sum + ×]

- تقوم بتنفيذ مجموعة التعابير [sum < - sum + ×] خمس مرات

كما يمكن أن تستخدم do: to بشكل آخر كما في المثال التالي :

2 to :10 by :2 do : [even l sum < - sum + even]

حيث أن:

البارامتر even يأخذ القيم الرقمية 2,4,6,8,10

طريقة if true : if false :

يتم إرسال رسالة إلى التعبير المنطقي . إذا كانت النتيجة true (صحيح)؛ فإن الرسالة تكون (true) في هذه الحالة يتم إرسال قيمة (true) إلى الطريقة if: true : if false إلى الجزء الأول من التعبير ويتم تجاهل الجزء الثاني ، أما إذا كان التعبير false (خطأ) يتم إرسال قيمة (false) إلى الجزء الثاني ويتم تجاهل الأول كما في المثال التالي :

"تعبير منطقي" total = o

if true : [average < - o]

if false : [average < - sum // total]

حيث أن:

إذا كان التعبير المنطقي يساوي true يتم تنفيذ مجموعة التعبير:

[average < - o]

إذا كان false يتم تنفيذ التعبير : [average < - sum // total]

ملاحظة :

تستخدم علامة // للقسمة

7- الفئات classes

تتكون الفئات في small talk من أربع أجزاء وهي :

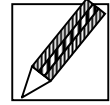
- اسم الفئة (Class)
- اسم الفئة الأساسية (super class).
- التصريح عن المتغيرات (variable).
- التصريح عن الدالات في الفئة (class methods).

ملاحظات على فئات small talk

- المتغيرات في small talk مخفية
 - الفئات هي عبارة عن كائنات مما يمكنها من استقبال رسائل
 - يمكن إنشاء أفراد من الفئة بواسطة إرسال رسالة new إلى الفئة كما في المثال التالي
- : our pen < - pen new
- حيث يقوم بإنشاء فرد من الفئة pen بإرسال رسالة new إلى الفئة pen ووضعها في المتغير our pen .

تدريب (1)

كون حلقة True while في لغة Smaltalk لجمع الأعداد الزوجية حتى 100



2.3. لغة C++

تتكون لغة C++ من مجموعة كبيرة من الفئات Classes والتي تدعم عملية تجريد البيانات في C++ والبرمجة الموجهة بالكائنات .

في هذا القسم سوف نتحدث عن المظاهر المهمة في لغة C++ التي تدعم البرمجة الموجهة بالكائنات .

أولاً: مكونات لغة C++

تعتبر لغة C++ لغة هجين بما تحتويه من صفات اللغات الإجرائية التي ورثتها عن لغة C و صفات البرمجة الموجهة بالكائنات المضافة إليها، لذا فإنها تدعم البرمجة الإجرائية والبرمجة الموجهة بالكائنات .

وفي هذا القسم سوف نتحدث عن الفئات والتوريث والربط الديناميكي في لغة C++ .

ثانياً: الفئات في لغة C++ Classes :

الفئة في C++ عبارة عن تجميع للبيانات والدوال التي تعمل في مكان واحد.

يتم إنشاء الفئة عن طريق ذكر كلمة Class متبوعاً باسم الفئة بعد ذلك نذكر البيانات والتي تسمى أعضاء البيانات Data members والدوال والتي تعرف بـ Function members

كما في المثال التالي :

```

Class base – class {
    Private :
    Int a ;
    Flaot × ;

Public :
    int geta ( ) ;
    Flaot pint ( ) ;

```

ملاحظات حول الفئات في C++:

- 1- كل الفئات في C++ لها دالة تسمى دالة الإنشاء Constructor، وهي تقوم بإعطاء قيمة ابتدائية لأعضاء البيانات في الكائن الجديد، والذي يتكون بنفس اسم الفئة . ويتم استدعاؤها مباشرة عند إنشاء الكائن .
- 2- أغلب الفئات في C++ تحوي دالة تسمى دالة الهدم، وهي عبارة عن اسم الفئة متبوعاً بعلامة (~) وتقوم بإنهاء عمل دالة الإنشاء، وأيضاً تقوم بتحرير الذاكرة التي حجزت للكائن .
- 3- تحتوي لغة C++ على مستويات مختلفة من التحكم في البيانات التي يتعامل معها البرنامج الذي يستخدم الفئة مثال ذلك : private (خاص) Public (عام) و Protected (محمي)

ثالثاً: التوريث Inheritance

يمكن إنشاء فئة جديدة من فئة موجودة فيما يسمى بالتوريث، حيث إن الفئة القديمة تسمى الفئة الأساسية (أو الأب) base Class – Parent، والفئة الجديدة تعرف بالابن (Child)، المثال التالي : يوضح عملية التوريث في C++

مثال :

أولاً: الفئة الأساسية Class base – Class

```

Private :
int a ;
flaot × ;

Pretecled :
int b ;
flaot y ;
public :
int c ;
flaot z ;
};

```

ثانياً: عملية التوريث :

Class sub - class : public base – class {...};

Class sub – class : public base – class {...};

3.3. البرمجة الموجهة في لغة جافا

تشبه لغة جافا لغة + C في الفئات والتوريث والدالات إلا أن هنالك بعض الاختلافات والتي سوف نتعرض لها هنا .

أولاً: المواصفات العامة للغة جافا

- 1- كل الفئات في جافا لابد أن تكون فئات فرعية من فئة أساسية لأن هنالك عمليات عامة تحتاجها البرمجة في جافا .
- 2- كل الفئات في جافا تكون ديناميكية التخزين ويستخدم المعامل new لحجز الذاكرة – وطريقة جمع القمامة لتحرير الذاكرة .

ثانياً: التوريث في جافا

الملاحظات المهمة حول التوريث في جافا هي :

- 1- تدعم لغة جافا التوريث الأحادي single Inheritance مباشرة.
- 2- في حالة التوريث المتعددة تستخدم جافا فئة وهمية تسمى الواجهة.
- 3- تستخدم جافا كلمة Final لتعريف الدالات التي لا يمكن إعادة تعريفها في الفئات التي تورث من فئات أعلى . مما يعني أن الفئة التي تعرف فيها لا يمكن أن تكون فئة أساسية (super class) لأي فئة أخرى.

أسئلة التقويم الذاتي



- 1) اذكر المواصفات العامة للغة small talk.
- 2) ما مكونات لغة small talk؟
- 3) اشرح كيف يتم التوريث في لغتي C++ و جافا مع التمثيل.

الخلاصة

عزيزي الدارس، في هذه الوحدة تم نقاش مفهوم البرمجة الموجهة بالكائنات، ثم بعد ذلك تم نقاش قصير عن الوراثة والربط الديناميكي ، بعد ذلك تحدثت الوحدة عن اللغات التي تدعم البرمجة الموجهة بالكائنات مثل لغات small talk ، Java ، ولغة Eiffel

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سوف نتحدث عن بعض أساليب البرمجة التي تتخذها الحواسيب. و قد تم تقسيم الوحدة الى قسمين هما:
القسم الأول يتحدث عن لغا البرمجة المنطقية، و هي التي تعتمد في عملها على المنطق، ومنها لغة Prolog.
أما القسم الثاني من الوحدة فيتحدث عن لغات البرمجة الوظيفية، وهي اللغات التي تعتمد في عملها على المعادلات الرياضية . من أولى اللغات الوظيفية لغة LISP .

مسرد المصطلحات

- الكبسلة Encapsulation :

هي عملية تجميع البيانات المجردة والوظائف التي تعمل على هذه البيانات ينتج عنه فئة (class)

- تعدد الأشكال والصور polymorphism :

يقصد به تعدد أشكال الوظائف التي تنشأ في حالة التوريث حيث يتم إنشاء وظيفة جديدة في الفئة الجديدة على أساس الوظيفة في الفئة القديمة مكون شكلاً هيكلياً من الوظائف .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
151	المقدمة
151	تمهيد
151	أهداف الوحدة
152	1. طرق معالجة الاستثناءات
152	2. فوائد معالجة الاستثناءات
153	3. قضايا التصميم
153	4. أمثلة اللغات في معالجة الأخطاء
154	1.4. الاستثناءات في لغة + Ada
155	2.4. الاستثناءات في لغة ++ C
157	الخلاصة
175	لمحة مسبقة عن الوحدة التالية
175	إجابات التدريبات
158	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة التاسعة من المقرر "مفاهيم لغات البرمجة".
تعرف الاستثناءات على أنها الأحداث الاعتيادية التي تحدث نتيجة خطأ أو غيره، والتي
تكشف بواسطة البرنامج أو المعدات الصلبة في الحاسوب، وتحتاج إلى معالجة خاصة .
ومثال ذلك خطأ في قراءة القرص أو نهاية الملف .

هي عملية الأخطاء التي تكشف بواسطة الحاسوب، و تتم بواسطة وحدة برمجية تسمى
معالج الاستثناءات Exception Handler. يتم إعلان الخطأ عندما يتم اكتشاف الحدث
المصاحب له وتختلف عملية معالجة الخطأ على حسب حالته ، فمثلاً الاستجابة الحدث
نهاية الملف تختلف عن الاستجابة لحدث خطأ في القراءة... وهكذا. وفي بعض الحالات
تتم الاستجابة بواسطة إظهار رسالة خطأ وإنهاء البرنامج .

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً
على أن:

- تعرف الإستثناءات.
- تتعرف على طرق معالجة الاستثناءات.
- تشرح أهم قضايا التصميم التي تهتم بمعالجة الاستثناءات.
- تتعرف على الاستثناءات في لغة C++.



أثناء تنفيذ البرمجيات في الحاسوب تحدث حالات خطأ يمكن اكتشافها بواسطة المعدات الصلبة للحاسوب أو بواسطة البرمجية. نجد أن معظم لغات البرمجة صممت وطبقت غير أن من الصعب على البرنامج المستخدم أن يكتشف أو يتعامل مع هذه الأخطاء، مما يسبب في إنهاء البرنامج ويرجع التحكم لنظام التشغيل . ثم يستجيب نظام التشغيل بإظهار رسالة ثم يوقف البرنامج .

لابد للغة البرمجة أن تحوي طريقة يمكن بها أن تتعامل مع الأخطاء التي سيتم اكتشافها بواسطة المعدات الصلبة أو بواسطة البرمجيات في الحاسوب . وأيضاً تسمح للمبرمج أن يعرف الأخطاء غير الاعتيادية الأخرى واستخدام هذه الطريقة للتعامل معها . وتسمى هذه الطريقة معالجة الاستثناءات .

1. طرق معالجة الاستثناءات

تختلف طرق معالجة الأخطاء حسب كل طريقة ومن الطرق المتبعة ما يلي:-

1- في بعض الحالات تكون الاستجابة الأنسب هي إنشاء رسالة خطأ، ثم بعد ذلك إنهاء البرنامج.

2- في بعض الحالات إرسال بارامتر مساعد، حيث يستخدم كمتغير للحالة، وتستخدم له قيمة في الوحدة المستدعاة حسب صحته . مباشرة بعد الرجوع إلى الوحدة قامت بالاستدعاء يتم اختيار متغير الحالة وفي حالة وجود خطأ يتم تنشيط معالجة الخطأ كما في لغة C.

3- في بعض الحالات يتم إرسال علامة (label) إلى البرنامج الفرعي مما يمكن الوحدة المستدعاة من التحول إلى نقطة أخرى في البرنامج عند حدوث الخطأ، كما في لغة باسكال .

4- في بعض الحالات يمكن أن يكون هنالك معالج خطأ كبرنامج فرعي مختلف يمرر كبارامتر الوحدة التي تقوم بالاستدعاء حيث تقوم الوحدة المستدعاة بإستدعاء معالج الأخطاء.

2. فوائد معالجة الاستثناءات

هنالك فوائد حقيقية في احتواء اللغة على معالج الاستثناءات ومن هذه الفوائد التالي :

1- من غير معالج الأخطاء يجب على المبرمج أن يقوم ببرمجة معالج الأخطاء مما يسبب عبء ثقيل على البرنامج. مثال ذلك إذا كان هنالك برنامج يقوم بعشرة عمليات قسمة على الصفر إذا لم يوجد معالج الاستثناءات يقوم المبرمج بإنشاء واحداً لكل عملية قسمة .

2- يستفاد من دعم معالج الاستثناءات في اللغة في عملية نشر الاستثناءات، حيث من الممكن استخدام حالة معالجة واحدة في أكثر من مكان في البرنامج والاستفادة منها في الحالات المشابهة


- 3- اللغة التي تدعم معالجة الاستثناءات تشجع المستخدم على الاهتمام بالاستثنائية التي تحدث أثناء تشغيل البرنامج وطريقة معالجتها .
- 4- في بعض البرامج يمكن لمعالجة الاستثناءات أن تسهل الحالات غير الاعتيادية

3. قضايا التصميم

أهم القضايا التي يجب على اللغة أن تهتم بها في معالجة الاستثناءات التالية:

- 1- كيف وأين يمكن تحديد معالج الاستثناءات وما هو مجاله؟
- 2- كيف يمكن ربط الاستثناء مع معالج الاستثناء؟
- 3- أين يمكن للبرنامج أن يستمر ، هل بعد تنفيذ معالج الاستثناءات؟
- 4- كيف نحدد الاستثناءات المعرفة بواسطة المستخدم؟
- 5- هل من الممكن أن يكون هنالك معالج استثناء افتراضي في حالة أن البرنامج لا يوفره؟
- 6- هل الأخطاء التي تكتشف بواسطة المعدات الصلبة تعامل كما الأخطاء التي تكتشف بواسطة البرمجيات
- 7- هل اللغة تحتوي على معالج الاستثناءات؟
- 8- هل من الممكن إيقاف تنشيط معالج الاستثناءات؟

أسئلة التقويم الذاتي

- 

 - 1) لماذا يستخدم المبرمج طريقة معالجة الاستثناءات؟
 - 2) اذكر فوائد طرق معالجة الإستثناءات.
 - 3) ما اهم القضايا التي يجب على اللغة أن تهتم بها في معالجة الاستثناءات؟

4. أمثلة اللغات في معالجة الاخطاء

عزيزي الدارس، تحتوي اللغات C++ و Java و Ada على ميزات لمعالجة الأخطاء في هذا القسم من الوحدة سوف نتعرض لهذه الميزات في بعض اللغات مثل C++ و Java و Ada من معالج الأخطاء والاستمرارية وعملية ربط الاستثناء مع المعالج مع مثال يوضح ذلك :

1.4. الاستثناءات في لغة Ada +

أ. معالج الأخطاء

تكون معالجة الأخطاء في Ada محلية في الوحدة البرمجية التي يكتشف فيها الخطأ حيث تستخدم نفس المرجعية .
الصيغة العامة لمعالجة الأخطاء هي:-

When Exception – choice - {1 exception – choice = > atate – sequ

حيث أن :

{ } : تعني إن الذي بداخلها اختياري .

exception -choice : تكون في شكل exception – name / other (exception)
name – يرمز إلى الأخطاء)

other ترمز إلى الأخطاء التي لم يتم ذكر في exception)

ربط الاستثناء مع معالج الاستثناءات

تتم عملية ربط الاستثناءات في Ada على حسب الحالات التالية :

1- إذا تم اكتشاف استثناء واحد في وحدة تحتوي على معالج لهذا الاستثناء يتم ربط الاستثناء مع هذا المعالج محلياً .

2- إذا لم تحتوي الوحدة على معالج يتم نشر الاستثناء إلى وحدة أخرى على حسب الحالات التالية :

أ- في حالة الإجراء بدون معالج الاستثناءات يتم نشر الاستثناء إلى البرنامج في نقطة الاستدعاء للإجراء .

ب- في حالة أن تكون الوحدة التي تقوم باستدعاء الإجراء بدون معالج استثناءات يتم نشر الاستثناء إلى الوحدة باستدعائها .

ت- في حالة أن الاستثناء نشر إلى البرنامج الرئيسي ولم يوجد معالج الاستثناء في البرنامج الرئيسي يتم إنهاء البرنامج .

3- إذا اكتشف الاستثناء داخل مجموعة صيغ (block) يتم نشر الاستثناء إلى البرنامج الذي استدعى المجموعة .

4- إذا اكتشف الاستثناء داخل جسم حزمة يتم نشر الاستثناء إلى القسم الذي يعرف الحزمة .

ج/ الاستمرارية : Continuation

يتم إنهاء مجموعات الصيغ التي يتم اكتشاف الاستثناءات داخلها في Ada بالإضافة إلى الوحدات التي يتم نشر الاستثناءات إليها والتي لا يمكنها أن تعالج تلك الاستثناءات . كما أن

التحكم لا يرجع إلى الوحدة التي تم اكتشاف الاستثناء داخلها بعد معالجتها . حيث إن التحكم يستمر بعد كلمة exception التي عادة ما تكون في آخر المجموعة أو الوحدة .

2.4. الاستثناءات في لغة ++C

هنا سوف نستعرض بعض ميزات لغة ++C في دعم معالجة الاستثناءات وهي :
أ/ معالج الاستثناءات :-

تستخدم لغة ++C تركيبة خاصة بواسطة كلمة (try) حيث تحتوي تركيبة (try) على صيغة مركبة تسمى (try clause) بالإضافة إلى قائمة الاستثناءات .
والصيغة العامة لتركيبية (try) هي :

```
try {  
                                                                    //  
    الوحدة التي يحتمل أن * *  
                                                                    //  
    تكشف استثناء * *  
  
    catch (formal parameter) {  
                                                                    //  
        جسم معالج الأخطاء * *  
                                                                    {  
            ....  
        }  
    }  
    catch (formal parameter) {  
                                                                    //  
        جسم معالج الأخطاء * *
```

حيث إن :

دالة catch : هي معالج أخطاء

ب/ ربط الاستثناء مع المعالج :

تستخدم لغة ++C كلمة **throw** للإعلام عن اكتشاف الاستثناءات وتستخدم بواسطة الصيغة العامة التالية:

```
throw [expression] ;
```


في لغة ++C يتم ربط الاستثناء بالطرق التالية :

1- الخطأ الذي يكتشف في هيكل try يسبب إيقاف تنفيذ الوحدة البرمجية في الحال، ثم بعد ذلك يبحث عن تطابق لمعالج عن طريق throw، ويتم التطابق بالتسلسل من أقرب معالج إلى try إلى أن يوجد .

2- تعالج الاستثناءات محلياً إذا وجد تطابق في معالج استثناءات محلي . وإذا لم يوجد محلياً يتم نشر الاستثناء إلى الدالة التي قامت بالاستدعاء، وإذا لم يوجد أي تطابق في البرنامج يتم إيقاف البرنامج .

ج/ الاستمرارية

ففي ++C بعد انتهاء معالج الاستثناءات من عمله يتم إرجاع التحكم إلى أول جملة هيكل try أي الجملة مباشرة بعد آخر يتبع لهيكل try .

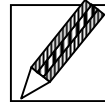
تدريب (1)

افترض أن لديك برنامج بلغة Add، وإذا كان هنالك خطأ في برنامج فرعي من هذا البرنامج ، وضح كيف تقوم Add بمعالجة ذلك .



أسئلة التقويم الذاتي

(1) اشرح الاستثناءات في لغتي Ada+ و ++C.



الخلاصة

عزيزي الدارس، في هذه الوحدة تم التعرف على الاستثناءات والتي هي عبارة عن الأحداث غير الاعتيادية التي تحدث نتيجة عن خطأ أو غيره، والتي تكشف بواسطة البرنامج أو المعدات الصلبة في الحاسوب وتحتاج إلى معالجة خاصة . وقد ناقشنا الاستثناءات في لغتي Ada+ و C++ .

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سوف نناقش مفهوم البرمجة الموجهة بالكائنات، ثم يتبع ذلك نقاش قصير عن الوراثة والربط الديناميكي ، بعد ذلك سوف نتحدث عن اللغات التي تدعم البرمجة الموجهة بالكائنات مثل لغات small talk ، Java ، ولغة Eiffel .

إجابات التدريبات

تدريب-1:

تتم معالجة الخطأ في البرنامج الفرعي الذي تم اكتشاف الخطأ فيه حيث تستخدم الصيغة التالية:

{ } choice – exception when

ويتم ربط الاستثناء مع معالج الاستثناءات بأحدى الطرق التالية :

- 1) إذا احتوى البرنامج الفرعي على معالج للاستثناءات يتم الربط محليا في البرنامج الفرعي ، وإذا لم يوجد يتم نشر الاستثناء إلى الوحدة التي قامت بالاستدعاء .
- 2) إذا لم يوجد معالج للاستثناءات يتم نشر الاستثناء إلى البرنامج الرئيس ، وإذا لم يوجد معالج استثناء ينتهي البرنامج .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.



محتويات الوحدة

الصفحة	الموضوع
161	المقدمة
161	تمهيد
161	أهداف الوحدة
162	1. لغات البرمجة المنطقية
163	2. لغات البرمجة الوظيفية
164	3. لغات قواعد البيانات
166	الخلاصة
167	المراجع

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في الوحدة العاشرة من المقرر "مفاهيم لغات البرمجة".
البرمجة المنطقية هي عبارة عن استخدام وصفي منطقي للتعبير عن الحوسبة المنطقية
في الحاسوب وتستخدم اللغات المنطقية الحساب الاستنتاجي للقيام بذلك .
عملية البرمجة في اللغات المنطقية عملية غير إجرائية . حيث تقوم البرامج في هذه
اللغات بوصف شكل النتائج ولا تنص على كيفية حوسبة هذه النتائج.

أهداف الوحدة

عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة أتوقع أن تكون قادراً
على أن:

- تعرف لغات البرمجة المنطقية.
- تعرف لغات البرمجة الوظيفية.
- تعرف لغات قواعد البيانات.



1. لغات البرمجة المنطقية Logical Programming Language

لغة البرمجة المنطقية : Prolog

عزيري الدارس، تم تطوير التصميم الأساسي من لغة Prolog بواسطة آليات كوليمرور وفيليب روسيل في مجموعة الذكاء الاصطناعي من جامعة أيكس - مارسيليا بالتعاون مع روبرت كواليسكي من قسم الذكاء الاصطناعي في جامعة إدنبرغ.

المكونات الأساسية للغة برولوج عبارة عن طرق تحديد الحساب الاستنتاجي وتطبيقات أشكال محدودة من الـ Resolution

أول مفسر للبرولوج تم تطويره في العام 1972م في مارسيليا .

مكونات لغة Prolog :

تتكون لغة برولوج من مجموعة من الصيغ . بما أن هذه الصيغ قليلة العدد إلا أنها معقدة. واحدة من استعمالات برولوج هو نوع قواعد البيانات الذكية .

قواعد البيانات للغة برولوج تتكون من نوعين من الصيغ - وهي **حقائق Truth وقواعد**

.Rules

مثال لحقائق الصيغة التالية :

mother (Joanne , jake)
father (vern . Joanne)

حيث تنص على أن :

Joanne هي أم jake

و vern هو والد Joanne

مثال لقواعد - الصيغ التالية :

grand parent (x , y) : _parent (x ,y) parent (y ,z)

حيث تنص على :

يمكن أن تستنتج X هو جد Z إذا كان X والد y و y هو والد Z

قواعد بيانات في برولوج يمكن أن نستعملها بواسطة صيغة الهدف

مثال ذلك :

father (bob , derice)

حيث يتم الاستعلام عن هل bob هو والد drice ، ويتم استخدام عملية resolution

لمحاولة تحديد حقيقة صيغة الهدف . فإذا كانت النتيجة صحيحة فإن الصيغة تطبع " true "

أما إذا لم تستطع إثبات ذلك فإنها تطبع " false ".

2. لغات البرمجة الوظيفية Functional Programming Language

في هذا القسم من الوحدة سوف نتحدث عن لغات البرمجة الوظيفية وهي اللغات التي تعتمد في عملها على المعادلات الرياضية .

مواصفات اللغات الوظيفية

هنالك صفات خاصة بلغات البرمجة الوظيفية وهي :

1- لا تستخدم لغات البرمجة الوظيفية المتغيرات ولا صيغ الإسناد، مما يجعل المبرمج الاهتمام بخلايا الذاكرة التي ينفذ فيها البرنامج.

2- البرامج في لغات البرمجة الوظيفية عبارة عن تعاريف للدالات ومواصفات لتطبيق الدالات . والتنفيذ عبارة عن تقديم لتطبيقات الدالات.

3- عمليات التكرار تستخدم الاستدعاء الذاتي للدالات لعدم وجود متغيرات

بداية الذكاء الاصطناعي وعمليات القوائم :

اللغة الوظيفية الأولى LISP تم تطويرها للمساعدة في خصائص اللغة الداعمة لعمليات القوائم في مجال الذكاء الاصطناعي .

بداية الاهتمام بالذكاء الاصطناعي ظهرت في منتصف 1950 في عدد من الأماكن . بعض هذه الاهتمامات نشأت في مجال اللغات الطبيعية . بعضها في مجال علم الاجتماع، وبعضها الآخر في مجال الرياضيات. اهتمت اللغات بمعالجة اللغات الطبيعية وفي مجال علم الاجتماع كان الإهتمام بتخزين واسترجاع المعلومات الإنسانية فقد كان اهتمامهم في حوسبة بعض العمليات الذكية مثل إثبات النظريات الرياضية، كل هذه التحقيقات أدت إلى نتيجة واحدة ألا وهي لابد من تطوير طرق تسمح للحواسيب لتقوم بعمليات على الرموز في القوائم المتصلة

مفهوم عمليات القوائم طور بواسطة آلن نيويل وهاربرت (1956) سيمون حيث تم نشرها في ورقة توصف أول برنامج في الذكاء الاصطناعي تم تطوير لغة IPLI، ثم بعد ذلك لغة IP11 (1960) وهي من نوع لغات البرمجة وهي عبارة عن لغة تجميع تطبيق بوساطة مفسر حيث يتم احتواء تعليمات القوائم .

العمليات في اللغات الوظيفية :-

تم تطوير لغة LISP على أنها وظيفية . كل عمليات الحوسبة في اللغات الوظيفية تتم بواسطة تطبيق الدالات، حيث لا يتم استخدام المتغيرات ولا الإسناد كما في اللغات الوظيفية. وأيضاً لا تستخدم الحلقات (Loops) في عمليات التكرار، حيث يستخدم الاستدعاء الذاتي للدوال (Recursion) .

هذه المفاهيم الأساسية للغات البرمجة الوظيفية جعلتها مختلفة عن لغات البرمجة الوظيفية .

تكوين لغة LISP

تختلف لغة LISP عن اللغات الوظيفية لأنها لغة برمجة وظيفية ولأن البرنامج بلغة LISP يختلف عن البرامج في اللغات الأخرى أمثال FORTRAN و C - لأن تكوين لغة C عبارة عن خليط من اللغة الإنجليزية والجبر - ولكن LISP عبارة عن نموذج مبسط الشفرة البرمجية والبيانات في LISP لها نفس الشكل - مثال ذلك :

(A , B e D)

حتى يمكن تفسيرها كبيانات قائمة مكونة من أربعة عناصر كما يمكن تفسيرها على أنها شفرة حي تمثل تطبيقاً للدالة A لثلاث معاملات هي B,C,D

اللغة الوظيفية : LISP

تعتبر لغة LISP من أول لغات البرمجة وظيفية تم تطويرها وتستخدم في مجال الذكاء الاصطناعي .

أنواع البيانات في لغة LISP

هنالك نوعان من البيانات في لغة LISP وهي :

أ- الذرات Atoms : وهي عبارة عن رموز تستخدم كمعرفات في لغة LISP أو ثوابت عددية

ب- القوائم Lists : تستخدم كهيكل بيانات.

تعرف القوائم في لغة LISP داخل قوسين () كما في المثال التالي :

(ABCD) وتمثل كما في الشكل (1) التالي

كما يمكن أن تستخدم قوائم متداخلة

(A(BC)D(E(FG)))

حيث إنها عبارة عن قائمة من أربعة عناصر الأول فيها الذرة A والعنصر الثاني (BC) والثالث D والرابع (FG)

3. لغات قواعد البيانات

وتدعى أيضا الجيل الرابع، وهي لغات قواعد البيانات والتي تساعد المستخدم النهائي في صناعة الملفات والشاشات والتقارير أو أي وظيفة أخرى دون كتابة برنامج. وباستخدام هذه اللغات Oracle ، Access فإن المبرمج يقوم بصناعة مجموعة ملفات ذات علاقة فيما بينها حسب أصول ومعايير معينة، ثم يصنع المبرمج مجموعة شاشات لتحديث هذه الملفات كإجراء عمليات الإضافة والحذف والتعديل وصناعة مجموعة تقارير واستعلامات لاسترجاع البيانات الموجودة في هذه الملفات، كل ذلك باستخدام جمل بسيطة تحدد ما هي السجلات المطلوبة

دون تحديد كيفية الحصول عليه. فمثلا لاسترجاع أسماء الطلاب الناجحين من ملف الطلاب
Students نكتب _____ ب :

From average st-name WHERE Students=50؛ TCELES

فيقوم الجهاز بعرض النتائج مباشرة، ولو استخدمنا لغات الجيل الثالث فإننا سنضطر حينها
لكتابة برنامج يتكون من مجموعة أوامر تبدأ بفتح الملف المذكور وقراءة السجل الأول
وفحص المعدل وطباعة اسم الطالب إذا كان المعدل أكبر من أو يساوي 50، ثم ينقل المؤشر
إلى السجل التالي حتى نهاية الملف حيث يتم إغلاقه.
ببساطة عندما تطلب من خادمك أن يعد لك كوبا من الشاي بإعطائه مجموعة أوامر :أحضر
الأبريق، املاه بالماء ، ضعه على النار... الخ فإنك تكون كمن يستخدم لغات الجيل الثالث ،
أما عندما نقول له : أريد كوبا من الشاي ليأتيك بعد دقائق فإن هذا يشبه مولدات التطبيقات.

الخلاصة

عزيزي الدارس، في هذه الوحدة قمنا بالتعرف على لغات البرمجة المنطقية التي تم تطويرها بواسطة آليات كوليمرور وفيليب روسيل بالتعاون مع روبرت كواليسكي . ثم تحدثنا عن لغات البرمجة الوظيفية ومواصفاتها ثم تعرفنا على لغات قواعد البيانات وتدعى أيضا الجيل الرابع، والتي تساعد المستخدم النهائي في صناعة الملفات والشاشات والتقارير أو أي وظيفة أخرى دون كتابة برنامج .

المصادر و المراجع

1. Edsger. W. Dijkstra. "The Humble Programmer" (Turing Award Lecture), Communications of the ACM, Vol 15, No. 10 (October 1972).
2. *Byte Magazine* 25th Anniversary issue. Located online at <http://www.byte.com/art/9509/sec7/sec7.htm>. Refer to <http://www.byte.com/art/9509/sec7/art19.htm> for the article entitled "A Brief History of Programming Languages. "
3. Wasserman, A. "Information System Design Methodology" *Software Design Techniques*, P. Freeman and A. Wasserman (eds). 4th Edition, IEEE Computer Society Press, 1983.
4. Booch, Grady. *Software Engineering with Ada*, 2nd edition. Benjamin Cummings, 1987.
5. Raymond, Eric. *The New Hacker's Dictionary* MIT Press, 1983.
6. Roger Pressman. *Software Engineering: A Practitioner's Approach*, 4th edition. McGraw Hill, 1997.
7. Dijkstra, Edsger. W. *Selected Writings on Computing: A Personal Perspective* Springer-Verlag, 1982.
8. Coad, Peter and Edward Yourdon. *Object-Oriented Analysis, 2nd Edition*. Prentice Hall, 1991.
9. Van Buren, Jim and David Cook. "Experiences in the Adoption of Requirements Engineering Technologies," *CrossTalk*, The Journal of Defense Software Engineering, December 1998, Vol 11, Number 12.
10. Sebesta, Robert *Concept of Programming Languages*, Addison Wesley Longman, 1999.
11. Davis, Alan M. *201 Principles of Software Development*, McGraw-Hill, 1995.
12. Grauer, Robert T., Carol Vasquez Villar, and Arthur R. Buss. *COBOL From Micro to Mainframe*, 3rd edition, Prentice Hall, 1998.
13. Fowler, Martin. *UML Distilled*, Addison Wesley Longman, Inc. 1997.
14. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, Addison Wesley, 1999.
15. Fowler, Martin. *Analysis Patterns: Reusable Object Models* Addison-Wesley, 1997.
16. M. Paulk, et.al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.