FACULTY OF COMPUTING & INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY

كلية الحاسبات وتقنية المعلومات
جامعة الملك عبدالعزيز

بسم الله الرحمن الرحيم

CPCS 301 – Programming languages

2nd semester 2020

Summary for the 1st mid exam

هذا التلخيص هوا جهد احد الطلاب فراح اوضح نقطتين

أولاً: هذا التلخيص لا يُغني عن السلايدات او الكتاب بالعكس انا هنا ملخص تقريبا 90 بالمية من الافكار وشارحها ب التفصيل, الافضل بعد ماتخلص قراءة الملخص تروح تمر على السلايدات وباذن الله تكون لميت المنهج كله

ثانياً: انا طالب مثلي مثلكم وهذا مجرد اجتهاداً مني, وهذا مايعني انه كل الكلام الي شرحته صح ولاكن انا بحثت ودورت ونقحت المعلومات وكتبت الاصح ولاكن ان كان فيه خطا او شي كلموني ع الواتس او اي وسيلة تواصل اخرى وراح اعدله باذن الله

عبدالله الكثيري

CH.1

Programming Domains

- Scientific applications
  - Large number of floating point computations
  - Fortran or Matlab
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL
- Artificial intelligence
  - Symbols rather than numbers manipulated
  - LISP
- Systems programming
  - Need efficiency because of continuous use
  - C
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

في هذا السلايد حط لك 5 جوانب مرتبطة بالحاسب واشهر اللغات المستخدمة في هذه الجوانب

مثل : في العلوم, لغة Fortranوالى اخره

بالعربي احفظ الي محدد عليه انا بالالوان

هنا يقولك انه كيف نحدد او نصنف اللغات

Language Evaluation Criteria

- **Readability**: the ease with which programs can be read and understood
- **Writability**: the ease with which a language can be used to create programs
- **Reliability**: conformance to specifications (i.e., performs to its specifications)
- **Cost**: the ultimate total cost

نفصلهم هنا ونشرح كل وحدة لحالها

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Few feature multiplicity (means of doing the same operation)
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures (e.g., `while` statement)
- Data types and structures
  - The presence of adequate facilities for defining data structures

اولا: سهولة القراءة

المقصود فيها انه هل هذي اللغه قرايتها سهلها لليوزر ؟ مثل لغة البايثون والاسيمبلي, البايثون سهولة القراءة فيها عليه, عكس الاسيمبلي, برضوا كمان انه فيه اشياء زي التحكم في البرنامج زي ال if, while statements

احفظ وافهم

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
  - A set of relatively convenient ways of specifying operations طريق مريحة لتحديد العمليات
  - Example: the inclusion of `for` statement in many modern languages

ثانياً: سهولة الكتابة

المقصود هنا هل اللغة سهله في كتابتها ؟ هل المبرمج حيقدر يكتب بسهولة وبسرعه وبدون مايتلخبط ؟

مثلاً انه يمديك تعرف كلاسات كثير زي ماتبغا, يمديك تتحكم في البرنامج من خلال العمليات الشرطية زي ال if,while,break,return بكل سهولة واريحية

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support "natural" ways of expressing an algorithm will necessarily use "unnatural" approaches, and hence reduced reliability

ثالثاَ: الفعالية

بمعنى اخر ال efficiency (ماراح يفهمها الا طلاب ذهب حقين الساعه 2) هههههههه

بيض صح ؟ المهم يعني هل اللغة فيها كفائة عاليه ؟ مثلا ال type checking هل اللغة تشيك على انواع المتغيرات خلال ال run time حق البرنامج ؟ هل تتحقق من انواع المتغيرات لمن تحاول تدمج نوعين مختلفين مع بعض ؟ مثلا ف البايثون يمديك تتدمج بين اي نوعين عادي بدون كاستنق, بس بالجافا لازم تسوي كاستنق

بررضوا ال exception handling يعني هل يمديك تتحكم بالاخطاء الي ممكن تطلع في البررنامج ؟

ال aliasing هذا نصفو لان راح اشرحه بعدييين تحت, سككككككيب

Evaluation Criteria: Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

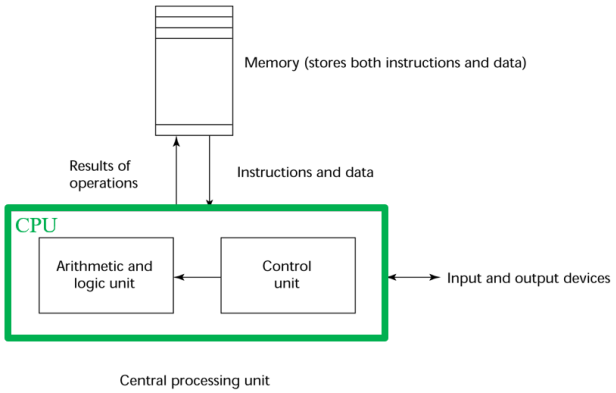
رابعاَ التكلفة

يقولك مثلاً هل اللغة ذي open source ولا لا, برضوا يقولك انه فيه compilers مجانية يمديك تستخدمها

ويقولك بررضوا انه هنا فيه علاقة عكسية بين ال التكلفة و الكفائة, انه ال الكفائة السيئة تؤدي الى تكلفة عاليه

خلصنا الاربعة نقاط ذي, كل الي عليه تحفظ وتفهمها

The von Neumann Architecture



Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory البيانات والبرامج يتخزنوا في الميموري
  - Memory is separate from CPU الميموري منفصل عن المعالج
  - Instructions and data are piped from memory to CPU التعليمات والبيانات يتم تسييرها بين المعالج والميموري
  - Basis for imperative languages
    - Variables model memory cells متغيرات
    - Assignment statements model piping
    - Iteration is efficient لوبات loops

هذا اخذناه ايام مقدمة الحوسبة

الزبدة يقولك انه معمارية الحاسب كيف شكلها, طبعا مرا سهل بس افهم الرسمة لان ممكن يجي عنها سوال

Language Categories

- Imperative الاوامر او لغة للأمر
  - Central features are variables, assignment statements, and iteration
  - Examples: C, Pascal
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Object-oriented
  - Data abstraction, inheritance, late binding
  - Examples: Java, C++
- Markup
  - New; not a programming per se, but used to specify the layout of information in Web documents
  - Examples: XHTML, XML

هنا ياقلبي ايش يقولك, شفت السلايد الي فووووق الي كان فيه لغات برمجة واستخداماتها ؟؟؟

هذا السلايد اخوه, برضوا احفظ وافهم

(طبعا افهم ال Imperative انها لغة تستخدم ال variables, assignment statements, iterations)

Language Design Trade-Offs

Reliability vs. cost of execution
- Conflicting criteria
- Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs

Readability vs. writability
- Another conflicting criteria
- Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability

Writability (flexibility) vs. reliability
- Another conflicting criteria
- Example: C++ pointers are powerful and very flexible but not reliably used

هنا يقولك كل خاصية من الخصائص الاربعة (يجيب وحدة منهم ويقارنها مع الثانية)

Reliability vs Cost of execution اولاً

هنا ي عسل يقولك انه في الجافا الموثوقية عالية والتكلفة عالية برضو, ليش؟ لانه لمن تسوي فور لوب على اراي

```
For(int i = 0; i < array.length ; i++)
```

هنا في الجافا لمن تسوي كذا الكوبايلر راح يسوي تحقق لكل عنصر في الاراي وهذا الشي ياخذ وقت مرااااا كبير, عشان كذا تكلفة التشغيل حتزيد عليك (لمن اللوب تطلع برا الراي راح يجيك ايرور null pointer exception هذا بسبب انه راح تشيك اللغة هل الاراي انتهت ولالا ؟)

اديكم توضيح اكثر ؟ مين فيكم يبرمج بلغة C ؟

في لغة السي لمن تسوي اراي وتدخلها في فور لوب لو انتهت الاراي واللوب صار يأشر على اندكس موموجود راح يرجع لك اي قيمة عشوائية في الميموري.... ليش؟

عشان السي ماتتحقق في الفور لوب عشان كذا السي مرااااا سرريعه والدليل انهم دايم يستخدموها في السيرفرات وفي مراكز الحوسبة عالية الاداء وفي برمجة انظمة التشغيل

Readability vs Writability ثانياً

هنا يقولك فيه لغة اسمها APL هذي اللغة يا شاطر تديك operators مرا كثير وكمان علامات ترقيم كثيرة جداً

بس انا استفدت ايه ؟ (بالمصري)

يقولك هنا في اللغة ذي يمديك تتحكم بالعمليات الرياضية والرمجية بسهولة وبقوة جداً

بس هذا الشي حيخلي الكود قرايته صعبه وتحليله صعب سواء لك او لمبرمج ثاني غيرك لو جا وقراء الكود حقك

Writability vs Reliability ثالثاً

يقولك انه في ال C++ فيه شي اسمه pointer (هذا شي يأشر على ال address في الميموري)

يقولك انه قوي ومفيد وكذا بس ماهو مفيد في الاستخدام او مو فعال

Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters مكس بين الاثنين

طلاسم صح ؟ ماعليك ياجميل بحطها لك ف جدول حلو زيك 😊

Compilation	هوا حاجة تترجم البرامج للغة الآلة مباشرة
Pure Interpretation	هوا برنامج يترجم البرامج (لاحظ هذا برنامج بس الي فوق مو برنامج)
Hybird Implementation System	مكس بين الاثنين ياخذ مميزات الاثنين وسلبياتهم برضوا

Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution

يقولك انه ال compilation يترجم ال high-level languages الى machine code

يعني زي ماقلت لك ف الجدول فوق,,, شي يحول البرنامج الى لغة الالة

وبرضوا يقولك انه يترجم ببطئ, بس يشتغل بسرعه (مهم)

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Becoming rare on high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript)

يقولك ان ال interpreter مايترجم البرنامج للغة الالة

ال errors تطلع لك في الrun-time

تشغيله بطيء من 10 الى 100 ضعف(عكس ال compilation)

يحتاج مساحة زيادة في الميموري

نادر الاستخدام في ال high-level languages

يستخدم في بعض لغات الويب زي javaScript

Hybrid Implementation Systems

- A compromise between compilers and pure interpreters (Mix)
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors

هنا يقولك ال Hybrid الي هو المكس بينهم

اسرع من ال interpretation

مثل لغة ال Perl

CH.3

Introduction

- **Syntax**: the form or structure of the expressions, statements, and program units يعني طريقة الكتابه وهيكلة بناء النص او السكربت
- **Semantics**: the meaning of the expressions, statements, and program units يعني السينتاكس يكون صح ومافيه مشاكل بس الخلل يكون في اللوجيك يعني مثلا يطلع لك ناتج خطا لانك كتبت علامه ضرب بدال جمع او زي كدا

The General Problem of Describing Syntax: Terminology

- A *sentence* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., *, `sum`, `begin`)

بالعربي هي الكلمات او الجمل الي محجوزة في اللغة زي ال If, else, for, void

- A *token* is a category of lexemes (e.g., identifier)

هذا الجدول باذن الله بيفهمك الفرق بي Lexeme and token

[Token]	[Informal Description]	[Sample Lexemes]
if	characters i, f	if
else	characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	letter followed by letters and digits	pi, score, D2
number	any numeric constant	3.14159, 0, 6.02e23
literal	anything but ", surrounded by "'s	"core dumped"

وهذا موقع فيه تفاصيل كثيرة لو تبغا تقرا

https://stackoverflow.com/questions/14954721/what-is-the-difference-between-a-token-and-a-lexeme

نرجع نكمل شغلنا...

BNF and Context-Free Grammars

- Context-Free Grammars
  - Developed by Noam Chomsky in the mid-1950s
  - Language generators, meant to describe the syntax of natural languages
  - Define a class of languages called context-free languages

Backus-Naur Form (BNF)

- Backus-Naur Form (1959)
  - Invented by John Backus to describe Algol 58
  - BNF is equivalent to context-free grammars
  - BNF is a *metalanguage* used to describe another language
  - In BNF, abstractions are used to represent classes of syntactic structures--they act like syntactic variables (also called *nonterminal symbols*)

هذي مقالة تفصلها اكثر لو تبغا تستفيد

https://stackoverflow.com/questions/6641017/what-are-context-free-grammars-and-backus-naur-form

Context Free Grammar (CFG) (BNF)

- $\mathcal{G} = (\mathcal{T}, \mathcal{N}, \mathcal{P}, \mathcal{S})$
  - $\mathcal{T}$: set of terminals
  - $\mathcal{N}$: set of non-terminals
  - $\mathcal{P}$: set of production rules
  - $\mathcal{S}$: start symbol and $S \in N$

CFG Example

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

- $\mathcal{T}$ = {begin, end}
- $\mathcal{N}$ = {<stmt>,<single_stmt>,<stmt_list>}
- $\mathcal{P}$ = {<stmt> → <single_stmt>
  | begin <stmt_list> end}

Or

- $\mathcal{P}$ = {<stmt> → <single_stmt> ,
  <stmt> → begin <stmt_list> end}
- $\mathcal{S}$ = <stmt>

BNF Rules

- A rule has a left-hand side (LHS) and a right-hand side (RHS), and consists of *terminal* and *nonterminal* symbols
- A grammar is a finite nonempty set of rules
- An abstraction (or nonterminal symbol) can have more than one RHS

يعني يقولك احيانا الطرف اليمين يكون له اكثر من طرف ايسر

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

BNF Fundamentals

- Non-terminals: BNF abstractions
- Terminals: lexemes and tokens
- Grammar: a collection of rules
  - Examples of BNF rules:

```
<ident_list> → <ident> | <ident>, <ident_list>
<if_stmt> → if <logic_expr> then <stmt>
```

اقراها بفهم وبعدها خش على المقطع الي تحت راح يديك الزبدة باذن الله

هذا مقطع راح يوضح لك باذن الله السلايد الي فوق

https://www.youtube.com/watch?v=QF-qh454M1k

An Example Grammar

```
<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | <const>
<const> → 1|2|3|4|5|6|7|9|0
```

An Example Derivation

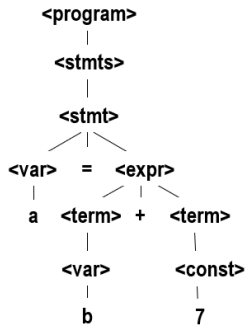
```
<program> => <stmts> => <stmt>
          => <var> = <expr> => a =<expr>
          => a = <term> + <term>
          => a = <var> + <term>
          => a = b + <term>
          => a = b + <const>
          => a = b + 7
```

وهذا مثال على نفس فكرة المقطع وباذن الله توضح لكم الفكره

Parse Tree

- A hierarchical representation of a derivation

هنا يقولك انو كيف تسوي اشتقاق للمعادلات الي قبل كذا بس بشكل
شجرة,ايزي ي عمي

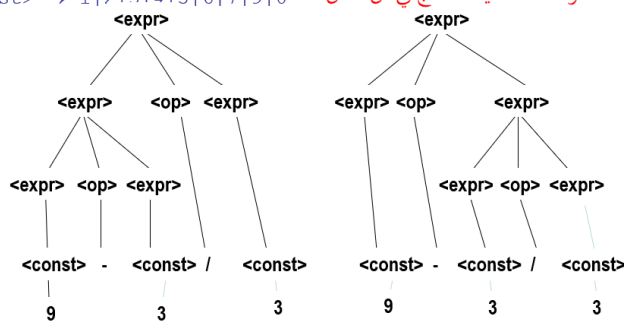


An Ambiguous Expression Grammar

الكلمة المحددة يعني انو الشجره ممكن تتفصل ع اليمين او يسار
زي ما انتو شايفين

```
<expr> → <expr> <op> <expr>  |  <const>
<op> → /  |  -
<const> → 1|2|3|4|5|6|7|9|0
```

بس انتبه
ترا تختلف قيمة الناتج في كل شكل



كلام بسيط درسناه ببرمجة 3, ايززززي

Extended BNF

- Optional parts are placed in brackets [ ]

```
<proc_call> -> <ident> [(<expr_list>)]
```

- Alternative parts of RHSs (right hand side) are placed inside parentheses and separated via vertical bars

```
<term> → <term> (+|-) <const>
```

- Repetitions (0 or more) are placed inside braces { }

```
<ident> → <letter> {<letter>|<digit>}
```

هنا جالس يقول بنديك اشياء متقدمة شوي وهي ال extended BNF ايش هووا ؟؟

يقولك الاشياء ال بين الاقواس ذي {} احتمال تكون موجودة ويمكن لا

والي بين الاقواس ذي () راح تكون وسطها اشارات مفصولة بالاشارة ذي | المهم انه راح يختار وحدة منهم بس

هذا مثال عشان توضح الصورة

Attribute Grammars

- Context-free grammars (CFGs) cannot describe all of the syntax of programming languages
- Additions to CFGs to carry some semantic info along parse trees
- Primary value of attribute grammars (AGs)
  - Static semantics specification
  - Compiler design (static semantics checking)

BNF and EBNF

- BNF

```
<expr> → <expr> + <term>
         | <expr> - <term>
         | <term>
<term> → <term> * <factor>
         | <term> / <factor>
         | <factor>
```

- EBNF

```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
```

طيب جينا لهذا السلايد الي فوق وفيه بعده 3 سلايدات تقريبا لو تقراها تحس انك غبي بس خلوكم معايا شوي

هوا يقول ان بعض الاشياء مايمديك تمثلها بالـ cfgتكون صعبه مايمدينا نستخدمها

عشان كذا صار عندنا ال Attribute grammer

طيب ايش يسوي Attribute grammer؟

خلونا نشوف المقطع هذا بعدها نرجع نكمل كلام هنا

https://www.youtube.com/watch?v=Im3QkQp_s_M

طيب اتمنى تكون وصلت الفكره

(ياجماعة ترا المقاطع مرا مهمة لاحد يسحب عليها)

بالعربي عندك طرف ايمن وطرف ايسر وانت لازم تحدد نوع الطرف الايسر بدلالة الطرف الايمن باستخدام ال 4 قواعد هذي

An Attribute Grammar for Simple Assignment Statements

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → A | B | C
   Semantic rule: <var>.actual_type ← look-up (<var>.string)

The look-up function looks up a given variable name in the symbol table and returns the variable's type.

على فكره برضو بالسلايدات 29,28,27 كلها فيها امثله برضوا حتوصلكم الفكره

اتمنى تفكوا السلايدات وتشوفوها مررا سهلة خصوصا لو فهمت المقطع الي فوق

Attribute Grammars: Definition

- Let $X_0 \rightarrow X_1 \ldots X_n$ be a rule
- Functions of the form $S(X_0) = f(A(X_1), \ldots, A(X_n))$ define *synthesized attributes*
- Functions of the form $I(X_j) = f(A(X_0), \ldots, A(X_n))$, for $i <= j <= n$, define *inherited attributes*
- Initially, there are *intrinsic attributes* on the leaves

خلينا نشرحها الثلاثة (شكرا ♥MOAB)

ال synthesized : هوا ان الجزء الي موجود في اليسار يعتمد على شي في اليمين زي موجود في رقم 2و3

ال inherited : هوا العكس, الي في اليمين يعتمد على شي في اليسار زي الي موجود في رقم 1

ال intrinsic : هوا يشبه ال synthesized بس اللهم ان الشي الي على اليمين يكون terminal زي رقم 4

طيب نجي للـ Regular Expressions

Metacharacter	Description	Examples
character	Any literal letter, number, or punctuation character (other than those that follow) matches itself.	apple matches apple.
(pattern)	Patterns can be grouped together using parentheses so that they can be treated as a unit.	see following
.	Match a single character (except linefeed).	s.t matches sat, sit, sQt, s3t, s&t, s t,...
?	Match zero or one of the previous character/expression. (When immediately following ?, +, *, or {min,max} it prevents the expression from using "greedy" evaluation.)	colou?r matches color, colour
+	Match one or more of the previous character/expression.	a+rgh! matches argh!, aargh!, aaargh!,...
*	Match zero or more of the previous character/expression.	b(an)*a matches ba, bana, banana, bananana,...
{number}	Match exactly number copies of the previous character/expression.	.o{2}n matches noon, moon, loon,...

.

هنا ال(.) يستبدلها مع اي شي ف الحياة, رقم او حرف او رمز(عدا النزول لسطر تحت – يعني ماحيخليك تسوي newline)

؟

هنا ال؟ يعني يا انه يحذف الحرف الي قبله او يخليه. مثلا لو عندنا

ca?r it will be equal to cr or car

او يشيله a يعني يا يخلي حرف الـ

+

يعني يا يحط تكرار واحد من شي معين او اكثر (ماله نهايه) مثلا {abc, abcabc, abcabcabc, abcabcabcabc} (abc)+ it will be equal to

*

نفس فكره ال + بس هنا فيه احتمال واحد اضافي انه ما يحط شي يعني فاضي مثلا {abc, abc,.....} (abc)* equal to

{n}

هنا لو حطينا اي رقم مكان n راح يخلي الررقم الي قبلها يتكرر بعدد تكراراتها مثلا {aabbbc, abbbbc, acbbbc,adbbbc} a.b{3}c equal to

لاحظ يا عسل هنا استخدم . وزي ماقلنا فوق مكانها بيحط اي حرف او رقم او رمز والقوس حق ال n حط قبله 3 حروف لأن هذي وظيفته

Metacharacter	Description	Examples
{min,max}	Match between min and max copies (inclusive) of the previous character/expression.	kabo{2,4}m matches kaboom, kabooom, kaboooom.
[set]	Match a single character in set (list and/or range). Most characters that have special meanings in regular expressions do not have to be backslash-escaped in character sets.	J[aio]b matches Jab, Jib, Job [A-Z][0-9]{3} matches Canadian postal codes.
[^set]	Match a single character not in set (list and/or range). Most characters that have special meanings in regular expressions do not have to be backslash-escaped in character sets.	q[^u] matches very few English words (Iraqi? qoph? qintar?).
|	Match either expression that it separates.	(Mi|U)nix matches Minix and Unix
^	Match the start of a line.	^# matches lines that begin with #.
$	Match the end of a line.	^$ matches an empty line.

{min,max}

هنا نفس فكرة ال n حقتنا الي قبل شوي بس هنا راح يكون عندنا بداية ونهايه يعني لها حدود زي ما انتو شايفين ف الجدول حاط لكم مثال فوق

[set]

يعني حياخذ اي حرف من الي بين الاقواس وبيحطه مثلا { FC[KAU]IT it equals {FCKIT, FCAIT , FCUIT

زي ما انتو شايفين ياخذ واحد بس من الحروف الي بين الاقواس ويحطهم

[^set]

بكل بساطه راح يسوي زي الي فوق بس العلامه ذي ^ يعني العكس مثلا {bdfgh.....xyz} [^ace] equals to

راح يجيب العكس يعني المكملة

|

ذي معروفه يواد, راح يختار يا اليمين او اليسار ويحطهم مثلا [FC|IT]KAU equals to {FCKAU, ITKAU}

^

انتبه فيه فرق بينه وبين لمن يكون جوا اقواس الي تشبه ذي [] المهم, انه يعني يكون يساوي بداية الكلمة نفس الحرف مثلا ^s لازم الكلمة تكون بدايتها s

$

زي الي فوق بس هذا يكون لنهاية الكلمة , مولازم اشرح واضحة صح ؟

خلاص امزح بشرح طيب ولا يهمك, المهم يعني مثلا $abc لازم الكلمة تكون نهايتها فيه abc زي kauabc او fcitabc

Metacharacter	Description	Examples
\	Interpret the next metacharacter character literally, or introduce a special escaped combination (see following).	* matches a literal asterisk.
\n	Match a single newline (carriage return in Python) character.	Hello\nWorld matches Hello World.
\t	Match a single tab character.	Hello\tWorld matches Hello World.
\s	Match a single whitespace character.	Hello\s+World matches Hello World, Hello World, Hello World,...
\S	Match a single non-whitespace character.	\S\S\S matches AAA, The, 5-9,...
\d	Match a single digit character.	\d\d\d matches 123, 409, 982,...
\D	Match a single non-digit character.	\D\D matches It, as, &!,...

\n

هذي ايزي, انزل سطر جديد والمثال موجود فوق بالجدول

\t

حط مسافه الي هي tab وتقريبا تتكون من 4 مسافات لمن نضغط على الكيبورد العادي بس ال tab تسويلك ال 4 بسرعه تختصر عليك الوقت

\s

تطنش وجود المسافات, طبعا هوا حط + يعني ممكن يتساوا مع اي كلمة فيها مسافه وحدة او اكثر, شوف المثال فوق وبتفهم باذن الله

\S

تشبه الي فوق بس هنا يشيل المسافات وما يخليهم

\d

يستبدلها مع اي رقم من 0 الى 9 يعني كل \d راح يحط مكانها رقم من بين ال 0 الى 9 , شوف الجدول فيه مثال ي حلو

\D

زي الي فوق بالزبط بس هنا راح يستبدلها ب اي حاجة في الحياة ماعدا الارقام, ممكن يحط حروف او رموز اي شي

شايفين انها سهله صح ؟لاتخاف ان بس ركز وطبق على الموقع هذا الي تحت عشان تثبت فهمك

والله يوفقكم

https://regex101.com/

CH.5

Names

- Design issues for names:
  - Maximum length?
  - Are connector characters allowed?
  - Are names case sensitive?
  - Are special words reserved words or keywords?

هنا يقولك اللغات وكيف تتعامل مع الاسماء (احفظها ترا مراااااا مهمة) طيب بعد ماتحفظها تعال نشرحها ي حلو

ال length وهي طول اسم المتغير, طبعا فيه انواع, هذي الي موجودة تحت احفظها...

هنا يقولك كل لغة وكم طول اسماء المتغيرات الي تستقبلها, انت احفظ وبس

- Length
  - If too short, they cannot be connotative
  - Language examples:
    - FORTRAN I: maximum 6
    - COBOL: maximum 30
    - FORTRAN 90 and ANSI and C: maximum 31
    - Ada and Java: no limit, and all are significant
    - C++: no limit, but implementers often impose one

نجي للشي الثاني وهو ال case sensitivity

- Case sensitivity
  - Disadvantage: readability (names that look alike are different)
    - worse in C++ and Java because predefined names are mixed case (e.g. `IndexOutOfBoundsException`)
  - C, C++, and Java names are case sensitive
    - The names in other languages are not

يعني هل اللغة حساسه لحاله الاحرف ؟ (كبير, صغير) زي: A ماتشبه a وزي كذا

- Special words
  - An aid to readability; used to delimit or separate statement clauses
    - A *keyword* is a word that is special only in certain contexts, e.g., in Fortran
      - `Real VarName` (`Real` *is a data type followed with a name, therefore* `Real` *is a keyword)*
      - `Real = 3.4` (`Real` *is a variable)*
  - A *reserved word* is a special word that cannot be used as a user-defined name

الشي الثالث Special words

تنقسم قسمين

الأول : keyword

هنا يقولك ان لغة Fortran يمديك فيها تستخدم كلمة Real كـ keyword وممكن تستخدمها كمتغير بالعربي كذا زي الجافا عندنا مثلا كلمة int هذي محجوزه ومستحيل تستخدمها كأسم متغير مثل string int, boolean int

بس في ال fortran يمديك (يعني في الfortran يمديك تستخدم نوع المتغير كنوع او كأسم)

زي المثال الي فوق Real مرة استخدمها كنوع متغير, ومرة كأسم متغير, عكس الجافا وكذا

ملاحظة ي عسل : ترا real في لغة Fortran تعتبر data type زيها زي ال int, string, boolean في الجافا

الثاني : reserved word

الي هي الكلمات المحجوزة في اللغة ومايمديك تستخدمها كمتغير, زي ال int, string, boolean في الجافا

Variables

- A variable is an abstraction of a memory cell
- Variables can be characterized as a sextuple of attributes:
  - Name
  - Address
  - Value
  - Type
  - Lifetime
  - Scope

هنا يقولك ان المتغيرات لها سمات او صفات معينة, وهم 6 اشياء (احفظها ترا جداً مهمة دايم تجي كخيارات)

يتبع للي فوق...

طيب خلينا نلخصهم في جدول حلو وجميل وكيوت عشان يسهل عليك الحفظ باذن الله

الخاصية	المعنى
Name	بكل بساطة, الاسم (يقولك ان مو كل المتغيرات لازم يكون عندها اسماء.. بعضهم ايه بعضهم لا)
Address	عنوان المتغير في ال memory (يقولك ان احيانا المتغير يتغير عنوانه الي موجود في الميموري خلال تشغيل البرنامج)
Value	المحتوى او القيمة الي فيه
Type	نوعه .... int,float,string الخ
lifeTime	هذي لازم تركز فيها لانها تلحس شوي شوف ال life time يعني حياة المتغير ف البرنامج ايش يعني؟؟؟ يعني مثلا فلنفترض عندي برنامج فيه متغير اسمه x داخل دالة اسمها sum دحين لمن نخش الدالة ونوصل للسطر حق تعريف المتغير, هنا البرنامج حيحط المتغير في الميموري وبيحجز له مكان, بس لمن يخلص نهاية الميثود ويطلع منها هنا بيصير deallocation للمتغير وبينحذف المتغير (هذا في حالة ان المتغير كان stack- dynamic طبعا, لاكن لو كان static ماراح ينحذف) هذا معنا الlife-time او حياة المتغير

scope	يعني المدى حق الدوال او الكلاسات مثلا في الدالة الي شرحناها فوق الscope حق الدالة sum يبدا من بداية فك القوس الى قوس النهاية (بالعربي المدى الي يمديك تشوف فيه المتغير او يمديك تستخدمه وتناديه variable can be visible)

Static and Dynamic Binding

- A binding is *static* if it first occurs before run time and remains unchanged throughout program execution.
- A binding is *dynamic* if it first occurs during execution or can change during execution of the program

زبدة الكلام ذا يقولك

ان ال static لمن يسوي له تعريف مثلا static int x = 5 بعد ما يخلص الكود ويطلع من الميثود ماراح ينحذف ال address في الميموري

ان ال dynamic لمن يسوي له تعريف مثلا int x = 5 بعد ما يخلص الكود ويطلع من الميثود راح ينحذف ال address في الميموري

فيه مثال تحت ف الصفحة التالية راح يشرح لك ال static, dynamic مززززبوط

Explicit/Implicit Declaration

- An *explicit declaration* is a program statement used for declaring the types of variables
- An *implicit declaration* is a default mechanism for specifying types of variables (the first appearance of the variable in the program)
- FORTRAN, BASIC, and Perl provide implicit declarations
  - Advantage: writability
  - Disadvantage: reliability (less trouble with Perl)

مافهمت شي من الطلاسم ذي ؟ تعال افهمك

شوف عندك نوعين

الاسم	المعنى
explicit	غير ضمني
implicit	ضمني

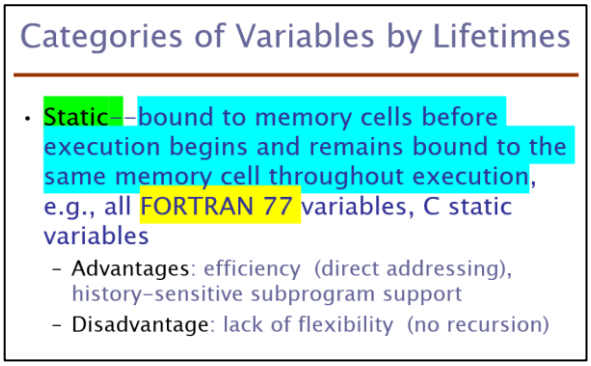
طيب ايش يعني ؟

يعني ال ضمني انت مو لازم تكتب نوع المتغير مثلا int, string, boolean اللغة هي بنفسها تعرف لو شافت رقم مثلا تعرف انه int

مثل : python, fortran, basic

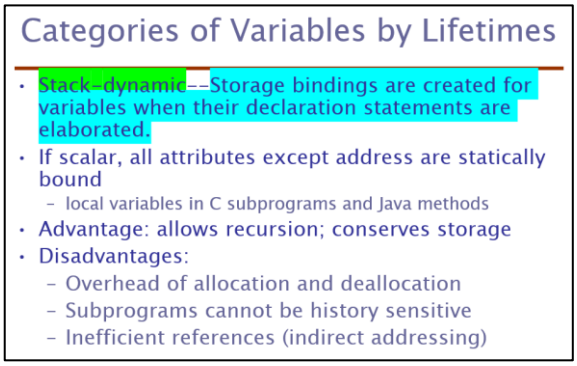
الغير ضمني هو العكس ي باشا, لازم تكتب نوع المتغير ولا البرنامج بيعطيك ايرور

مثل : java, C, C++




```
#include <stdio.h>

void fun1 (){
   static int x = 7;
   printf("%d\n",x++);
}
int main ()
{
   fun1();
   fun1();
   fun1();
}
```




```
#include <stdio.h>

void fun1 (){
    int x = 7;
   printf("%d\n",x++);
}
int main ()
{
   fun1();
   fun1();
   fun1();
}
```

طلاسم صح ؟

طيب تعال نفهمها حبة حبة

شوف كل مثال وجنبه كود

ايش الي حيصير ؟

لمن يتعرف المتغير الي قبله static لمن يطلع من الميثود حق fun1 ماراح ينحذف يعني لمن يرجع المين ويرجع يستدعي الميثود ثاني مرا راح يسوي تخطي لمرحلة تعريف المتغير وراح يعتمد على القيمة القديمه الي تم تعريفها اصلا من الاستدعاء الاول

طبعا النتيجة حق الكود حتكون 7,8,9

ليش ؟ عشان ف البدايه كان ب 7 وبعدين زوده ب 1 وخرج ورجع المين

بعدين رجع استدعى الفنكشن ثاني مرا بس هنا ماسواoverwriteللقيمه لانها static يعني بس حيطبعها ويزدها ب 1

والاستدعاء الثالث نفس الكلام, هنا ال address ثبته في الميموري وما مسحه وجلس يستخدمه كل شوي

بس ال stack-dynamic لمن يطلع من الفنكشن راح يمسحه من الميموري

يعني فكل مرا يتم استدعاء الفنكشن حق fun1 راح يعرف المتغير بقيمه 7 وبعدها يطبع ولمن يرجع المين حيمسح المتغير

ولمن يتم استدعاء الفنكشن مرا ثانيه راح يسوي متغير جديد بعنوان جديد

طبعا النتيجة حتكون 7,7,7

تنبيه: ترا لازم تكتب stack-dynamic لان انا في الكويز كتبت dynamic والدكتور عطاني 0

Categories of Variables by Lifetimes

- *Explicit heap-dynamic* -- Allocated and deallocated by explicit directives, specified by the programmer, which take effect during execution
- Referenced only through pointers or references, e.g. dynamic objects in C++ (via new and delete), all objects in Java
- Advantage: provides for dynamic storage management
- Disadvantage: inefficient and unreliable

Categories of Variables by Lifetimes

- *Implicit heap-dynamic*--Allocation and deallocation caused by assignment statements
  - all variables in APL; all strings and arrays in Perl and JavaScript
- Advantage: flexibility
- Disadvantages:
  - Inefficient, because all attributes are dynamic
  - Loss of error detection

طيب نشرح ال explicit and implicit heaps-dynamic

في ال explicit heap-dynamic ال opjects راح تتخزن او تنحذف في الميموري عن طريق المبرمج يعني لازم انت بنفسك تجبرها انها تتخزن

طبعا بتقولي كيف ؟ زي ما انت شايف هو يقولك ب استخدام كلمات مفتاحية معينه زي new, delete وهذي تستخدم في ال java, C

في ال implicit heap-dynamic ال opjects راح تتخزن او تنحذف من نفسها او خلينا نقول(من سياق الجملة) ياخي كلام فاضي المهم انت افهم الي فوق

وبتفهم ذي الي تحت

المهم نرجع لموضوعنا اسف خرجت عن النص شوي, المهم انها مو لازم تحط لها كلمات مفتاحية هي بنفسها تنحذف وتتخزن, كيف ؟ الله واعلم مثل APL, Perl

هذا مقال لدكتور في جامعة اجنبيه وضح نفس الكلام الي فوق

https://www.quora.com/What-is-Explicit-heap-dynamic

Strong Typing

- ***Strong typing*** facility is used to prevent mixing of different types

```
int x=5;
float y = 3.1*x;
```

- Advantage of strong typing: allows the detection of the misuses of variables that result in type errors
- Some programming languages make it easy to use a value of one type as if it were a value of another type. This is sometimes described as "weak typing“
- It is affected by:
  - Aliasing
  - Coercion
  - Type casting
  - Union (not all)

الزبدة هنا ال strong typing يقولك انو في بعض اللغات تكون مرااا حساسه للحالة حق المتغير وتكون ذكيه ولاكن بعضهم لا يمديك تلعب ب انواع المتغيرات زي ماتريد وتقدر تغير من نوع لنوع

مافهمت صح ؟ يب ادري كلامي مو مفهوم

طيب شوف الصوره ذي وبتفهم ب اذن الله

- Language examples:
  - Pascal is not strong typing: variant type
    - Assign any simple type of values in a variant variable. The type of a value stored in a variant is only determined at runtime

```
Program UsingVariant;
uses variants;
var
  v : variant;
  i : integer;
  r: real;
  as : ansistring;

begin
  i := 100;
  v:= i;
  writeln('Variant as Integer: ', v);
  r:= 234.345;
  v:= r;
  writeln('Variant as real: ', v);
  as:= ' bla bla bla';
  v:= as;
  writeln('Variant as AnsiString: ', v);
end.
```

طبعا في لغة ال Pascal عندهم متغير اسمه variant طبعا يمديه يتحول لاي نوع من المتغيرات زي ما انتو شاييفين في البرنامج تحول ل 3 انواع مختلفه وشغال تمام

برضو مافهمت ؟ هممممم طيب شوف هذي المقارنه وبتفهم باذن الله

Strong	Weak
add(5, '7') --> ?	add(5, '7') --> ?
concatenate('5', 7) --> ?	concatenate('5', 7) --> ?
Throws Error	NO Error
Explicit Conversion of DataType Required	Implicit DataType Conversion Done by Compiler
add(5, **converToInt**('7')) -->12	add(5, '7') --> 12
concatenate('5', **converToString**(7)) --> 57	concatenate('5', 7) --> 57

هنا ف ال weak قرا ال '7' كانها عدد مع انها اصلا المفروض تكون حرف, هنا ما يميز بين الانواع وال datatypes مثل ال Pascal فوق ف الصوره الي قبل ذي

عكس ال strong تماما, زي ما انتو شايفين على طووول اعطاه ايرور لانه حساس على نوع البيانات مثل ال java

- Aliasing
- Coercion
- Type casting
- Union (not all)

خلينا نعدي عليهم سرييييييييع هذولي الاربعه ونشرحهم

الاول Aliasing

هوا لمن يكون عندك اوبجكت, بس له اسمين. مثلا زيي انا عباره عن شخص واحد في العالم مافي نسخه ثانيه مني, بس عندي اكثر من اسم عبدالله, كثيري, عبدالله الكثيري, ابو راشد, عبدالله برمجة, القمر... المهم ممكن تكون فيه اسماء كثيره لنفس الشخص صح ؟ برضو الاوبجكت ممكن يكون له اكثر من اسم

وهذي الصوره توضح

```
#include <stdio.h>
union Data{
   int y;
   int x;
};
int main() {
   union Data data;
   data.y = 1;
   //data.x = 1;
   printf( "data.y : %d\n", data.y);
   printf( "data.x : %d\n", data.x);
   return 0;
}
```

```
data.y : 1
data.x : 1
```

الثاني coercion

طبعا ال coercion يشبه شوي ال casting

بس ال coercion يصير مباشرة من اللغة نفسها (من الكومبايلر), يعني مثلا زي لمن تخزن متغير int داخل متغير اخر من نوع float, ايش النتيجة ؟ المتغير بالاخير بيصير float لان ال float يعتبر اعم واكبر من ال int

وهذا الشي زي ما قلنا يصير تلقائي من الكومبايلر (يعتبر Implicit type conversion)

الثالث type casting

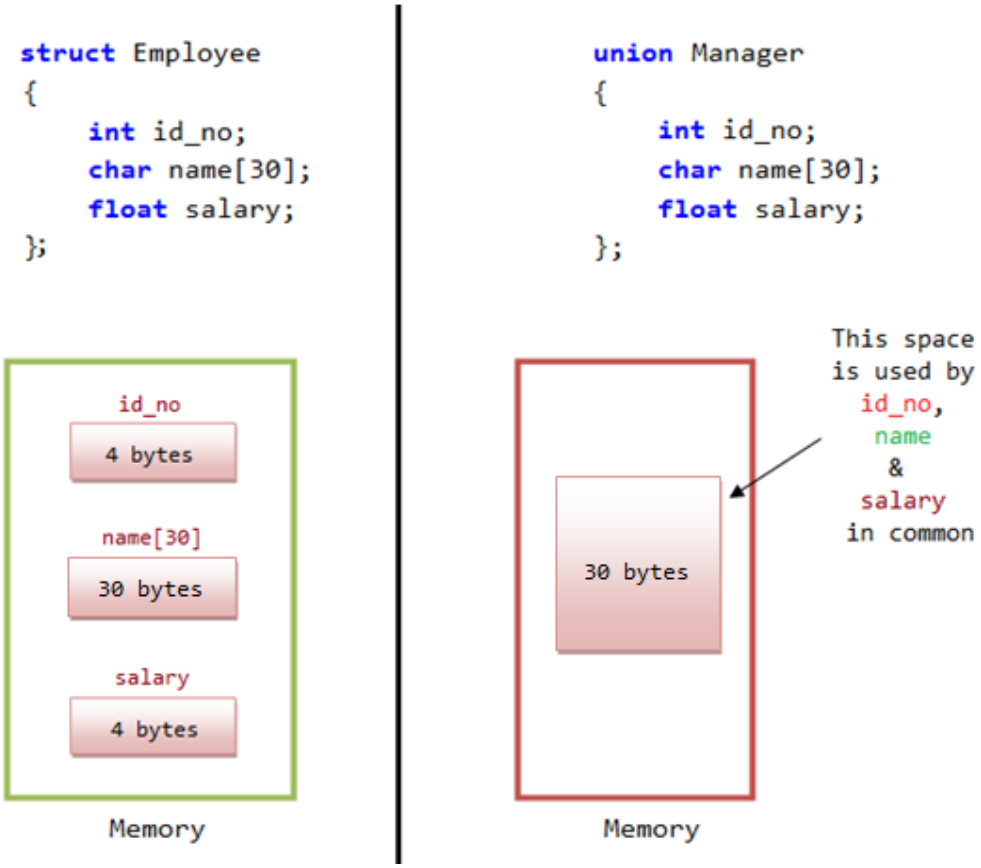
```
double d   = 10;
int i;
i = (int) d
```

Type Cast Operator

هو التحويل من نوع متغير كبير الى صغير, زي من float الى int, ولازم المبرمج يكتب هذا الشي لانها ماتصير تلقائياً من اللغة او الكومبايلر (Explicit type conversion)

الرابع Union

يعني انو المكان الواحد في الميموري راح يستقبل اكثر من نوع في نفس المكان او نفس ال opject او نفس ال array (ترا يشبه ال aliasing شوي بس بينهم فرق ترا)



نجي لل Type Compatibility ولها نوعين (ركز لها نوعين مختلفين بس فيه بينهم تشابه انتبه تتلخبط)

Name Type Compatibility

- *Name type compatibility* means the two variables have compatible types if they are in either the same declaration or in declarations that use the same type name
- Easy to implement but highly restrictive:

ال Name Type Compatibility يعني المتغيرات ماراح يكون لها نفس النوع الا في حالتين

1- لو كانوا في نفس ال declaration, زي int a,b,c;
هنا كل المتغيرات الثلاثه لهم نفس النوع الي هو int عشانهم تعرفوا مع بعض

2- لو كانوا متعرفين في اماكن متفرقه من البرنامج, بس استخدموا نفس الكلمة المفتاحية(نوع المتغير)
زي int d,e;
الحين المتغيرين d,e لهم نفس نوع المتغيرات a,b,c عشان كلهم اصلا تعرفوا بنفس الكلمة وهي int

تطبيقها سهل, بس راح تقييد سهولة البرمجة (حتصير البرمجة معقدة شوي)

Structure Type Compatibility

مثال →

```
class C1 {
  int a;
  string b;
  boolean c;
}

class C2 {
  int x;
  string y;
  boolean z;
}

main {
  C1 opj1;
  C2 opj2;
}
```

- *Structure type compatibility* means that two variables have compatible types if their types have identical structures
- More flexible, but harder to implement

ال Structure Type Compatibilit يعني لو كان عندنا متغيرين من كلاسين مختلفين, بس الكلاسين لهم نفس الهيكلة ونفس المتغيرات وانواعها ونفس كل شي, بس الفرق الوحيد اسم الكلاس

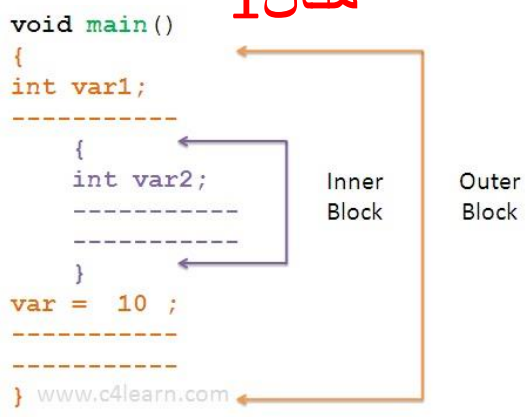
يعني مثل الصوره الي فوق ع اليمين الي فيها اطار ازرق هما كلاسين مختلفين بس عشانهم لهم نفس ال structure اللغة راح تحسبهم كانهم كلهم نفس الكلاس... ان شاء الله تكون وصلت الفكرة

التعامل معاها مرن, بس صعبه في التطبيق

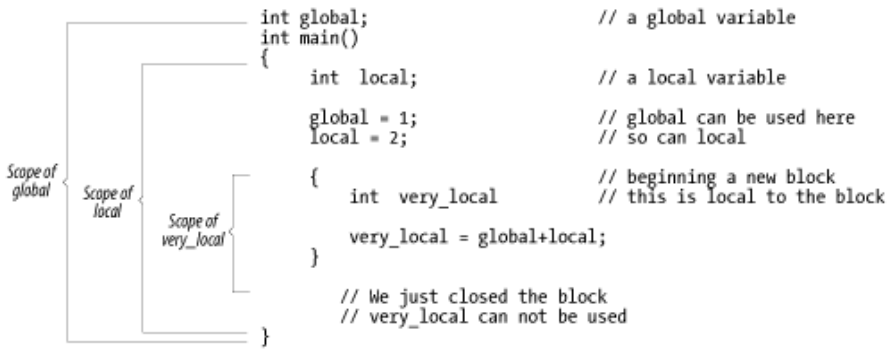
Variable Attributes: Scope

- The *scope* of a variable is the range of statements over which it is visible
- The *nonlocal variables (globals)* of a program unit are those that are visible but not declared there
- The scope rules of a language determine how references to names are associated with variables

مثال1



مثال2



طيب ايش هوا ال scope ؟ هو المجال الي يمكن من خلاله التعامل مع المتغير... طيب ايش يعني ؟

تعالوا نفسرها زيادة, شايفين الصوره الي فوق ع اليمين الي فيها المثال1 ؟

طيب ال scope حق ال var1 يعتبر خلال ال outer Block

وال scope حق ال var2 يعتبر خلال ال inner Block

مثال اخر, في المثال2

شوفو الصوره وبيكون موضح فيها كل شي, شايفين ال scope حق كل متغير؟ وصلت الفكرة ؟

طيب هذا مثال اخر

بس هنا يتكلم عن ال local scope

ايش هوا ال local scope ؟

شايفين ال x الاولى الي قيمتها 40 ؟

هذي ال scope حقها global لانها هي اصلا global

C Language (Local Scope)

```
#include<stdio.h>
int x = 40 ; // Scope : Whole Program (Global)
int main() {
      int x = 10 ; // Scope : In main
     {
           int x = 30 ; // Scope : within a bloack
           for(int i =0;i<5;i++){
               int x = 30 ; // Scope : within for bloack
           }
           printf("%d",x);
     }
  printf("%d",x);
}
```

Nested Subroutines

- Nested subroutines usually inherit the variable scope of the "parent" function
- Parameters are generally still possible where needed or helpful in sub-functions
- Examples of nested functions: Pascal, Ada and Php
- Reusable in another context

ال nested subroutine يعني انه يكون عندك دالة وسط دالة ثانية, طبعا هذا ماهو موجود في كل اللغات بس معروف في اللغات التالية Pascal,Ada,PHP

Pascal (Nested Subroutines)

```
function E(x: real): real;
      function F(y: real): real;
      begin
            F := x + y
      end;
begin
      E := F(3) + F(4)
end;
```

هذا مثال بلغة ال Pascal ومكتوب فيه الدالة F داخل ال scope حق الدالة E

ملاحظة لقيتها في احد المصادر على الانترنت(ممكن تجيك ف الاختبار): يقولك انه دايما لمن تنادي subroutine الaddress حقها على طول يروح يتخزن ف ال stack, ليش؟ عشان الstack يعتبر LIFO فدايم لمن يكون عندك دالة وسط دالة وسط دالة لمن يخلص من الدالة الاخيره بيرجع للي قبلها بعدين للي بعدها وعلى هذا الحال

الدكتور ذكر حاجة مهمة بس مو مكتوبة في السلايدات

انو لمن تخش subroutine وبعدين تخش واحد بعده واواحد بعده

يمكن يسألك سوال يقوولك وش هو ال active subroutine ؟

طبعا الجواب هوا ال subroutine الي موجود انت فيه حالياً او الي انت داخله وشغال فيه, سوا الاخير او الي بالنص او الاول

Lifetime and Scope

- Both lifetime and scope should exist to access a variable

Lifetime without Scope (C Language)	Scope without Lifetime (Java)

`#include <stdio.h>`
`void fun1 (){`
`  static int x = 0;`
`  .....`
`}`
`int main ()`
`{`
`fun1 ();`
`printf("%d\n", x);`
`}`
	Book b1;
	.
	.
	.
	.
	.

يقولك انه اذا تبغا توصل لأي variable عندك لازم يكون شيئين: lifetime and scope

طيب نسألكم سوال ذكي شوي!!!!

الصورة الي ع اليمين ليش تعتبر scope without lifetime ؟

متى ممكن نقول عنها انها scope with lifetime ؟

لا تطالع تحت فكر شوي درسناها ف البرمجة

.

.

.

.

.

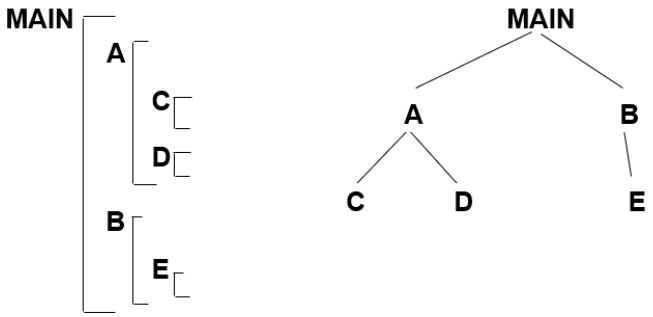
.

يب عشان ماحطينا كلمة new

طيب ياعسل انت وياه

الحين بنشرح الفرق بين ال static and dynamic scopes...

اولاً static scope



عندنا دالة ال main وسطها دالتين A,B وسط الA عندنا دالتين C,D ووسط الدالة Bعندنا دالة وحدة وهي E

طيب الحين زي ما احنى متعودين لو ابغا انادي الدالة D لازم انانديها من خلال الدالة A بس ما حقدر انانديها من خلال الدالة B مثلا صح ؟؟ يب طبعا صح

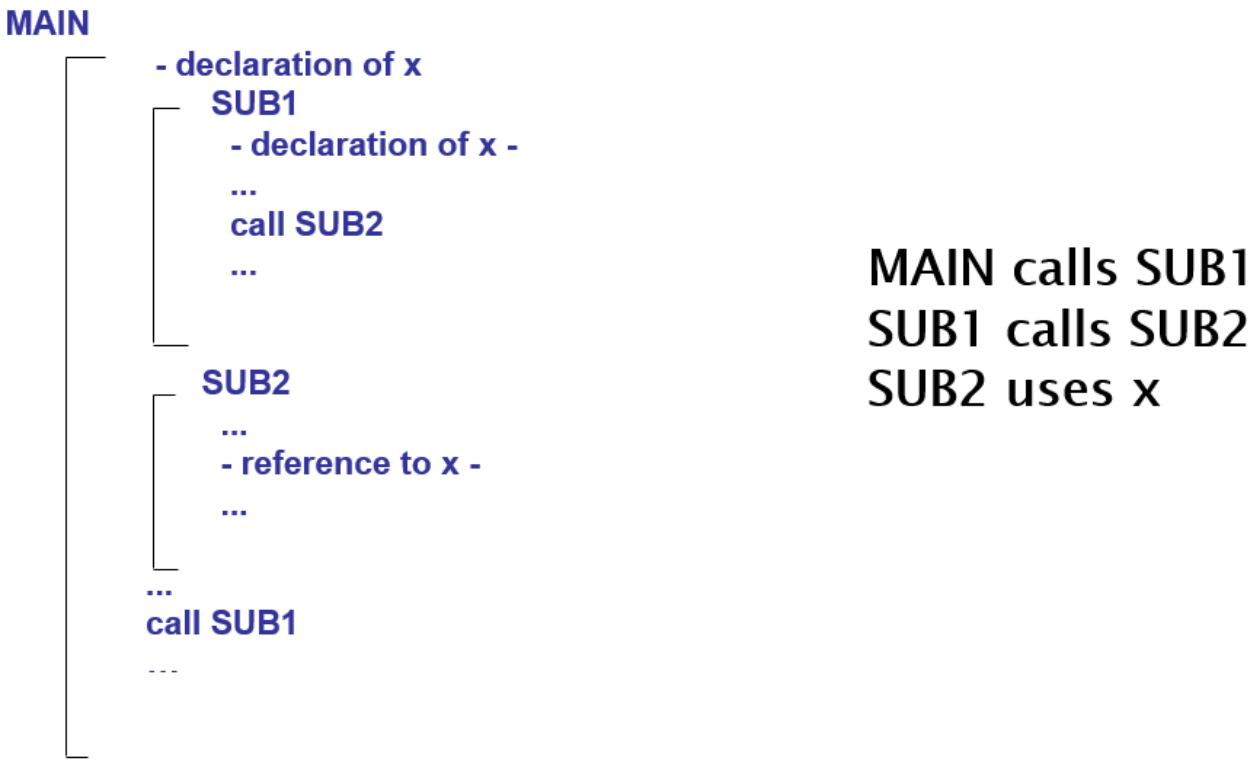
طيب لو اني ابغا مثلا انادي الدالة D من خلال B ايش حسوي ؟

عندي طريقتين لا ثالث لهما..

الاولى: احط الدالة D وسط الدالة B

الثانية : اخلي الدالة D تكون global يصير اقدر اوصل لها من كل مكان

ثانياً dynamic scope



طيب خلونا نشرحها حبة حبة

خلونا نستخدم مفهوم الstack ونسوي tracing للكود

Main[int x , sub1()]
Sub1[int x , sub2()]
Sub2[print(x)]

طيب الحين تتوقعوا حيطبع اي x ؟

طبعا الاخيره الي هي في sub1

مزايا ال dynamic scoping هي انه سهل ومريح لك كمبرمج, بس عيبه انه يتعبك في الtracing

طبعا هنا ال readability مرااا سيئ

طيب جبت لكم مقطع اسطوري يشرح نفس الفكرة الي فوق بس المثال حقه اطول ب شوي واستخدم فيه 3 متغيرات بدال متغير واحد, (المثال مرااا بطل انصحك تشوفه)

https://www.youtube.com/watch?v=v06ZFEmoNas

```
void fun() {
 int a, b, c;        /* Definition No.1 */
 ...
 while (.....) {
  int b, c, d;       /* Definition No.2 */
    ... <---------------- 1
  if (.....) {
     int c, d, e;    /* Definition No.3 */
     ... <---------------- 2
   }
     ... <---------------- 3
}
     ... <---------------- 4

} //end fun
```

هنا بيغا يخليك تتمرن بس على هرجة الscope and lifetime

شايفين الارقام ؟ 1 و 2 و 3 و 4

اعتبروا كل رقم عنده دالة طباعه System.out.print()

ايش راح تكون القيم الي حتنطبع ؟

مكان الطباعة	النتيجة

1	حيطبع ال a حق Definition no 1 و ال b,c,d حق definition no 2
2	حيطبع ال a حق Definition no 1 و ال b حق definition no 2 والc,d,e حق definition no 3
3	حيكون نفس النتيجة حق 1 (لانهم في نفس الscope)
4	حيطبع a,b,c حق definition no 1

وشكرا لكم😊

الله يوفقكم ويسعدكم,,, ولا تنسونا من دعائكم