الأكاديمية العربية الدولية
Arab International Academy

لغات البرمجة

Programing Languages

عبدالقادر العبدالله

كلية العلوم – تخصص البرمجة

المحاور

- مقدمة عن لغات البرمجة
- الخصائص الوظيفية و التقنية للغات البرمجة
- عبارات التحكم
- البرامج الفرعية وتنفيذها
- أنواع البيانات المجردة

المخرجات المتوقعة من الدرس

- تعريف لغات البرمجة وشرح الغرض منها
- التمييز بين الأجيال المختلفة من لغات البرمجة
- شرح كيفية عمل المترجم (Compiler)والمفسر (Interpreter)
- التعرف على أنواع عبارات التحكم
- تطبيق عبارات التحكم لحل مشكلات منطقية بسيطة
- التمييز بين أنواع البرامج الفرعية
- شرح كيفية تمرير المعاملات
- فهم فوائد البرامج الفرعية في تقليل التكرار وزيادة تنظيم الكود
- فهم مفهوم أنواع البيانات المجردة

مقدمة عن لغات البرمجة

1) أهمية لغات البرمجة : لغات البرمجة تعتبر الوسيلة الرئيسية للتواصل بين الإنسان والكمبيوتر والسبب هو أن معظم الآلات تعتمد على نظام العد الثنائي الذي يعد وسيلة غير فعالة وغير ملائمة للتواصل البشري.

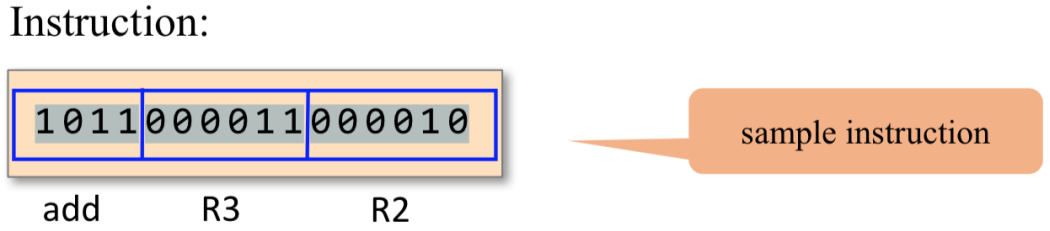
- لغات البرمجة تعمل كواجهة أساسية بين مستخدم الكمبيوتر ونظام التشغيل
- تحسين الإنتاجية بفضل الأوامر المبسطة
- زيادة القدرة على حل المشكلات المعقدة

❖ قد تكون بعض اللغات معقدة أو صعبة التعلم وهناك مشكلات في الأداء مقارنة بلغة الآلة

مقدمة عن لغات البرمجة

2) لغة الآلة (Machine Language) : اللغة التي يفهمها الكمبيوتر مباشرة وتتكون من مجموعة محددة من التعليمات التي يمكن للآلة تنفيذها يتم تخزين هذه التعليمات في ذاكرة الكمبيوتر وتفسر مباشرة بواسطة وحدة المعالجة المركزية و تكون لغة الآلة في شكل رموز ثنائية من خصائصها :

- لغة منخفضة المستوى : تعتبر أقرب لغة يمكن فهمها بواسطة العتاد (Hardware)
- تعليمات مباشرة : تتعامل مع وحدات مثل المسجلات (Registers) ومواقع الذاكرة (Memory Locations)
- تعقيد الصياغة : يتم التعبير عن الأوامر في صورة سلاسل ثنائية طويلة ومعقدة



مقدمة عن لغات البرمجة

❖ لغة الآلة (Machine Language) : للتغلب على بعض الصعوبات، تم إدخال رموز مساعدة (Mnemonics)

تستخدم لتمثيل التعليمات و سمح باستخدام الأرقام العشرية بدلا من الثنائية لتمثيل مواقع الذاكرة والمسجلات:

Mnemonic	Instruction
ADD	Add
SUB	Subtract
STA	Store
LDA	Load
BRA	Branch always
BRZ	Branch if zero
BRP	Branch if positive
INP	Input
OUT	Output
HLT	End program
DAT	Data location

```
000100 000000000001
```

تحميل قيمة من عنوان معين في الذاكرة إلى المسجل

```
LOAD 1
```

مقدمة عن لغات البرمجة

3) لغة التجميع الرمزية (Symbolic Assembly Language) : لغة تستخدم فيها الرموز النصية (Symbols) بدلا من العناوين الرقمية أو النسبية، مما يجعل البرمجة أكثر سهولة ومرونة وتم تطويرها على مرحلتين :

a) العنونة الرقمية الرمزية (Numeric Symbolic Addressing) : استخدام رموز رقمية ليس لها معنى منطقي لكنها تمثل مواقع في الذاكرة

b) العنونة الرمزية بالكامل (Fully Symbolic Addressing) : استخدام رموز نصية لكل من التعليمات (Instructions) والبيانات (Data)

مقدمة عن لغات البرمجة

العنونة الرقمية الرمزية	العنونة الرمزية بالكامل
تستخدم أرقاما كرموز لمواقع الذاكرة	تستخدم أسماء نصية للمتغيرات أو المواقع
لا تقدم معلومات إضافية حول طبيعة البيانات	الأسماء النصية توضح معنى البيانات أو الوظيفة
مثال : LOAD 5	مثال : LOAD TEMP
تتطلب من المبرمج تتبع معنى الأرقام	الأسماء النصية تجعل الكود أكثر وضوحاً

مقدمة عن لغات البرمجة

❖ مثال :

برنامج بسيط لتحميل محتوى متغيرين، جمعهما، ثم تخزين الناتج في متغير آخر.

العنونة الرقمية الرمزية

```
LOAD 2
ADD 3
STORE 4
```

العنونة الرمزية بالكامل

```
LOAD NUM1
ADD NUM2
STORE RESULT
```

لغة الآلة

```
0001 000000000010  ; LOAD value from memory location 2
0010 000000000011  ; ADD value from memory location 3
0100 000000000100  ; STORE result in memory location 4
```

مقدمة عن لغات البرمجة

لغة الآلة	لغة التجميع الرمزية
تعتمد على العناوين الرقمية أو الثنائية	تستخدم رموزا نصية للتعليمات والعناوين
يصعب تعديل الكود بسبب التغييرات في العناوين	أسهل في التعديل بفضل الرموز النصية
تتطلب معرفة تفصيلية بالبنية الداخلية للحاسوب	أكثر تجريدا وأقرب إلى اللغة الطبيعية

مقدمة عن لغات البرمجة

4) الأنظمة المبكرة لتطوير لغات البرمجة : تضمنت هذه الأنظمة أولى المحاولات لجعل البرمجة أسهل وأكثر فاعلية :

النظام	الهدف الرئيسي	السمات الرئيسية	الإنجازات والتأثير
Short Code (UNIVAC)	تسهيل البرمجة من خلال تقديم رمزيات مختصرة للعمليات الرياضية	- وظائف رياضية أساسية - اعتمد على صيغة ثابتة - الكود أقرب للآلة منه للغة طبيعية	أول محاولة جادة لتطوير لغة مختصرة للتعامل مع العمليات الرياضية البسيطة
Speedcoding (IBM 701)	تحسين الكفاءة البرمجية ودعم الحسابات العلمية بشكل أسرع	- أداة برمجية تدعم العمليات الرياضية المعقدة مثل الجذور التربيعية - مكتبة جاهزة لبعض الوظائف	ساعد في تقليل وقت كتابة الأكواد الرياضية
Laning and Zierler	تقديم إمكانية كتابة تعبيرات رياضية في صيغة قريبة من التدوين الطبيعي	- دعم التدوين الرياضي الطبيعي	اول نظام يدعم التدوين الطبيعي للتعبيرات الرياضية، مما سهل على المبرمجين كتابة أكواد مفهومة
BACAIC (IBM 701)	تسهيل العمليات النصية والطباعة	- أدوات لإدارة النصوص وطباعة البيانات - مكتبات جاهزة للمعالجة النصية	ساعد في تحسين إدارة النصوص والبيانات، ومهد الطريق لتحسين التعامل مع الملفات

مقدمة عن لغات البرمجة

5) لغة البرمجة الحديثة (Programing Languages) : لغة البرمجة تعتمد على مجموعة محددة من الرموز والأحرف و تمتلك قواعد تحكم كيفية تركيب الرموز لتكوين أوامر ذات معنى تستخدم لتطوير برامج قادرة على التحكم في الآلة وتنفيذ الأوامر من خصائصها :

- التجريد : تخفي التفاصيل الداخلية للجهاز، مثل تعليمات الجهاز أو طريقة تمثيل البيانات
- التوافق : تمكن البرامج المكتوبة بلغة واحدة من العمل على أجهزة مختلفة
- الكفاءة : توفر لغات البرمجة أداء مناسبا لمعظم التطبيقات

❖ لغات البرمجة تخفي التفاصيل التقنية المعقدة، مما يتيح للمبرمج التركيز على حل المشكلة بدلا من إدارة الموارد.

مقدمة عن لغات البرمجة

الميزة	الأنظمة المبكرة	لغات البرمجة الحديثة
المرونة	منخفضة، تعتمد على تنسيق محدد	عالية، تدعم تدوينات طبيعية متنوعة
سهولة الفهم	معقدة وتتطلب معرفة عميقة بالنظام	سهلة وقريبة من اللغة الطبيعية
الوظائف المدمجة	وظائف رياضية أساسية	وظائف رياضية، نصية، وبيانية متطورة
إدارة الملفات	محدودة أو غائبة	إدارة متقدمة للملفات وقواعد البيانات

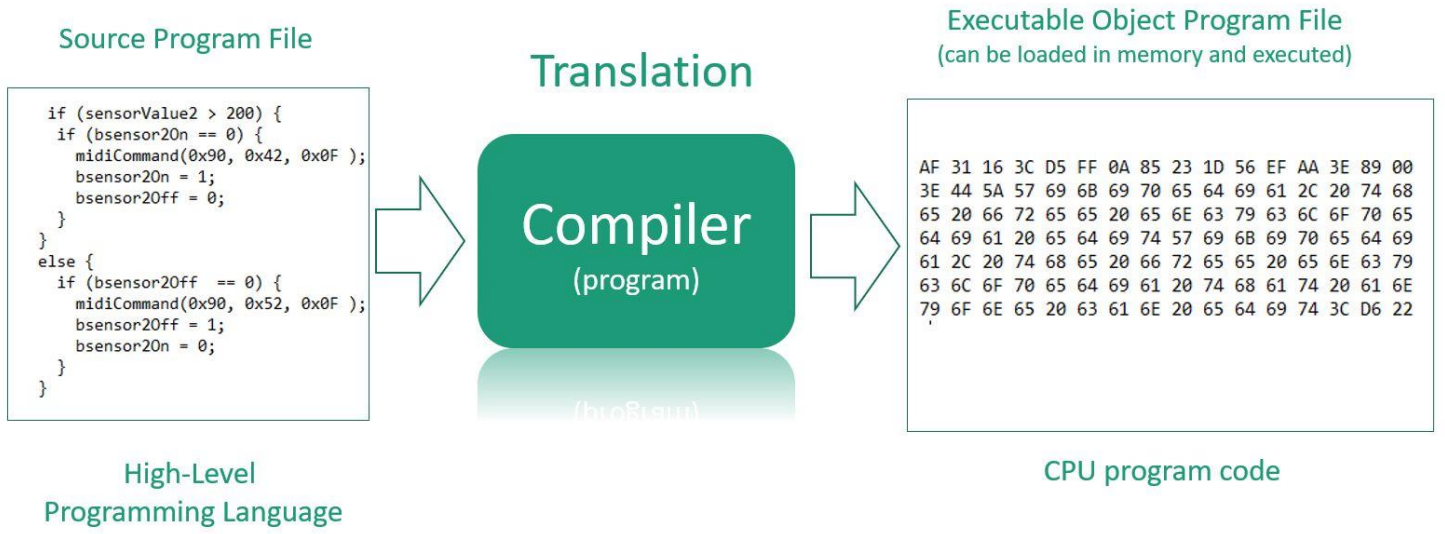
مقدمة عن لغات البرمجة

الميزة	لغات الآلة	لغات التجميع	لغات البرمجة منخفضة المستوى	لغات البرمجة عالية المستوى
التعريف	لغة برمجة مكونة من أوامر ثنائية مفهومة مباشرة بواسطة الحاسوب	لغة برمجة تستخدم رموزا نصية بدلا من الأكواد الثنائية	لغات قريبة من لغة الآلة مع بعض التجريد لتحسين سهولة البرمجة	لغات قريبة من اللغة الطبيعية توفر أدوات برمجية متقدمة للتطوير
المرونة	غير مرنة، مرتبطة بجهاز معين	أقل مرونة من اللغات عالية المستوى، تعتمد على الجهاز	مرونة محدودة، غالبا تستخدم لتطبيقات محددة	مرنة جدا، تستخدم لمجالات متعددة
الكفاءة	عالية جدا لأن الكود يتم تنفيذه مباشرة	كفاءة عالية ولكن أقل من لغات الآلة	كفاءة متوسطة، لكنها أفضل من اللغات عالية المستوى	كفاءة أقل مقارنة باللغات منخفضة المستوى
سهولة التعديل	صعبة جدا، تتطلب تعديلات دقيقة لكل تعليمة	صعبة ولكنها أسهل من لغة الآلة	متوسطة السهولة، ولكن تتطلب بعض المعرفة بالبنية الداخلية	سهلة جدا، تتطلب تغييرات قليلة للتكيف مع الأجهزة المختلفة
المثال	10101101 11010001	LOAD A ،ADD B	لغة C	Python, Java, C++, C# لغات
التطبيقات	أنظمة التحكم والبرامج الثابتة	كتابة برامج الأجهزة وأنظمة التشغيل	أنظمة التشغيل، تطوير الألعاب، البرمجة المضمنة	تطوير تطبيقات سطح المكتب، الويب، الأجهزة المحمولة

مقدمة عن لغات البرمجة

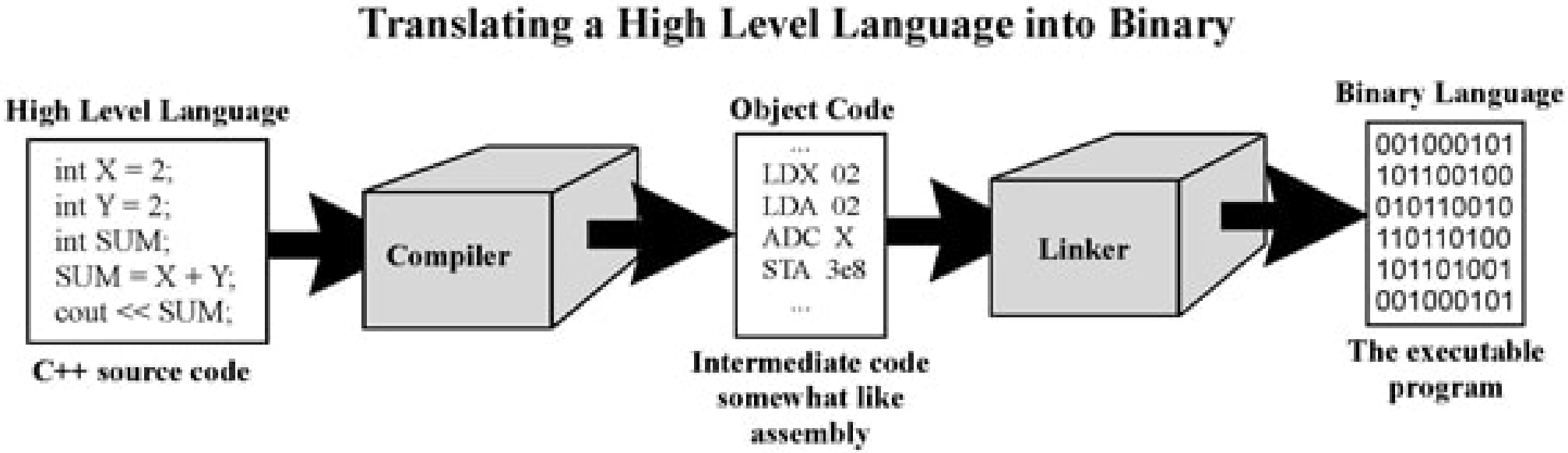
7) المصطلحات الأساسية (Basic Terminology) :

a) البرنامج المصدر (Source Program) : هو البرنامج المكتوب بلغة برمجة عالية المستوى ويمثل الكود الذي يكتبه المستخدم ويدخل إلى الحاسوب للحصول على نتائج وهو يتميز عن البرنامج الكائني الذي ينتج بعد الترجمة.



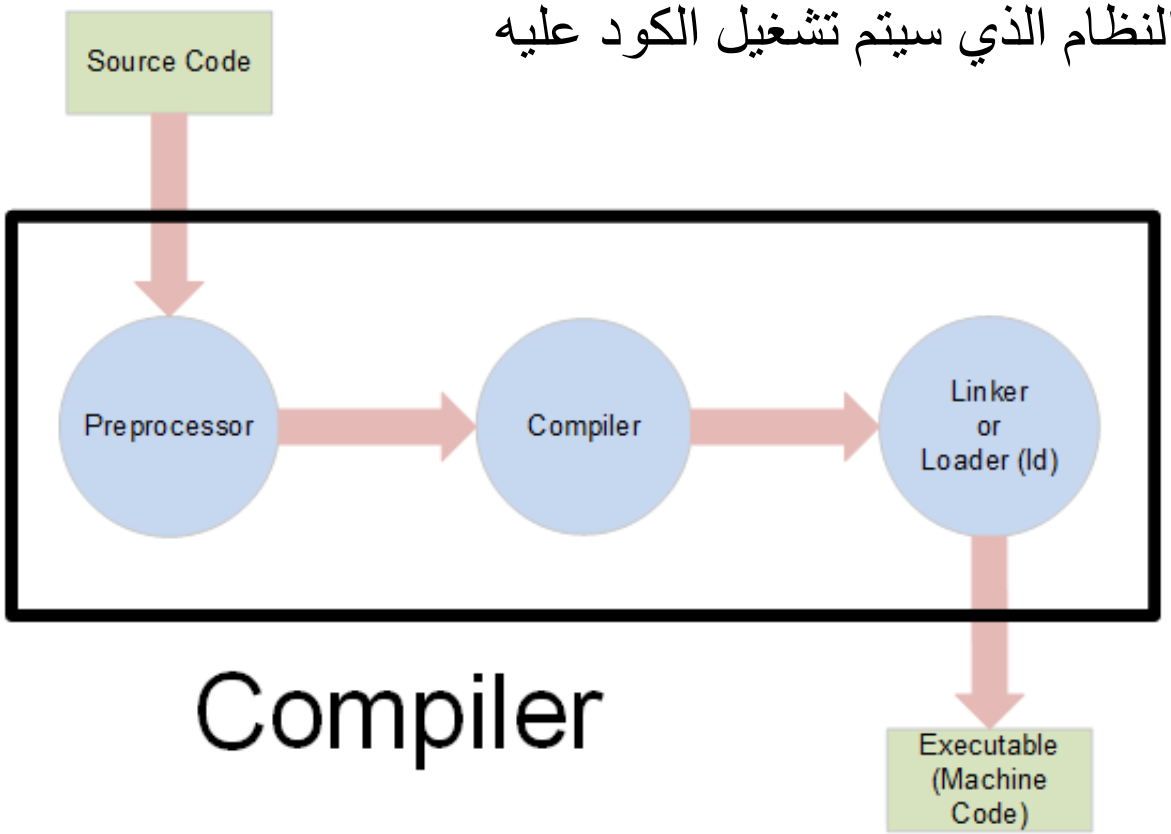
مقدمة عن لغات البرمجة

b) البرنامج الكائني (Object Program) : هو النتيجة التي تنتج عند ترجمة البرنامج المصدر وقد يظهر في عدة أشكال شكل ثنائي قابل للتنفيذ أو في صورة لغة تجميع رمزية



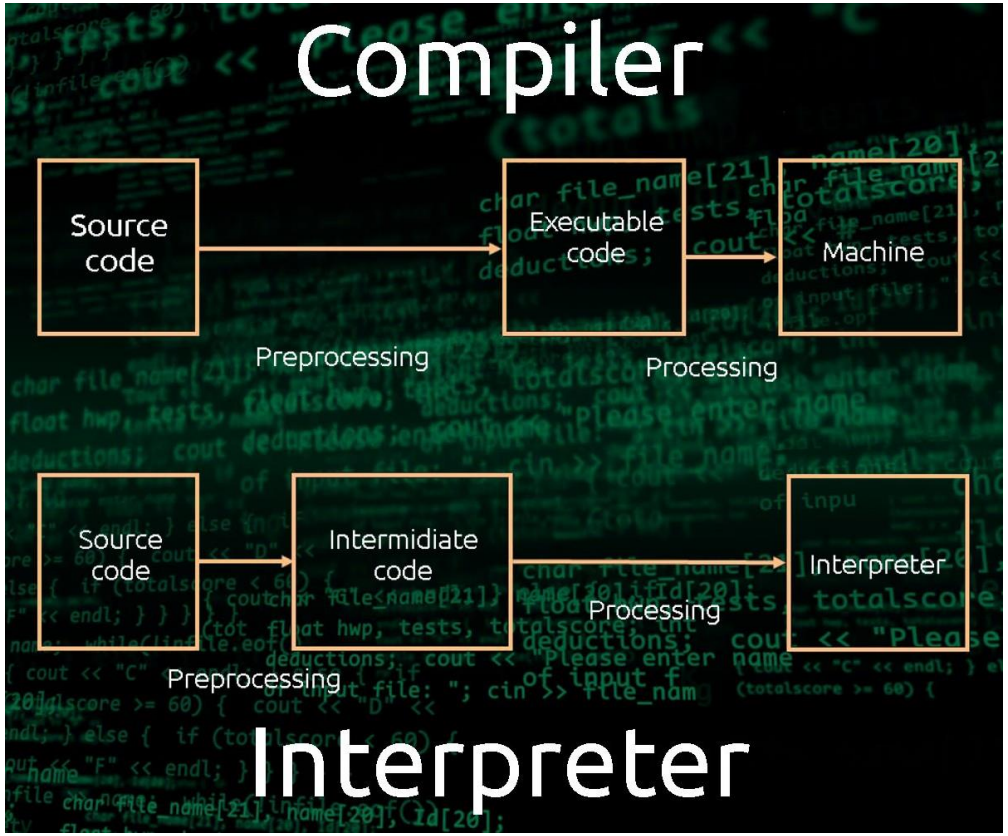
مقدمة عن لغات البرمجة

c) المترجم (Compiler) : هو برنامج يقوم بتحليل الكود المصدر و يترجم الكود المصدر إلى كود كائني يعتمد على اللغة المستخدمة والنظام الذي سيتم تشغيل الكود عليه



مقدمة عن لغات البرمجة

d) المفسر (Interpreter) : هو برنامج ينفذ الكود المصدر مباشرة، عادة سطرا بسطر أو وحدة بوحدة



❖ الفرق بين المترجم والمفسر :

- المفسر يعطي نتائج مباشرة (Output)
- المترجم ينتج كودا كائنيا يتم تنفيذه لاحقا

مقدمة عن لغات البرمجة

e) حزمة التطبيقات (Application Package) **:** مجموعة من التعليمات الجاهزة والمخصصة لمجال معين و قد يحتاج المستخدم إلى كتابة كود بلغة برمجة لتنظيم استدعاء التعليمات

من حيث	لغة البرمجة	حزمة التطبيقات
المرونة	عالية، تستخدم لتطبيقات متعددة	منخفضة، موجهة لأغراض محددة
الاستخدام	إنشاء برامج وحلول برمجية متنوعة	تنفيذ عمليات محددة مثل التقارير أو الإدارة
الوظائف الأساسية	توفر أدوات لإنشاء التعليمات أو الحزم	تعتمد على تعليمات جاهزة
التخصص	عامة أو موجهة لتطبيقات محددة	موجهة لمجال ضيق جدا

مقدمة عن لغات البرمجة

8) تصنيفات لغات البرمجة (Programming Language Categories) :

a) اللغات الإجرائية

b) اللغات غير الإجرائية

c) اللغات الموجهة للمشكلة

d) اللغات الموجهة للتطبيقات

e) اللغات ذات الأغراض الخاصة

f) لغة تعريف المشكلة

g) لغة وصف المشكلة

h) لغة حل المشكلة

k) لغة مرجعية

l) لغة النشر

m) لغة الأجهزة

مقدمة عن لغات البرمجة

الفئة	الهدف	الأمثلة	المرونة
إجرائية	تحديد خطوات حل المشكلة	COBOL، FORTRAN	عالية
غير إجرائية	وصف المشكلة دون تحديد الخطوات	مولدات التقارير	متوسطة
موجهة للتطبيقات	مخصصة لتطبيق معين	COGO، APT	منخفضة
أغراض خاصة	هدف محدد واحد	لغات الأنظمة المضمنة	منخفضة
تعريف المشكلة	تحديد المدخلات والمخرجات فقط	مولدات التقارير	منخفضة
حل المشكلة	تحديد الحلول الكاملة	Java، Python	عالية
مرجعية	تعريف رسمي للغة	ALGOL	منخفضة
النشر	تسهيل الطباعة والنشر	الصيغة الرمزية	متوسط
الأجهزة	إدخال مباشر إلى الحاسوب	تمثيلات برمجية للأجهزة	منخفضة

مقدمة عن لغات البرمجة

9) عوامل اختيار لغة البرمجة (FACTORS IN CHOICE OF A LANGUAGE) : يعتمد اختيار اللغة على عدة عوامل منها :

- ملاءمة اللغة لمجال المشكلة والمستخدمين المتوقعين
- كفاءة تنفيذ اللغة
- التوافق وإمكانية النمو
- الخصائص التقنية وغير التقنية
- توفر اللغة و المترجم الخاص بها على نظام التشغيل المطلوب

الخصائص الوظيفية والتقنية للغات البرمجة

1) **الخصائص الوظيفية (Functional Characteristics):** تشير إلى الجوانب غير التقنية للغات البرمجة، والتي لا تعتبر جزءًا من المواصفات الفنية للغة نفسها ترتبط باستخدام اللغة في السياقات العملية، مثل التكلفة الاقتصادية، الدعم، والتوافق مع الأنظمة المختلفة وتؤثر على :

- سهولة الاستخدام والتعلم : لغة مثل Python تعتبر سهلة التعلم والاستخدام
- التوافق والتحويل : لغة Java تستخدم على نطاق واسع بسبب قابلية برامجها للعمل على منصات مختلفة
- للتوحيد : لغات مثل SQL تعتبر معيارا في قواعد البيانات، مما يجعلها متوافقة على نطاق واسع بين مختلف الأنظمة.

الخصائص الوظيفية والتقنية للغات البرمجة

2) الخصائص الأساسية للغات البرمجة :

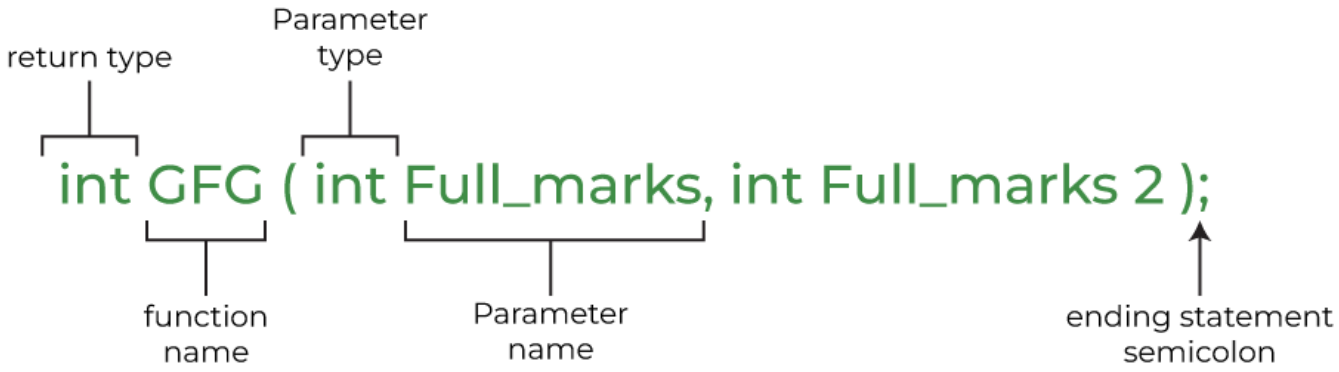
الخاصية	التعريف	مثال
العمومية	قدرة اللغة على التعامل مع مجموعة واسعة من التطبيقات والمشاكل	توفرمكتبات متعددة للتعامل مع أنواع مختلفة من التطبيقات مثل Python
البساطة	سهولة تعلم اللغة واستخدامها وتنفيذها	سهلة التعلم بسبب بناء جملها البسيط مثل Python
الإيجاز	التعبير عن العمليات بشكل مختصر	كتابة عملية معقدة بسطر واحد باستخدام في تعبيرات Lambda مثل Python
الطبيعية	تكون التعليمات قريبة من اللغة البشرية	توفر تعبيرات قريبة من اللغة الإنجليزية مثل COBOL
التناسق	استخدام نفس القواعد في جميع أجزاء اللغة	تعتمد قواعد واضحة للكتابة مثل Java
الكفاءة	فعالية اللغة من حيث الأداء والسرعة	لغة معروفة بكفاءتها العالية في الأداء C
سهولة التعلم	سهولة كتابة الكود البرمجي	سهلة القراءة والكتابة بسبب قواعدها البسيطة مثل Python

الخصائص الوظيفية والتقنية للغات البرمجة

3) الجوانب التقنية لتعريف للغات :

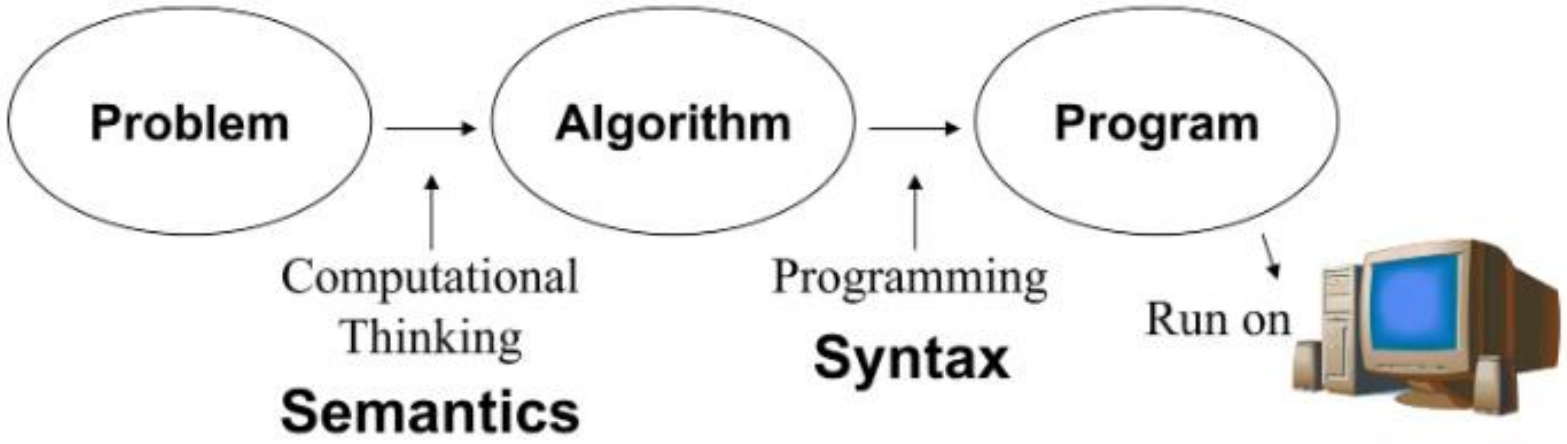
a) العناصر الأساسية لتعريف اللغة :

- النحو (Syntax) : يشير إلى القواعد والتراكيب التي تحدد كيفية كتابة الكود البرمجي يتعامل مع شكل الكود



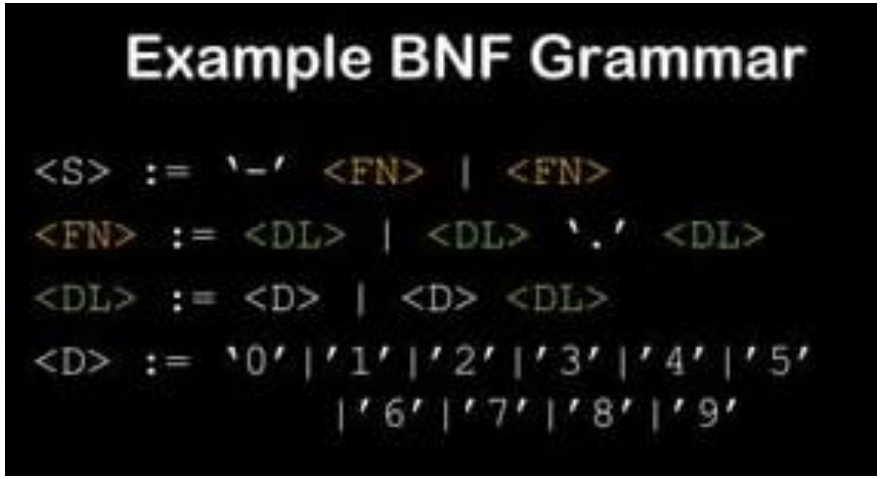
الخصائص الوظيفية والتقنية للغات البرمجة

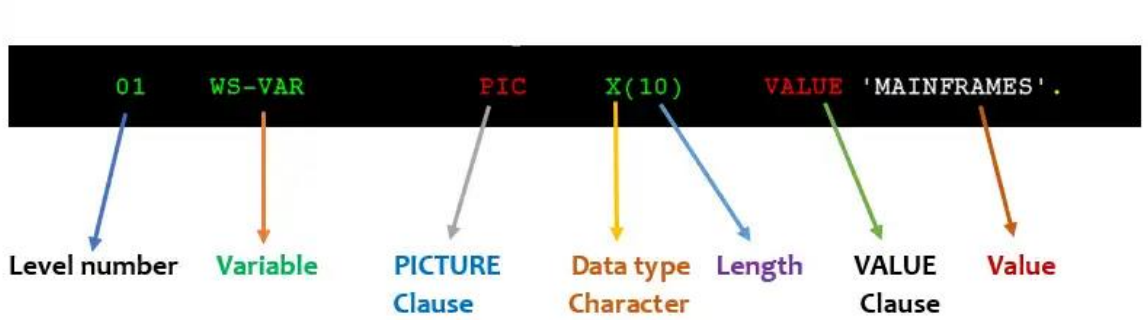
- الدلالات (Semantics) : تشير إلى المعنى أو الهدف من الكود المكتوب



الخصائص الوظيفية والتقنية للغات البرمجة

- البراغماتية (Pragmatics) : العلاقة بين المستخدم ومعاني الرموز
- تدوين اللغة (Formalized Notation) : تستخدم لغة وصفية تسمى " اللغة الوصفية " (Metalanguage) لتعريف اللغة من اشهر صيغها صيغة Backus-Naur (BNF) و صيغة COBOL





الخصائص الوظيفية والتقنية للغات البرمجة

b) أنواع وثائق لغات البرمجة :

- دليل المرجع (Reference Manual) : يحتوي على المواصفات الكاملة للغة باستخدام لغة وصفية رسمية يستخدم لتوضيح التفاصيل التقنية الدقيقة
- دليل المستخدم (User Manual) : يركز على تعليم المستخدمين كيفية استخدام اللغة
- دليل التنفيذ (Implementation Manual) : يركز على كيفية تنفيذ اللغة على أجهزة معينة و قد يتضمن معلومات عن قيود اللغة على الأجهزة المختلفة
- الوثائق المختصرة (Quick Reference) : عبارة عن ملخص قصير للغة يستخدم كمرجع سريع للمستخدمين المتمرسين

الخصائص الوظيفية والتقنية للغات البرمجة

4) العناصر الأساسية للغات البرمجة :

- البيانات ووصفها (Data and Its Description) : هي العناصر الأساسية التي يتم إجراء العمليات عليها داخل البرنامج قد تكون البيانات أرقاما، أو قوائم بأسماء وعناوين، أو صيغ رياضية، أو نصوصا وتساعد المترجم (Compiler) في فهم طبيعة المتغيرات

```c
#include <stdio.h>

int main() {
    // وصف البيانات
    char name[] = "Alice";    // نصوص
    int age = 25;             // عدد صحيح
    float height = 5.7;       // عدد عشري
    int is_student = 1;       // قيمة منطقية (تعني 1 True)

    // عرض البيانات
    printf("Name: %s\n", name);
    printf("Age: %d\n", age);
    printf("Height: %.1f\n", height);
    printf("Is Student: %d\n", is_student);

    return 0;
}
```

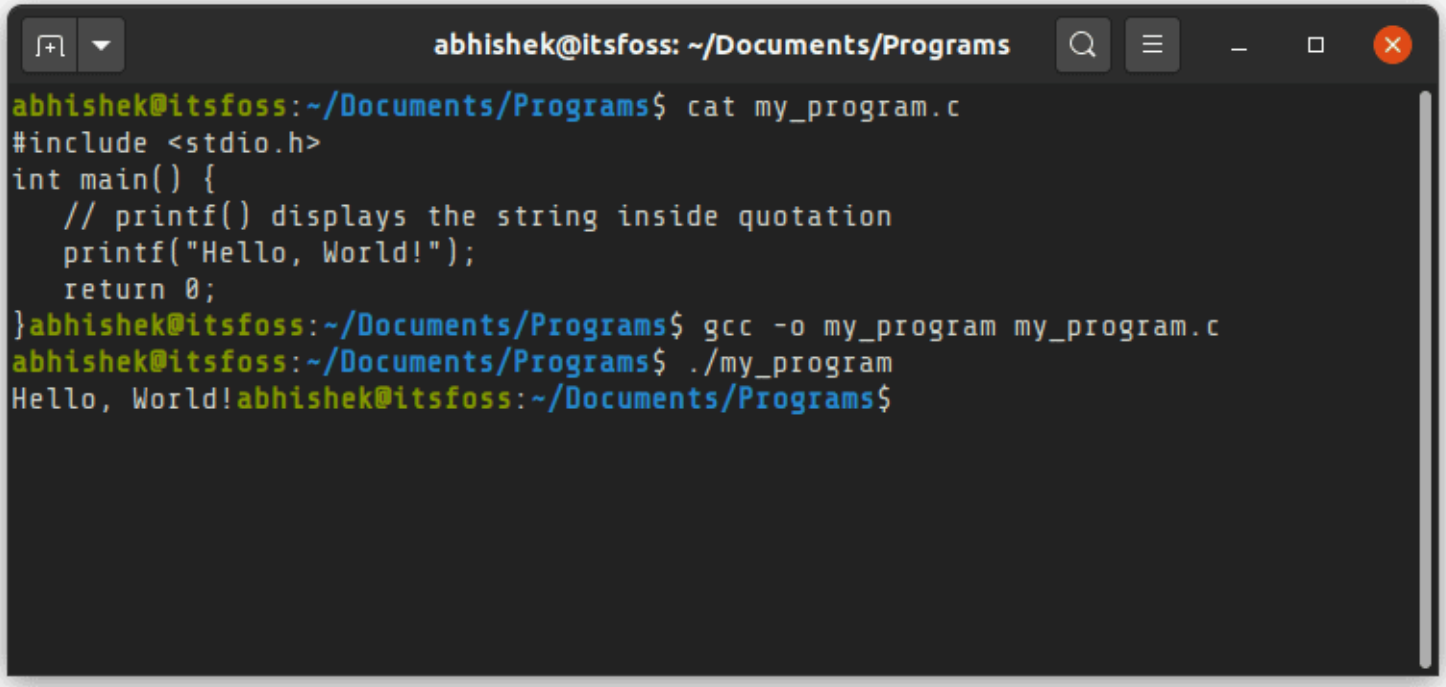
الخصائص الوظيفية والتقنية للغات البرمجة

- المشغلات (Operators) : هي رموز أو كلمات تستخدم لإجراء عمليات على البيانات داخل البرنامج تختلف أنواع المشغلات حسب نوع العملية مثل العمليات الحسابية أو المنطقية أو العلائقية

	Operators	Type
Unary Operator →	++, --	Unary Operator
Binary Operator	+, -, *, /, %	Arithmetic Operator
	<, <=, >, >=, ==, !=	Rational Operator
	&&, ||, !	Logical Operator
	&, |, <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator →	?:	Ternary or Conditional Operator

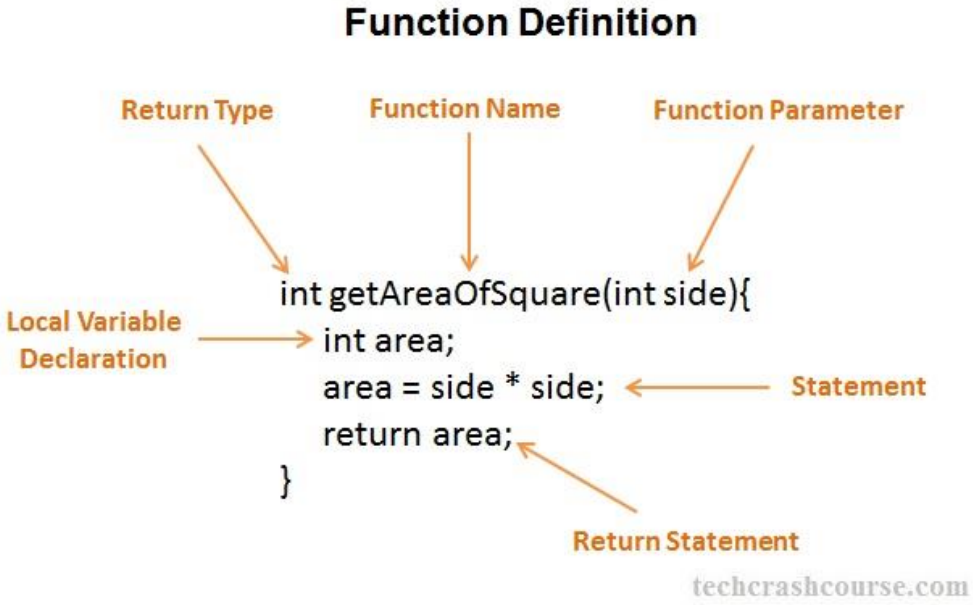
الخصائص الوظيفية والتقنية للغات البرمجة

- الأوامر (Commands) : الأوامر هي أساس أي لغة برمجة، وهي عبارة عن تعليمات يقوم البرنامج بتنفيذها أثناء التشغيل تمثل الأوامر الأفعال التي يمكن أن تقوم بها لغة البرمجة مثل تخصيص القيم، اتخاذ القرارات، أو تنفيذ عمليات معينة



الخصائص الوظيفية والتقنية للغات البرمجة

- التصريحات (Declarations) : هي جزء أساسي من كتابة الكود البرمجي حيث يتم استخدامها لتعريف المتغيرات والبيانات وتوضيح نوعها تساعد التصريحات المترجم (Compiler)على فهم البيانات التي سيتم استخدامها في البرنامج، مما يحسن من كفاءة البرنامج ويقلل الأخطاء.



الخصائص الوظيفية والتقنية للغات البرمجة

- توجيهات المترجم (Compiler Directives) : هي تعليمات خاصة يتم تضمينها في الكود لتزويد المترجم بمعلومات إضافية حول كيفية ترجمة الكود البرمجي هذه التعليمات لا تنفذ مباشرة في وقت تشغيل البرنامج، وإنما تؤثر على عملية الترجمة نفسها وتقوم بضبط بيئة الترجمة وتحسين الأداء.

Directive	Function
#define	Defines a **Macro** Substitution
#undef	Undefines a **Macro**
#indude	Includes a File in the Source Program
#ifdef	Tests for a **Macro** Definition
#endif	Specifies the end of #if
#ifndef	Checks whether a **Macro** is defined or not
#if	Checks a Compile Time Condition
#else	Specifies alternatives when #if Test Fails

الخصائص الوظيفية والتقنية للغات البرمجة

- المحددات (Delimiters) : المحددات هي رموز أو علامات تستخدم لتحديد حدود أجزاء الكود المختلفة في لغات البرمجة تلعب دورا أساسيا في تنظيم الكود وفصل التعليمات لضمان فهم المترجم(Compiler)أو المفسر (Interpreter) للبرنامج.

Delimiters

- Delimiters are used for syntactic meaning in C.

:	colon	used for label
;	semicolon	end of statement
( )	parentheses	used in expression
[ ]	square brackets	used for array
{ }	curly braces	used for block of statements
#	hash	preprocessor directive
,	comma	variable delimiter

الخصائص الوظيفية والتقنية للغات البرمجة

- بنية البرنامج (Program Structure) : بنية البرنامج تشير إلى الطريقة التي يتم بها تنظيم الكود البرمجي لتشكيل برامج قابلة للفهم، التطوير، والصيانة تتضمن هذه البنية القواعد التي تحدد كيفية ترتيب الأوامر والبيانات في البرنامج لتحقيق وظيفة محددة بشكل منظم

Structure of C Program

	1	`#include <stdio.h>`	Header
	2	`int main(void)`	Main
BODY	3	`{`	
BODY	4	`printf("Hello World");`	Statement
BODY	5	`return 0;`	Return
BODY	6	`}`	

الخصائص الوظيفية والتقنية للغات البرمجة

5) أسماء البيانات غير المجمعة (Non-Aggregate Data Names): تشير إلى المتغيرات أو الكيانات التي تخزن قيمة فردية واحدة فقط وهي بيانات بسيطة (Primitive Data Types) لا تحتوي على مجموعة من القيم أو عناصر فرعية بعكس البيانات المجمعة.

- الأعداد الصحيحة (Integer) : تمثل أرقاما صحيحة
- الأعداد العشرية (Floating Point) : تمثل أرقاما تحتوي على كسور عشرية
- النصوص (String) : تمثل سلسلة من الحروف أو النصوص
- القيم المنطقية (Boolean) : تمثل قيمتين فقط صحيح او خطا

```
int age = 25;
```

```
float salary = 4500.75;
```

```
char name[] = "Alice";
```

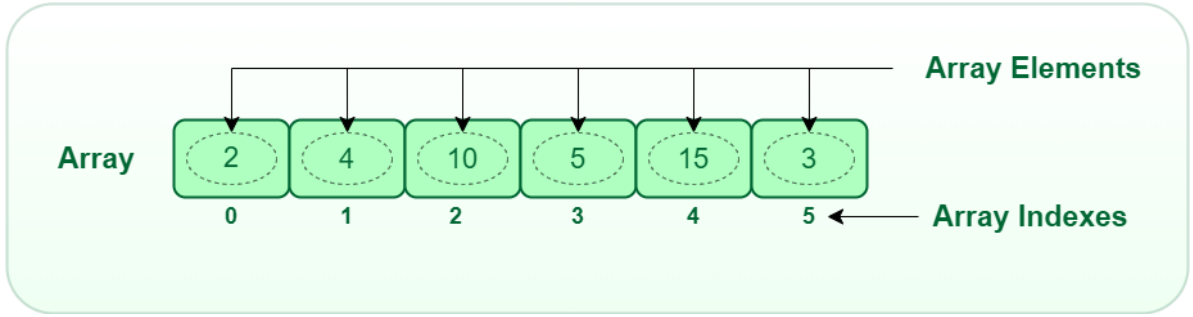
```
_Bool isStudent = 1;
```

الخصائص الوظيفية والتقنية للغات البرمجة

6) أسماء البيانات المجمعة (Data Names for Aggregates): تشير إلى المتغيرات أو الكيانات البرمجية التي تمثل مجموعة من البيانات المرتبطة ببعضها بحيث يمكن التعامل معها ككتلة واحدة أو بالوصول إلى العناصر الفردية بداخلها

- المصفوفات (Arrays) : مجموعة من القيم المتشابهة من حيث النوع، يتم الوصول إليها باستخدام فهرس

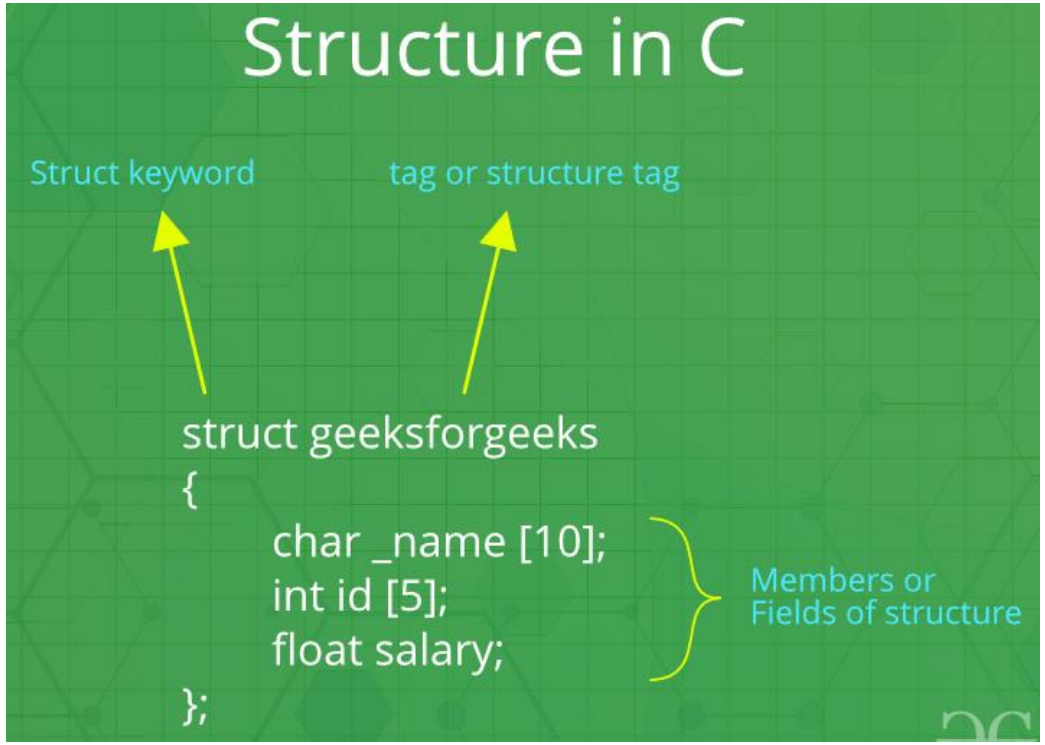
```
int numbers[5] = {1, 2, 3, 4, 5};
        printf("%d", numbers[0]);
```



الخصائص الوظيفية والتقنية للغات البرمجة

- الهياكل (Structures) : مجموعة من البيانات مختلفة الأنواع مجمعة تحت اسم واحد

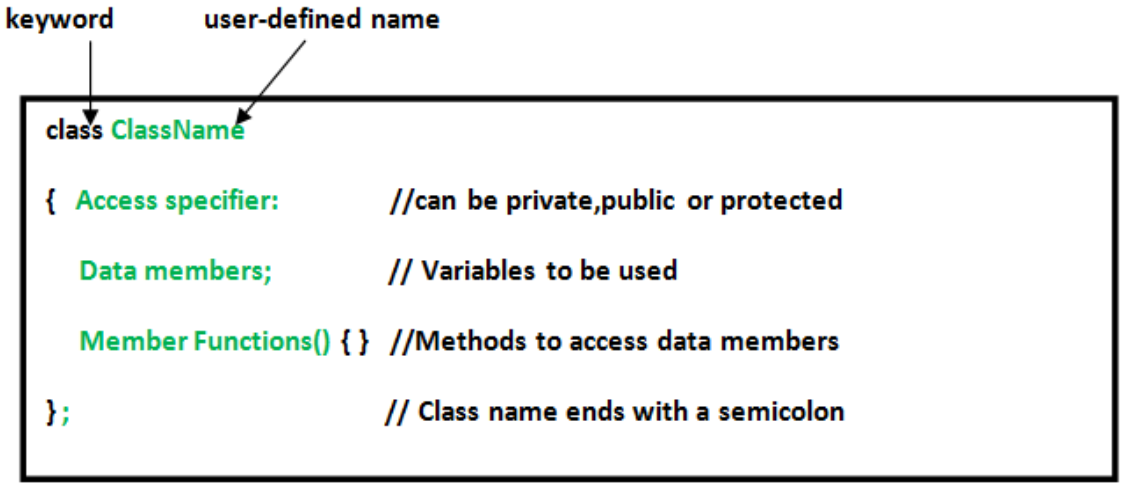
```c
struct Student {
    char name[50];
    int age;
    float gpa;
};

struct Student student1 = {"Alice", 20, 3.5};
printf("Name: %s, Age: %d, GPA: %.2f", student1.name, student1.age, student1.gpa);
```



الخصائص الوظيفية والتقنية للغات البرمجة

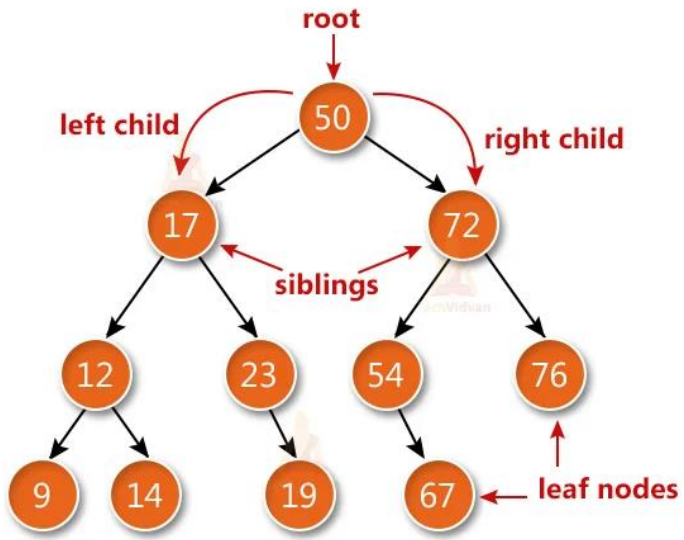
- الكائنات (Objects) : مفهوم يستخدم في البرمجة الكائنية حيث يمثل الكائن تجميعا للبيانات (الخصائص) والوظائف (السلوكيات)

```python
class Student:
    def __init__(self, name, age, gpa):
        self.name = name
        self.age = age
        self.gpa = gpa

student1 = Student("Alice", 20, 3.5)
print(f"Name: {student1.name}, Age: {student1.age}, GPA: {student1.gpa}")
```



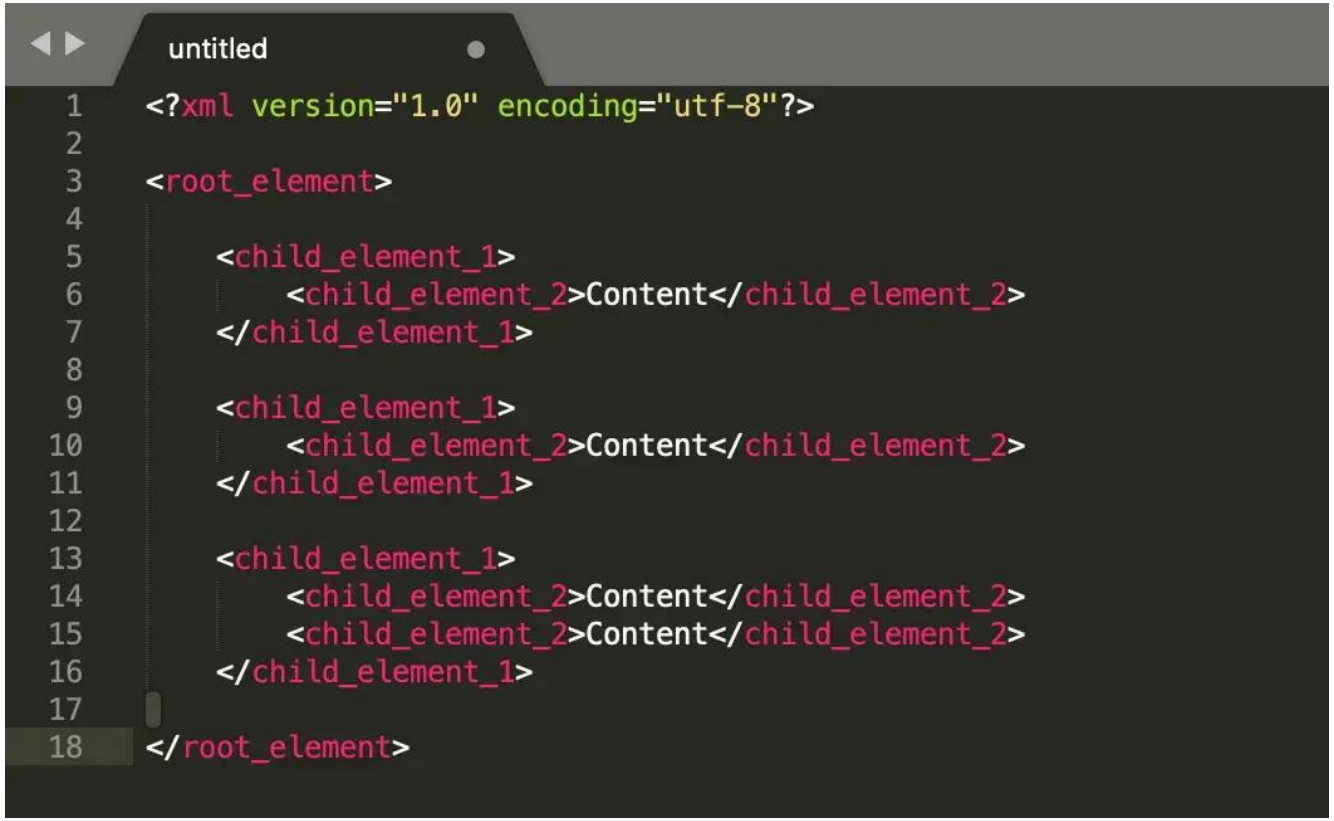
الخصائص الوظيفية والتقنية للغات البرمجة

- البيانات الهيكلية (Hierarchical Data) : بيانات يتم تنظيمها بطريقة تعتمد على التسلسل الهرمي تكون البيانات مرتبطة ببعضها البعض بعلاقات رئيسية وفرعية مثل :
  - الأشجار (Trees) : تمثل البيانات بشكل هرمي حيث يكون هناك جذر (Root) وأفرع (Branches)وأوراق (Leaves)



الخصائص الوظيفية والتقنية للغات البرمجة

- XML (Hierarchical Data Representation) : هي لغة تستخدم لتخزين ونقل البيانات بشكل منظم وهرمي



الخصائص الوظيفية والتقنية للغات البرمجة

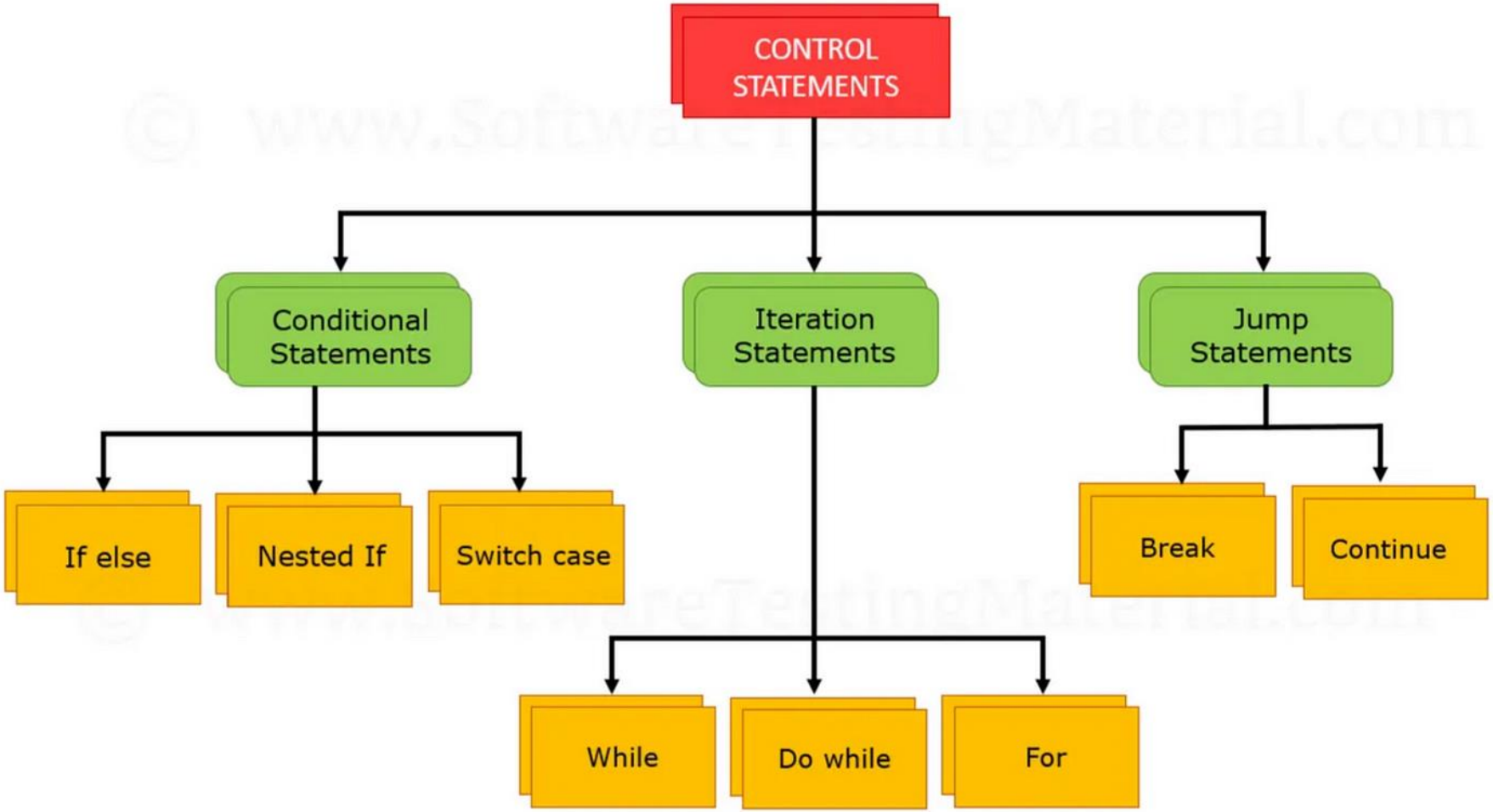
- JSON (JavaScript Object Notation) : هي صيغة خفيفة وسهلة الاستخدام لتبادل البيانات تمثل البيانات على هيئة أزواج من المفاتيح والقيم في هيكل هرمي

عبارات التحكم

1) **عبارات التحكم (Control Statements):** هي العبارات التي تتيح المرونة والقوة اللازمة للبرامج لتحقيق العمليات الحسابية وتدفق التنفيذ في لغات البرمجة يمكن تقسيم هذه الآليات إلى نوعين رئيسيين:

- **عبارات الاختيار (Selection Statements) :** تتيح اختيار مسار معين من بين مسارات متعددة
- **عبارات التكرار (Iteration Statements) :** تتيح تنفيذ عبارات أو مجموعة من العبارات بشكل متكرر
- **عبارات القفز (Jump Statements) :** تستخدم لنقل تدفق التحكم في البرنامج إلى موقع معين بدلا من تنفيذ التعليمات بشكل تسلسلي

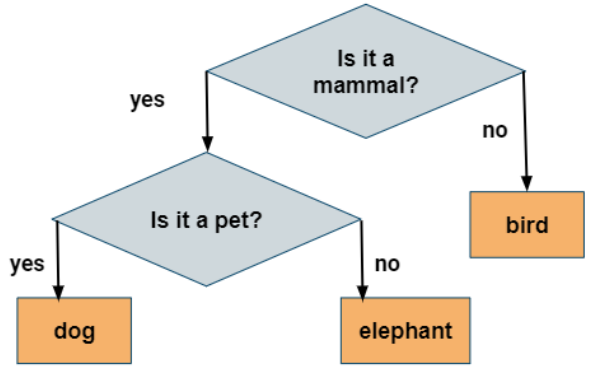
عبارات التحكم



عبارات التحكم

A) **عبارات الاختيار (Selection Statements) :** عبارات الاختيار هي نوع من التعليمات في البرمجة تتيح للمبرمج اختيار طريق واحد من بين مسارين أو أكثر لتنفيذ التعليمات في البرنامج وتعتبر عبارات الاختيار جزءا أساسيا في جميع لغات البرمجة ومنها:

- عبارات الاختيار الثنائية (Two-Way Selection Statements) : عبارات الاختيار الثنائية هي بناء برمجي يستخدم لاتخاذ قرار بين خيارين لتنفيذ أحدهما بناء على تحقق شرط معين

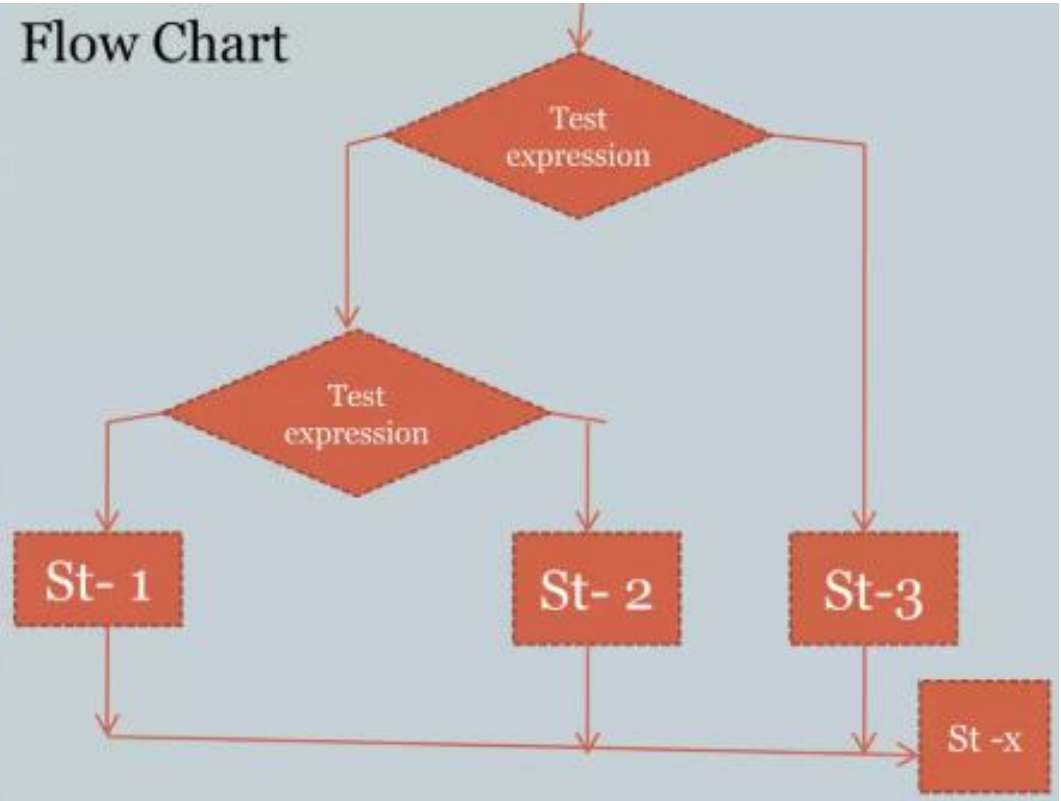


عبارات التحكم

Syntax :-

```
if(test expression1)
{
    if(test expression2)
    {       statement block –1;
    }
    else
    {      statement block – 2;
    }
}
else
{     statement block – 3;
}
Statement – x;
```

Flow Chart

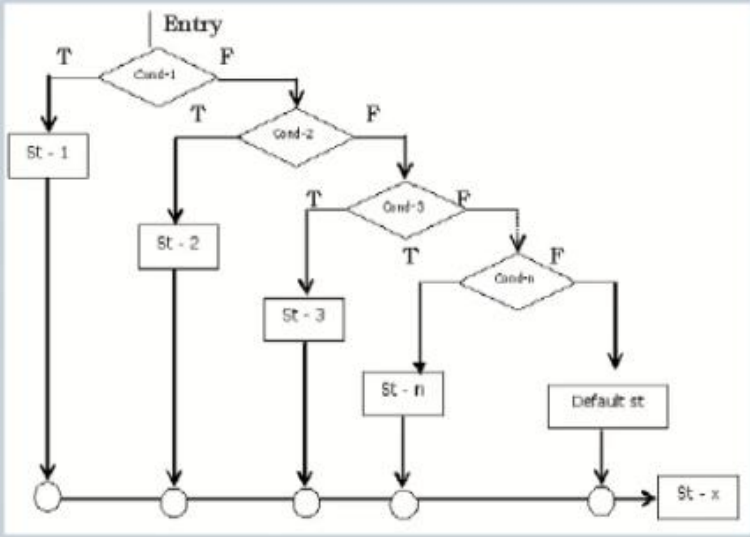


عبارات التحكم

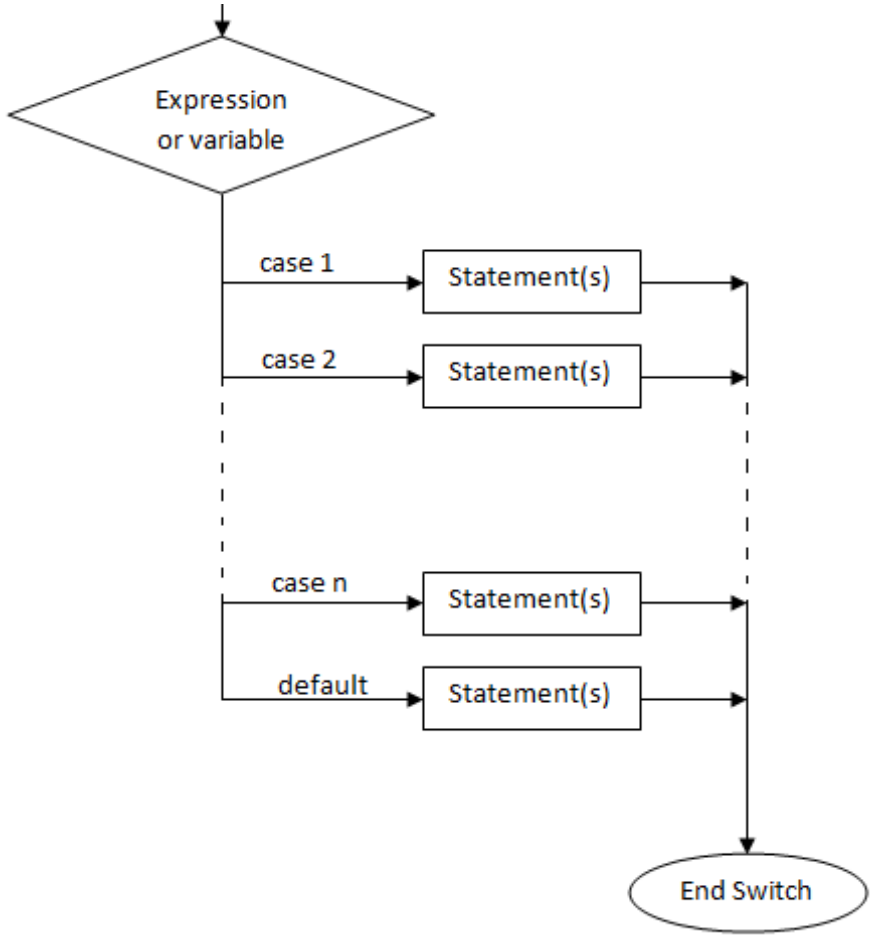
- عبارات الاختيار المتعدد (Multiple-Selection Statements) : عبارات التحديد المتعدد هي أدوات برمجية تسمح باختيار وتنفيذ عدة مسارات أو مجموعات تعليمات بناء على قيمة تعبير معين :

```
Syntax :-
   if (condition1)
         St block–1;
   else  if (condition2)
         St block –2;
            else  if (condition 3)
                  St block –3;
               else
                  St block -4;
    St –x;
```

Flow Chart



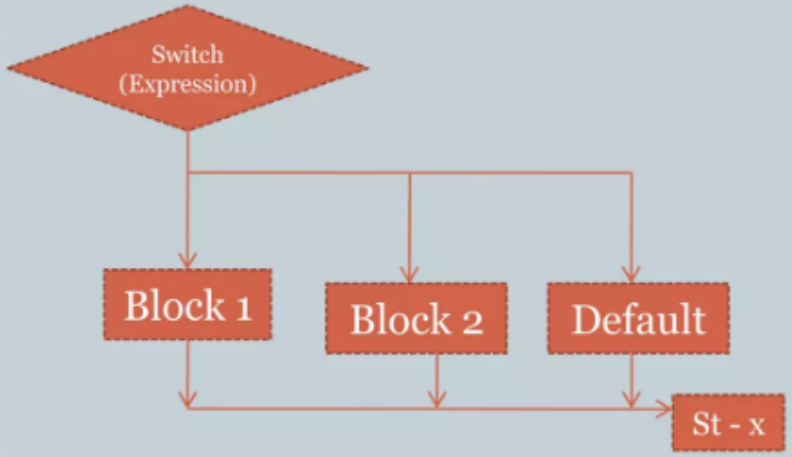
عبارات التحكم



Syntax :-

```
switch (expression)
{
case value1 :    block1;
                 break;
case value 2 :   block 2;
                 break;
default  :       default block;
                 break;
}
st – x;
```

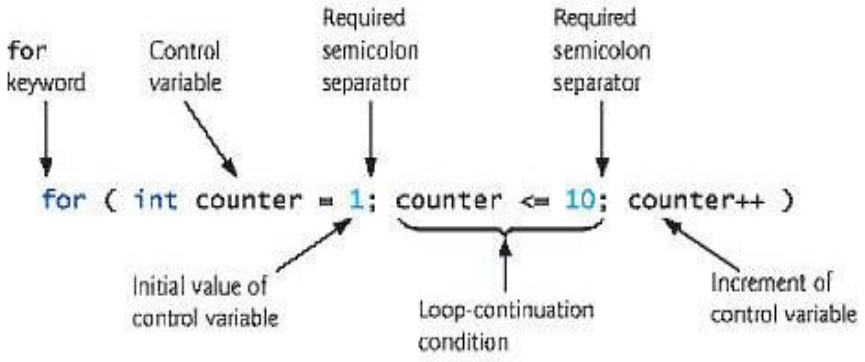
Flow Chart



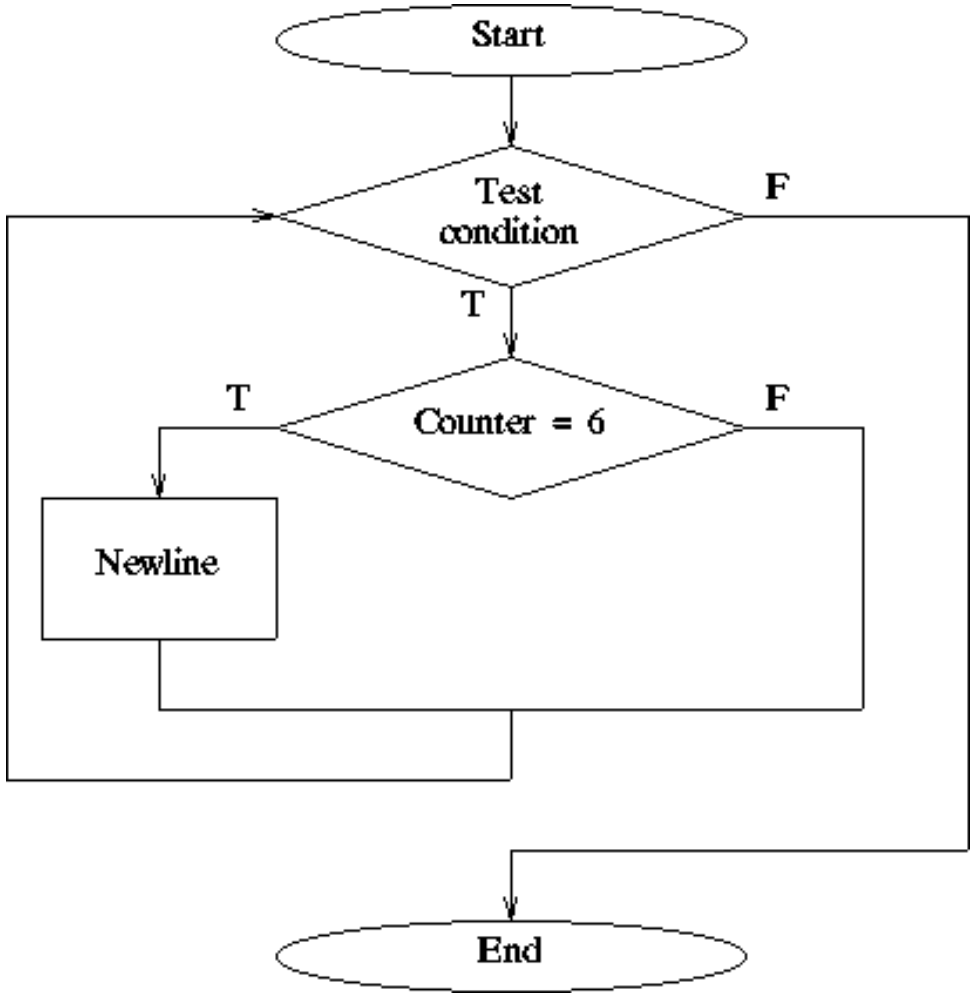
عبارات التحكم

B) عبارات التكرار (Iteration Statements) : عبارات التكرار، أو الحلقات هي تعليمات تتيح تنفيذ مجموعة من التعليمات بشكل متكرر تعتبر هذه العبارات جوهر القوة في البرمجة لأنها تمكن المبرمجين من معالجة كميات كبيرة من البيانات أو تنفيذ عمليات معقدة دون الحاجة إلى تكرار كتابة التعليمات يدويا ومنها :

- عبارات التكرار ذات التحكم العددي **(Counter-Controlled Loops) :** تعرف عبارات التكرار ذات التحكم العددي بأنها تلك التي تعتمد على متغير يسمى متغير الحلقة لضبط عدد مرات التكرار



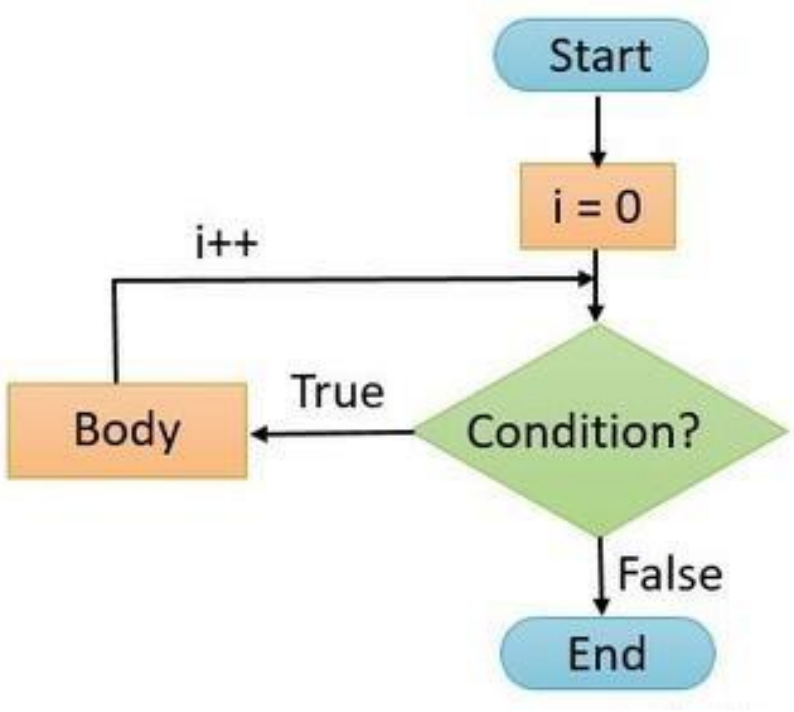
عبارات التحكم



عبارات التحكم

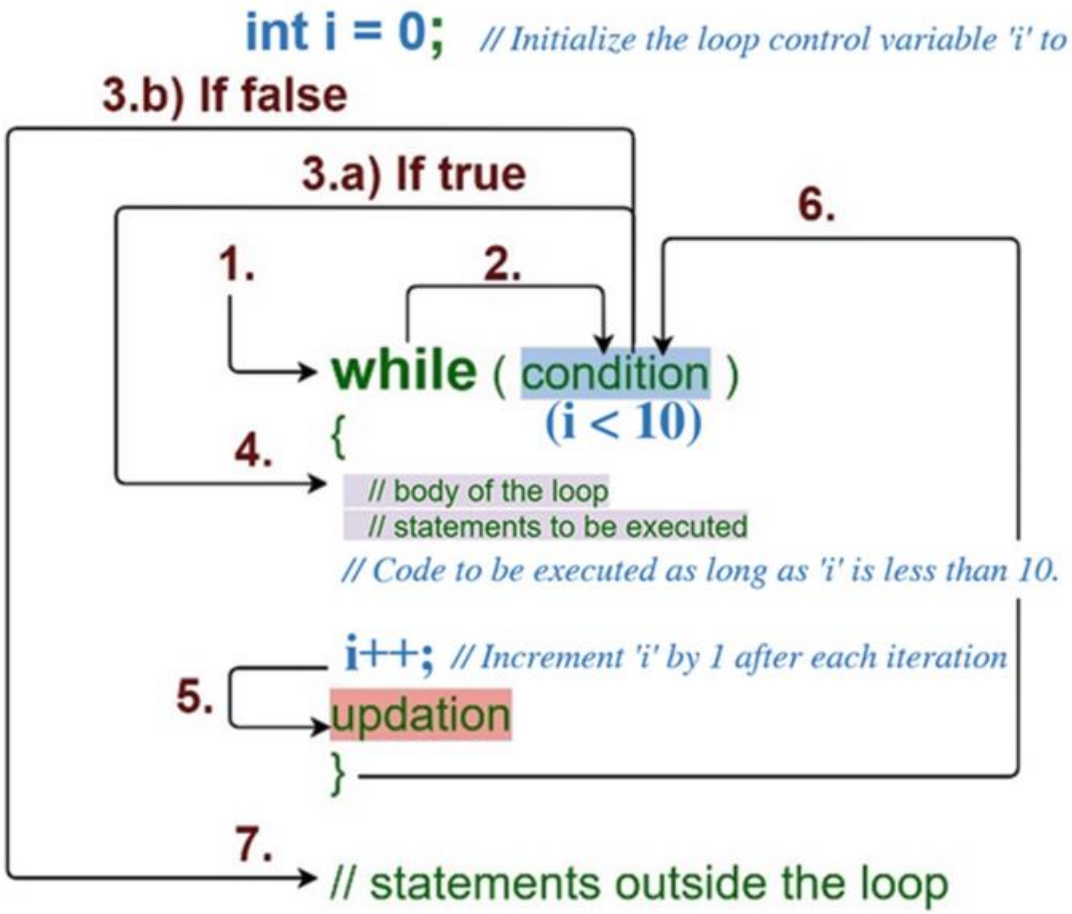
- عبارات التكرار ذات التحكم المنطقي **(Logically -Controlled Loops) :** الحلقات ذات التحكم المنطقي

تستخدم لتكرار مجموعة من العبارات استنادا إلى تعبير منطقي يتم تنفيذ الحلقة طالما أن التعبير المنطقي صحيح



عبارات التحكم

WHILE loop



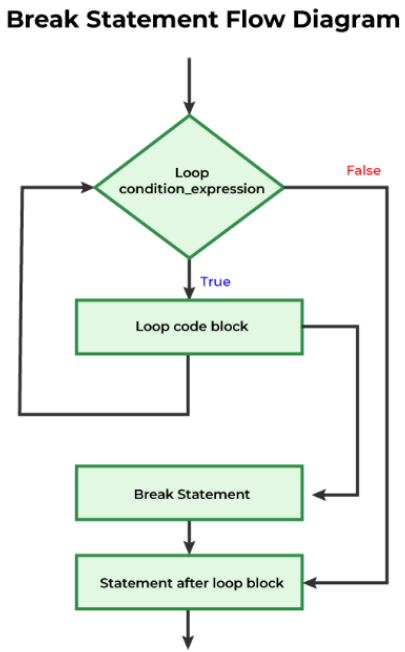
عبارات التحكم

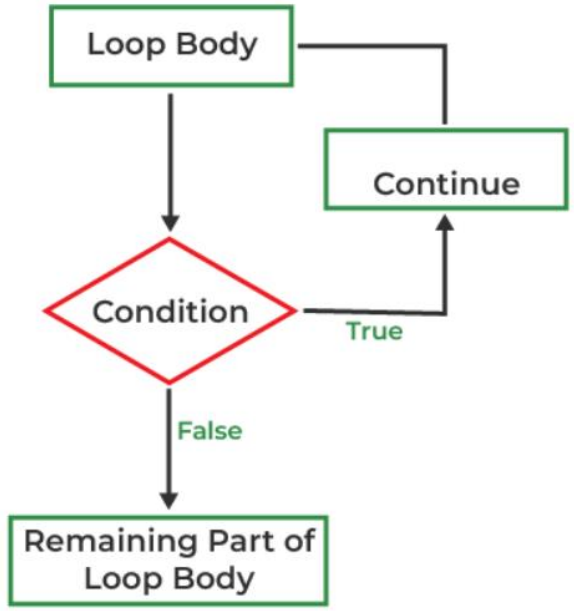
- عبارات التحكم في الحلقات التي يحددها المستخدم **(User-Located Loop Control Mechanisms) :** تتيح للمبرمج تحديد موقع الخروج أو التحكم في الحلقة بدلا من أن يكون التحكم محصورا في البداية أو النهاية فقط تتمثل الفائدة الرئيسية لهذه الآليات في مرونتها حيث يمكن الخروج من الحلقة بناء على شروط معينة في أي نقطة داخل جسم الحلقة

```
while sum < 1000:
value = get_next()
if value < 0:
continue # تخطي القيم السلبية
sum += value
```

عبارات التحكم

C) عبارات القفز (Jump Statements) : تعليمات برمجية تستخدم لنقل تدفق التحكم من نقطة معينة في الكود إلى نقطة أخرى تعتبر جزءا من الهياكل التحكمية في البرمجة وتتيح للمبرمجين تجاوز أو تكرار أجزاء معينة من الكود بناء على شروط أو احتياجات محددة





عبارات التحكم

2) الأوامر المحمية (Guarded Commands): الأمر المحمي يتكون من شرط وتعليمات تنفيذ يتم تنفيذ التعليمات فقط إذا كان الشرط محققا يتم استخدام حراسة الشروط للتعامل مع حالات عدم اليقين وتجنب الأخطاء.

```
if   i = 0 -> sum := sum + i
[]   i > j -> sum := sum + j
[]   j > i -> sum := sum + i
fi
```

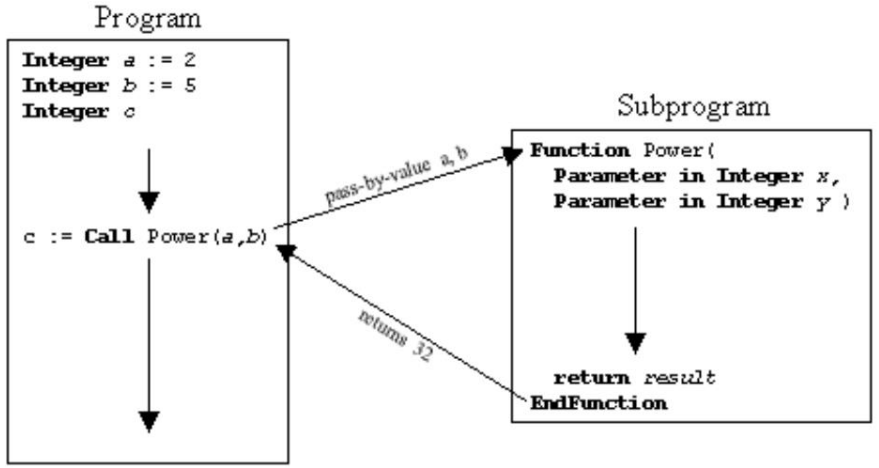
```
if   x >= y -> max := x
[]   y >= x -> max := y
fi
```

```
do <boolean> -> <stmt>
[ ] <boolean> -> <stmt>
    ...
[ ] <boolean> -> <stmt>
od
```

البرامج الفرعية وتنفيذها

1) البرامج الفرعية (Subprograms): هي وحدات برمجية صغيرة تحتوي على مجموعة من التعليمات تؤدي مهمة محددة، وتستدعى من أماكن مختلفة داخل البرنامج لها نوعان اساسيان:

- الإجراءات (Procedures) :لا ترجع قيمة ولكنها تنفذ تعليمات محددة
- الدوال (Functions) : ترجع قيمة وتستخدم غالبا في التعبيرات

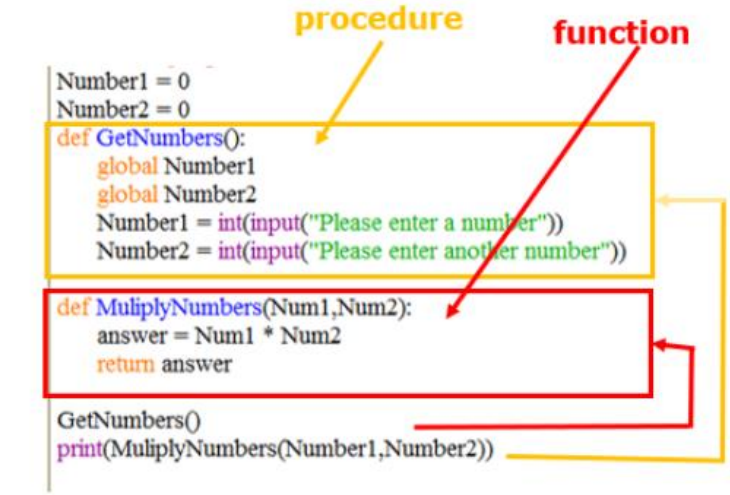


البرامج الفرعية وتنفيذها

```
def get_distance():
    print ("Enter a distance in metres")
    distance = float(input("Enter a value between 1 and 20"))
    while distance<0 or distance >20:
        print ("Invalid value - Try again")
        distance = float(input("Enter a value between 1 and 20"))
    return distance

def display_result(distance):
print ("The distance you entered was", distance)

# main program
distance=get_distance()
display_result(distance)
```

Functions returns a Value

A procedure performs an operation




```
Number1 = 0
Number2 = 0
def GetNumbers():
    global Number1
    global Number2
    Number1 = int(input("Please enter a number"))
    Number2 = int(input("Please enter another number"))
def MultiplyNumbers(Num1,Num2):
    answer = Num1 * Num2
    return answer
GetNumbers()
print(MultiplyNumbers(Number1,Number2))
```

These variables are declared at the top of program

Adding global enables this procedure to access the variables content

MultiplyNumbers() is a function because it returns a value

البرامج الفرعية وتنفيذها

2) تصميم البرامج الفرعية (Design Issues for Subprograms): يتطلب تصميمها مراعاة العديد من القضايا والعوامل منها:

- المتغيرات المحلية : المتغيرات المحلية هي تلك التي تعلن داخل البرنامج الفرعي وتستخدم خلال تنفيذه فقط
  - المتغيرات المحلية الثابت (Static Allocation) :تبقى قيم المتغيرات موجودة في الذاكرة طوال مدة تنفيذ البرنامج ولكن لا يمكن إعادة استخدام الذاكرة بعد انتهاء البرنامج الفرعي
  - المتغيرات المحلية الديناميكي (Dynamic Allocation) : يتم تخصيص المتغيرات في وقت التنفيذ وتحذف القيم من الذاكرة بمجرد انتهاء البرنامج الفرعي ولكنه يستهلك وقت إضافي لإدارة الذاكرة

البرامج الفرعية وتنفيذها

❖ حساسية للتاريخ تعني أن البرنامج الفرعي يمكنه الاحتفاظ بحالة معينة عبر استدعاءات متعددة

من حيث	المتغيرات الثابتة	المتغيرات الديناميكية
الكفاءة	أعلى بسبب عدم الحاجة إلى التخصيص	أقل بسبب التخصيص الديناميكي
دعم التكرار	غير مدعوم	مدعوم
حساسية للتاريخ	مدعوم	غير مدعوم
استخدام الذاكرة	أقل مرونة	أكثر مرونة

```
total = 0

def add_to_total(value):
    global total
    total += value
```

```
int adder(int list[], int listlen) {
    static int sum = 0; // ثابت
    int count;          // ديناميكي
    for (count = 0; count < listlen; count++)
        sum += list[count];
    return sum;
}
```

البرامج الفرعية وتنفيذها

- تعريف البرامج الفرعية المتداخلة : السماح بتعريف برنامج فرعي داخل برنامج فرعي آخر يوفر هذا النهج إمكانية الوصول المباشر إلى المتغيرات المحلية للبرنامج الأب.

```
program MAIN_2;
  var X : integer;
   procedure BIGSUB;
        var A, B, C : integer;
        procedure SUB1;
                var A, D : integer;
                begin { SUB1 }
                A := B + C;  <-----------------1
                end;  { SUB1 }
        procedure SUB2(X : integer);
                var B, E : integer;
                procedure SUB3;
                        var C, E : integer;
                        begin { SUB3 }
                        SUB1;
                        E := B + A:    <-------------2
                        end; { SUB3 }
                begin { SUB2 }
                SUB3;
                A := D + E;  <----------------3
                end; { SUB2 }
        begin { BIGSUB }
        SUB2(7);
        end; { BIGSUB }
  begin
  BIGSUB;
  end. { MAIN_2 }
```

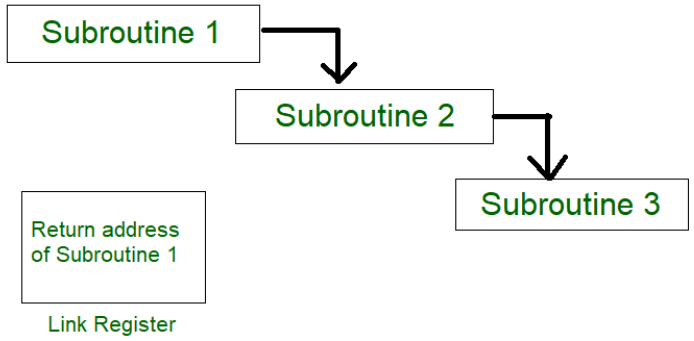
Calling sequence:

Main_2 calls BIGSUB

BIGSUB calls Sub2

Sub2 calls Sub3

Sub3 calls Sub1

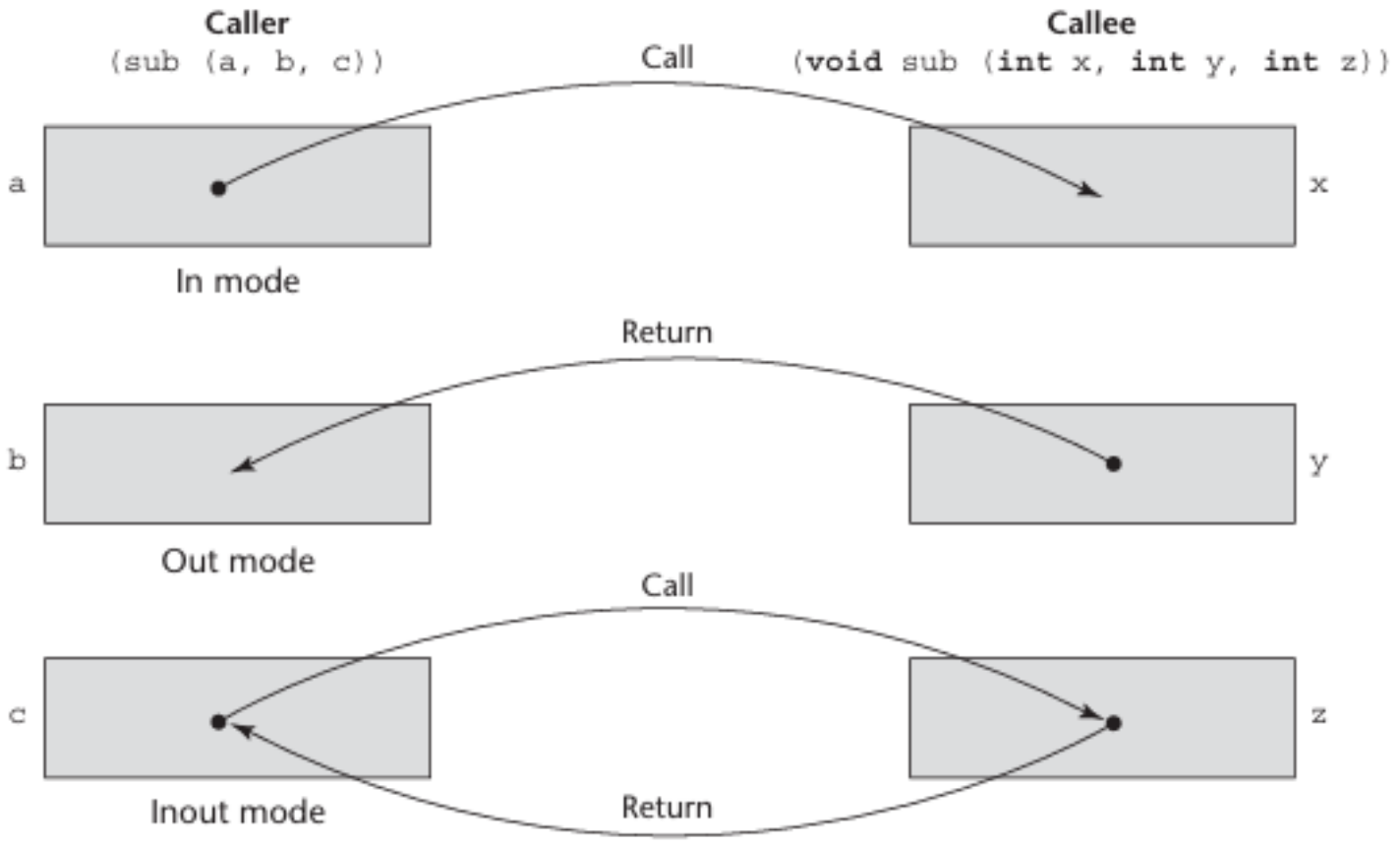


البرامج الفرعية وتنفيذها

- طرق تمرير المعاملات :

- تمرير بالقيمة (Pass-by-Value) : يتم تمرير نسخة من القيمة الأصلية للبرنامج الفرعي تقوم بحماية البيانات الأصلية من التعديل ولكن تستهلك اكثر من للذاكرة عند نسخ القيم الكبيرة
- تمرير بالمرجع (Pass-by-Reference) : يتم تمرير عنوان المتغير بدلا من نسخته وكفاءة في استهلاك الذاكرة ولكن يمكن تعديل البيانات الأصلية
- تمرير بالقيمة المرجعية (Pass-by-Value-Result) : يتم نسخ القيمة الأصلية عند التمرير وإعادة القيمة النهائية بعد التنفيذ
- تمرير بالأسماء (Pass-by-Name) : يتم تمرير التعبير النصي بدلا من القيمة أو المرجع

البرامج الفرعية وتنفيذها



البرامج الفرعية وتنفيذها

Pass-by-Value

```c
// C program to illustrate
// call by value
#include <stdio.h>

void func(int a, int b)
{
    a += b;
    printf("In func, a = %d b = %d\n", a, b);
}
int main(void)
{
    int x = 5, y = 7;

    // Passing parameters
    func(x, y);
    printf("In main, x = %d y = %d\n", x, y);
    return 0;
}
```

```
In func, a = 12 b = 7
In main, x = 5 y = 7
```

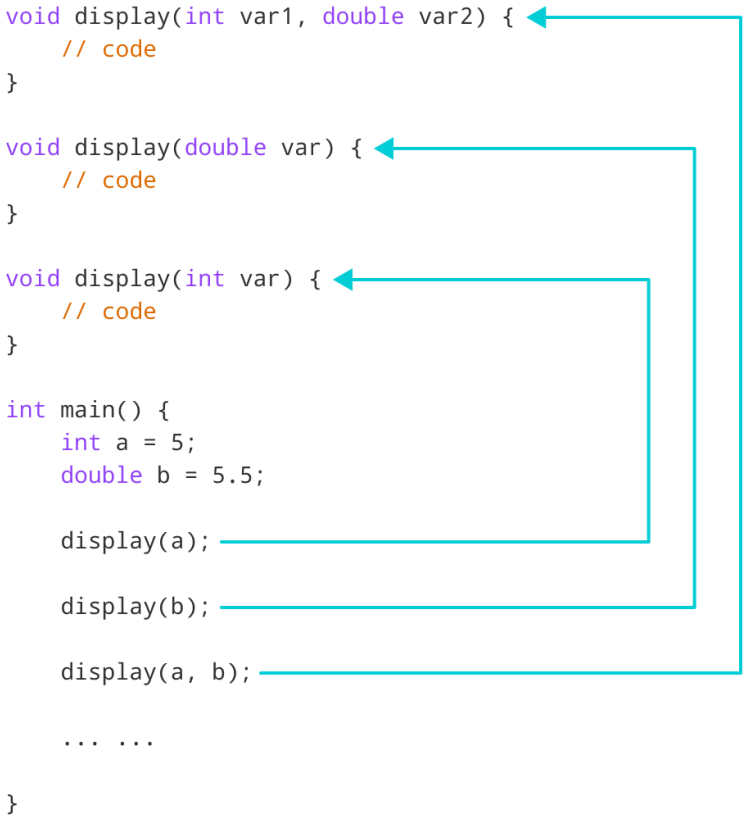
Pass-by-Reference

```c
// C program to demonstrate the pass by pointer in Function

#include <stdio.h>

// Function to modify the value passed as pointer to an int
void modifyVal(int* myptr)
{
    // Access and modifying the value pointed by myptr
    *myptr = *myptr + 5;
}

int main()
{

    int x = 5;
    int* myptr = &x;

    // Passing the pointer ptr to the function
    modifyVal(myptr);

    // printitng the modified value of x
    printf("Modified value of x is: %d\n", x);
    return 0;
}
```

```
Modified value of x is: 10
```

البرامج الفرعية وتنفيذها

- التحميل الزائد (Overloading) : القدرة على تعريف برامج فرعية بنفس الاسم ولكن بمعلومات مختلفة (عدد أو نوع العمليات).



```
void display(int var1, double var2) {
    // code
}

void display(double var) {
    // code
}

void display(int var) {
    // code
}

int main() {
    int a = 5;
    double b = 5.5;

    display(a);

    display(b);

    display(a, b);

    ... ...

}
```

البرامج الفرعية وتنفيذها

3) البرامج الفرعية البسيطة (Simple Subprograms): البرامج التي لا يمكن أن تكون داخل برامج فرعية أخرى **والمتغيرات المحلية ثابتة** ويتطلب تفيذها:

- أثناء استدعاء البرنامج الفرعي (Subprogram Call):
  - حفظ حالة التنفيذ : حفظ المسجلات والمعلومات الضرورية لاستئناف تنفيذ البرنامج
  - تمرير المعاملات : حساب قيم المعاملات الفعلية وتمريرها إلى المعاملات الشكلية للبرنامج الفرعي
  - تمرير عنوان الإرجاع : يتم حفظ العنوان الذي يجب أن يعود إليه البرنامج الفرعي بعد الانتهاء
  - تحويل التحكم : يتم تحويل التحكم إلى البرنامج الفرعي الذي تم استدعاؤه

البرامج الفرعية وتنفيذها

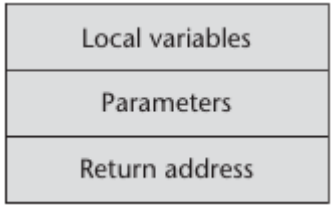
- أثناء إنهاء البرنامج الفرعي (Subprogram Return):
  - إرجاع القيم : إذا كانت هناك عمليات يتم نقل القيم المحسوبة إلى المعاملات الفعلية
  - إرجاع قيمة الدالة: إذا كان البرنامج الفرعي دالة، يتم نقل القيمة المحسوبة إلى منطقة يمكن للبرنامج المستدعي الوصول إليها
  - استعادة حالة التنفيذ : إعادة الحالة المحفوظة للبرنامج المستدعي
  - ارجاع التحكم : يتم إعادة التحكم إلى البرنامج المستدعي عند النقطة التي توقف عندها

البرامج الفرعية وتنفيذها

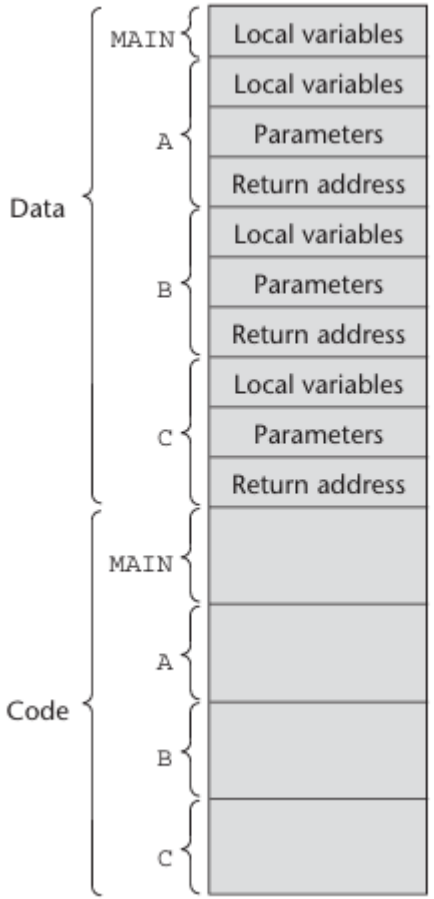
- سجل التنشيط (Activation Record): يمثل البنية التخزينية التي تحتوي على جميع المعلومات الضرورية لتنفيذ البرنامج الفرعي يتكون من :
  - المتغيرات المحلية (Local variables)
  - المعاملات (Parameters)
  - عنوان الإرجاع (Return address)

❖ ثبات السجل (Static Allocation): نظرا لأن البرامج الفرعية البسيطة لا تدعم التكرار يتم تخصيص السجل مرة واحدة ويمكن ربطه مباشرة بكود البرنامج الفرعي

البرامج الفرعية وتنفيذها



سجل تنشيط للبرامج الفرعية البسيطة



الكود وسجلات التنشيط
لبرنامج ببرامج فرعية بسيطة

البرامج الفرعية وتنفيذها

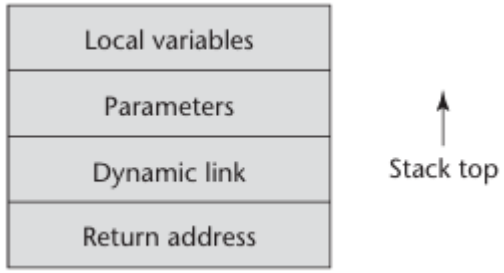
4) البرامج الفرعية مع متغيرات محلية ديناميكية (Subprograms with Stack-Dynamic Local Variables)**:**

عندما تحتوي البرامج الفرعية على **متغيرات محلية ديناميكية** يتم تخصيص ذاكرة لهذه المتغيرات ديناميكيا عند استدعاء البرنامج الفرعي و يتم تحرير هذه الذاكرة عند انتهاء تنفيذ البرنامج الفرعي هذه الطريقة تتيح دعم التكرار مما يعني إمكانية استدعاء نفس البرنامج الفرعي عدة مرات يختلف عن البرامج البسيطة في:

- التخصيص الديناميكي للمتغيرات و التكرار

- هيكل سجل التنشيط (Activation Record) يشمل :
- عنوان الإرجاع (Return Address), المعاملات(Parameters) والمتغيرات المحلية(Local Variables)
- و الرابط الديناميكي (Dynamic Link) :يشير إلى قاعدة سجل التنشيط الخاص بالبرنامج المستدعي

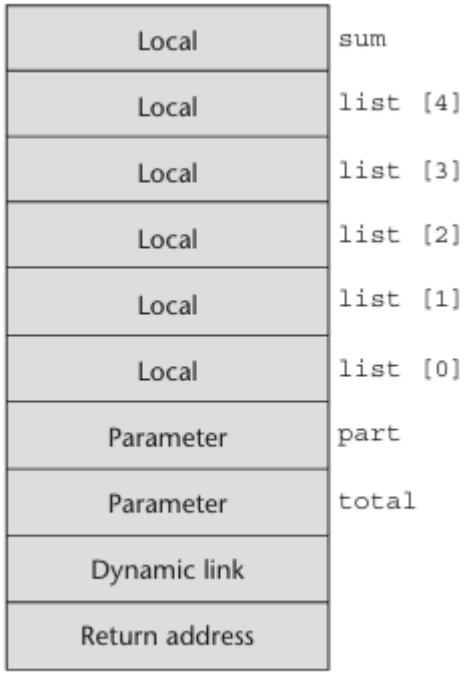
البرامج الفرعية وتنفيذها



سجل تنشيط مكدس للغة ذات متغيرات محلية ديناميكية

```
void sub(float total, int part) {
  int list[5];
  float sum;
  . . .
}
```

الدالة الفرعية



سجل التنشيط للدالة الفرعية

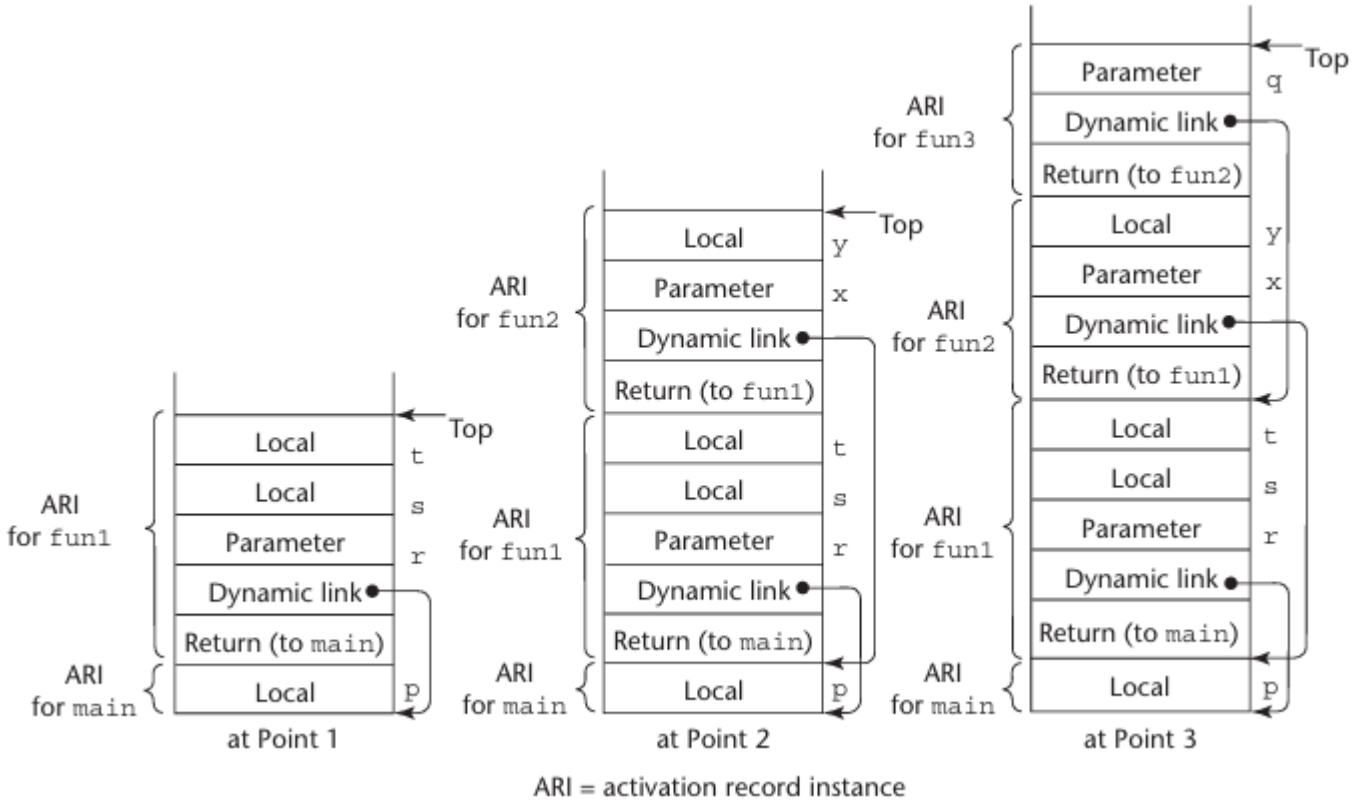
البرامج الفرعية وتنفيذها

Consider the following skeletal C program:

```
void fun1(float r) {
  int s, t;
  ...      <-------- 1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <-------- 2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <-------- 3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```

The sequence of function calls in this program is

```
main calls fun1
fun1 calls fun2
fun2 calls fun3
```



تكديس المحتويات لثلاث نقاط في البرنامج

أنواع البيانات المجردة

1) تجريد البيانات (Data Abstraction): هو طريقة لتعريف هيكل بيانات (مثل السجلات) مع تحديد العمليات المرتبطة به مع إخفاء تفاصيل التنفيذ عن باقي أجزاء البرنامج لتقليل من تعقيد البرامج الكبيرة وجعلها أكثر قابلية للإدارة والصيانة و من مكونات النوع المجرد من البيانات :

- هيكل بيانات محدد (Data Structure): تمثيل البيانات
- برامج فرعية (Subprograms) :العمليات التي تعمل على البيانات مثل الإضافة، الحذف، التعديل، والبحث
- تحكم في الوصول (Access Control) : إخفاء التفاصيل غير الضرورية عن المستخدمين الخارجيين

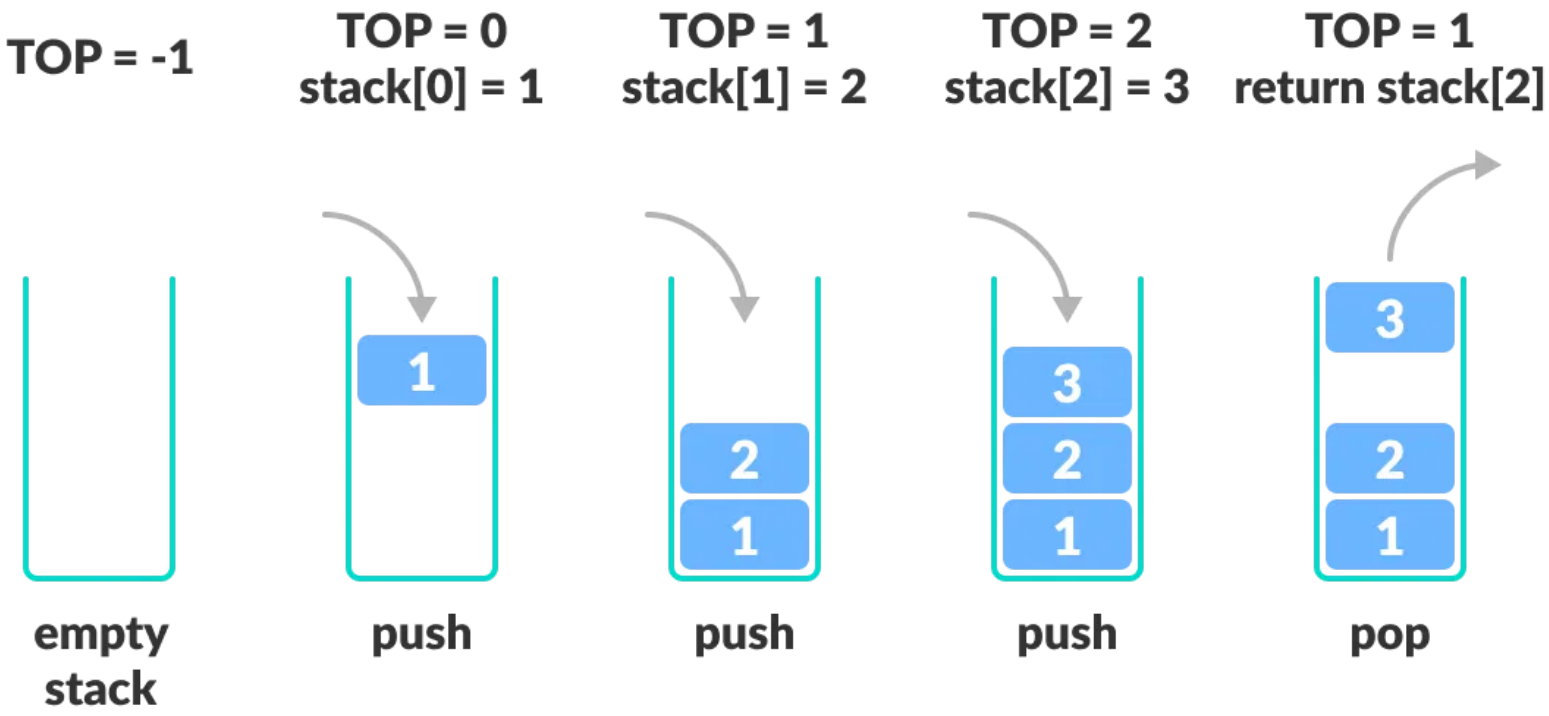
أنواع البيانات المجردة

2) أنواع البيانات المجردة (Abstract Data Types - ADTs)**:** هي نماذج برمجية تستخدم لتمثيل البيانات مع التركيز على الوظائف والعمليات التي يمكن تنفيذها عليها دون الاهتمام بتفاصيل التنفيذ الداخلي :

- المكدس (Stack) : هيكل بيانات يعتمد على مبدأ LIFO (Last In, First Out)؛ العنصر الأخير الذي أُضيف هو الأول الذي يتم حذفه وعملياته الرئيسية هي :
  - Push : إضافة عنصر إلى أعلى المكدس
  - Pop : إزالة العنصر الموجود في أعلى المكدس
  - Peek/Top : إرجاع العنصر الموجود في الأعلى دون إزالته

أنواع البيانات المجردة

```
public class Stack {
  private int top = 0;
  private int[] elements;
  …
  …
  public int pop() {
  1. int element = -1;
  2. if( top>=2 ) {
  3.    top = top - 1;
  4.    element = elements[ top ];
  5. } //END if STATEMENT
  6. return element;
  } //END pop() METHOD
  …
  …
} //END Stack CLASS
```

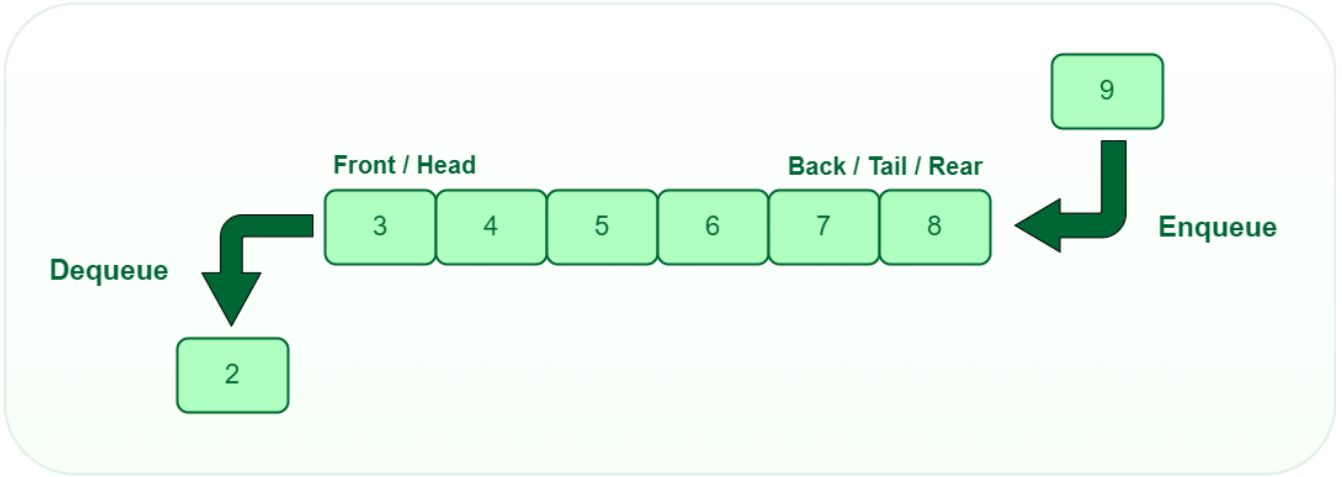


أنواع البيانات المجردة

- الطابور (Queue) : هيكل بيانات يعتمد على مبدأ **FIFO** (First In, First Out)؛ العنصر الأول الذي أُضيف هو الأول الذي يتم حذفه و العمليات الرئيسية هي :
  - Enqueue : إضافة عنصر إلى نهاية الطابور
  - Dequeue : إزالة العنصر الأول في الطابور
  - Front : ارجاع العنصر الموجود في بداية الطابور دون إزالته

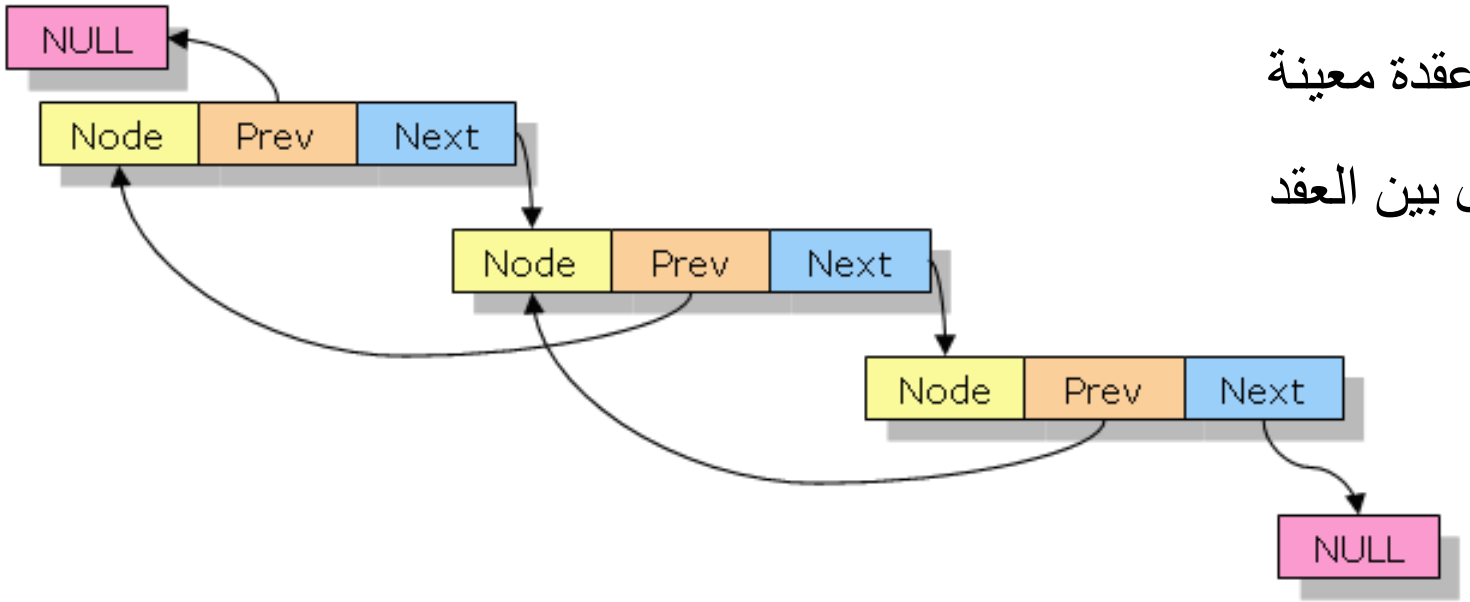
أنواع البيانات المجردة

```
5
6   void Dequeue() {
7       struct Node* temp = front;
8       if(front == NULL) {
9           printf("Queue is Empty\n");
10          return;
11      }
12      if(front == rear) {
13          front = rear = NULL;
14      }
15      else {
16          front = front->next;
17      }
18      free(temp);
19  }
20
```



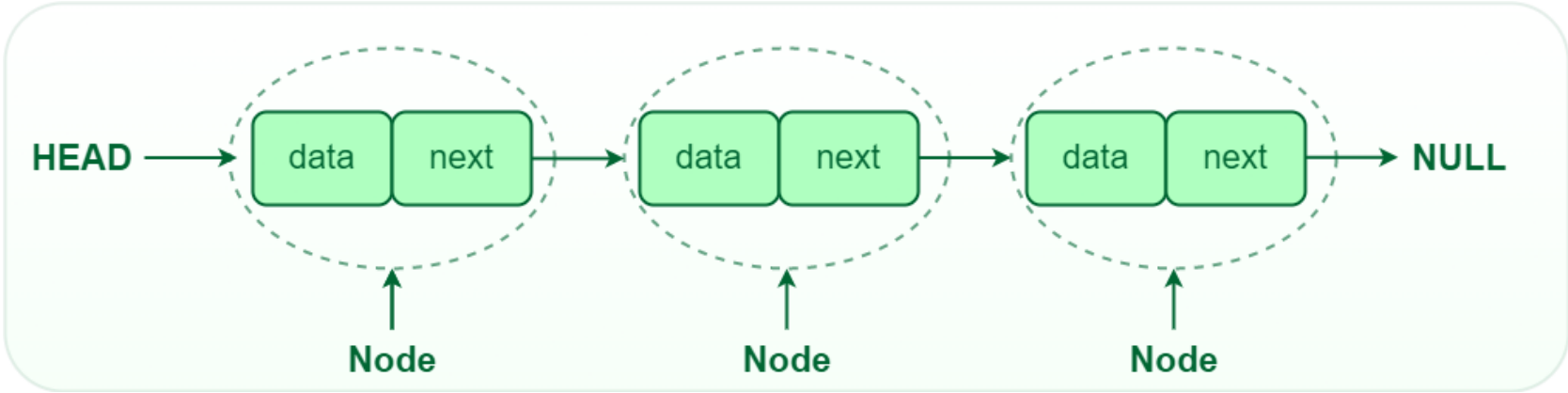
أنواع البيانات المجردة

- القائمة المرتبطة (Linked List) : مجموعة من العقد (Nodes) حيث تحتوي كل عقدة على قيمة ورابط يشير إلى العقدة التالية يمكن أن تكون أحادية الاتجاه أو ثنائية الاتجاه و العمليات الرئيسية هي :
  - Insert : إدخال عقدة في مكان معين
  - Delete : حذف عقدة معينة
  - Traverse : التنقل بين العقد



أنواع البيانات المجردة

```c
//Required Libraries
#include <stdio.h>
#include <stdlib.h>

//Defining structure of a Node
typedef struct Node{
  int data;
  struct Node *next;
}Node;

//Defining structure of a List
typedef struct List{
  int size;
  Node *head;
}List;

//Initilizes head and size of list
void initList(List *l){
  l->head = NULL;
  l->size = 0;
}

//Determines if list is empty (no nodes in list)
int isEmpty(List *l){
  return (l->head == NULL);
}
```

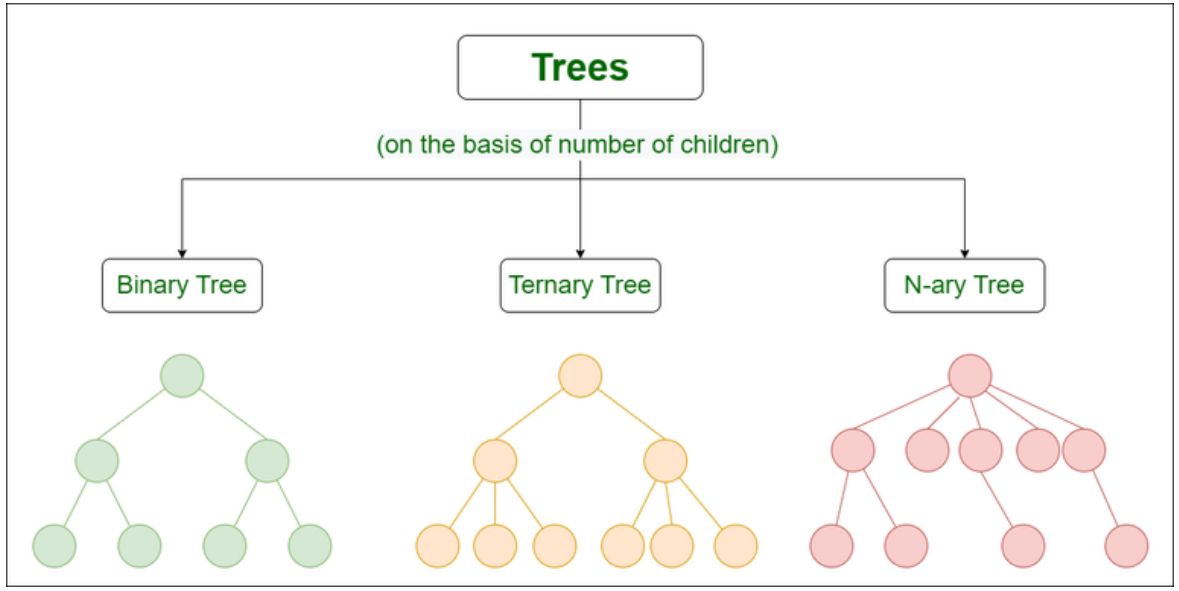


أنواع البيانات المجردة

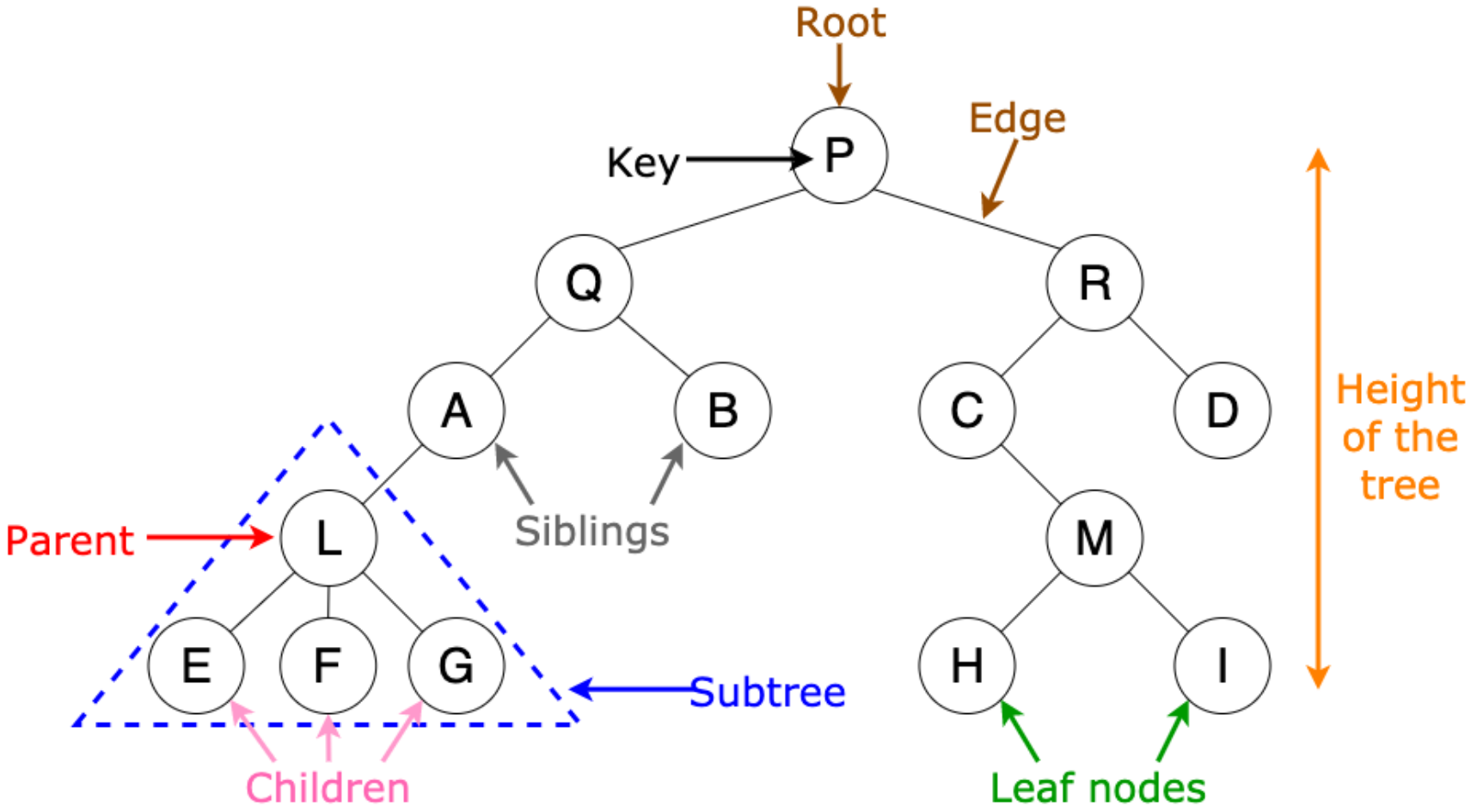
- الشجرة (Tree) : هيكل بيانات هرمي يتكون من عقد تحتوي كل عقدة على قيمة وروابط إلى العقد الفرعية

من انواعها شجرة ثنائية, شجرة البحث الثنائية و شجرة AVL و العمليات الرئيسية هي :

- Insert : إدخال عقدة في مكان معين
- Delete : حذف عقدة معينة
- Traverse : التنقل بين العقد
- Search : البحث عن قيمة

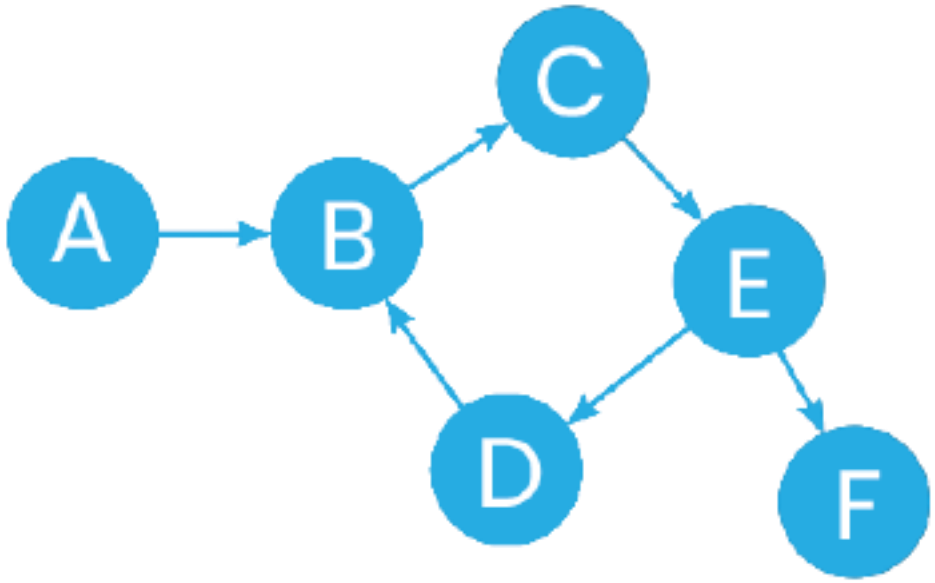


أنواع البيانات المجردة

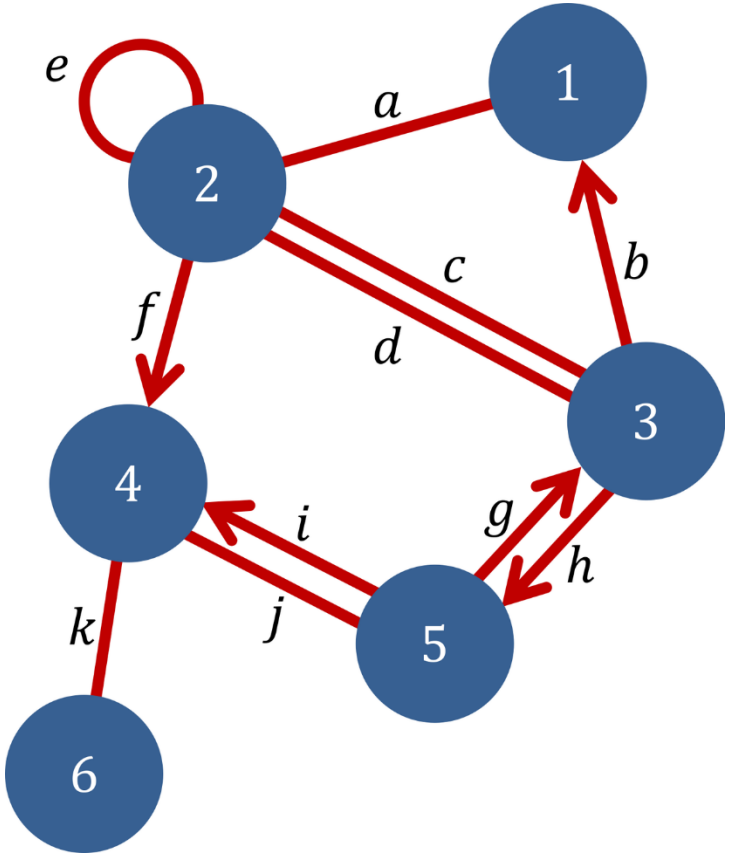


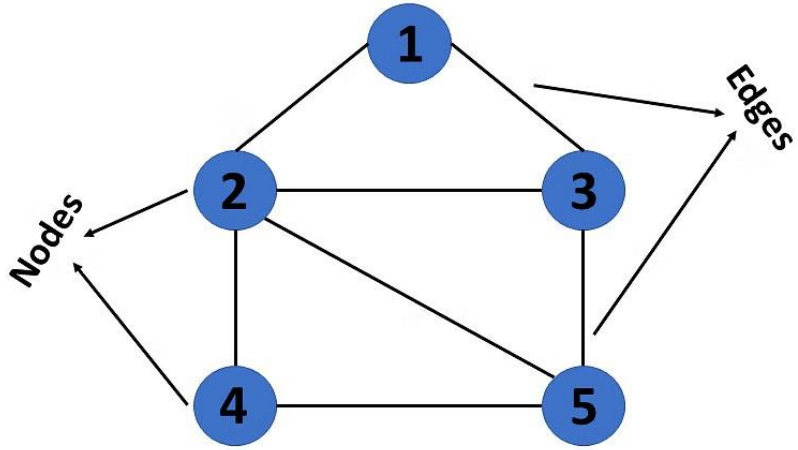
أنواع البيانات المجردة

- الرسوم البيانية (Graph) : هيكل بيانات يتكون من عقد وروابط بينها يمكن أن يكون الرسم البياني موجها أو غير موجه و العمليات الرئيسية هي :
  - AddNode : اضافة عقدة
  - AddEdge :اضافة رابط
  - Traversal : التنقل بين العقد



أنواع البيانات المجردة

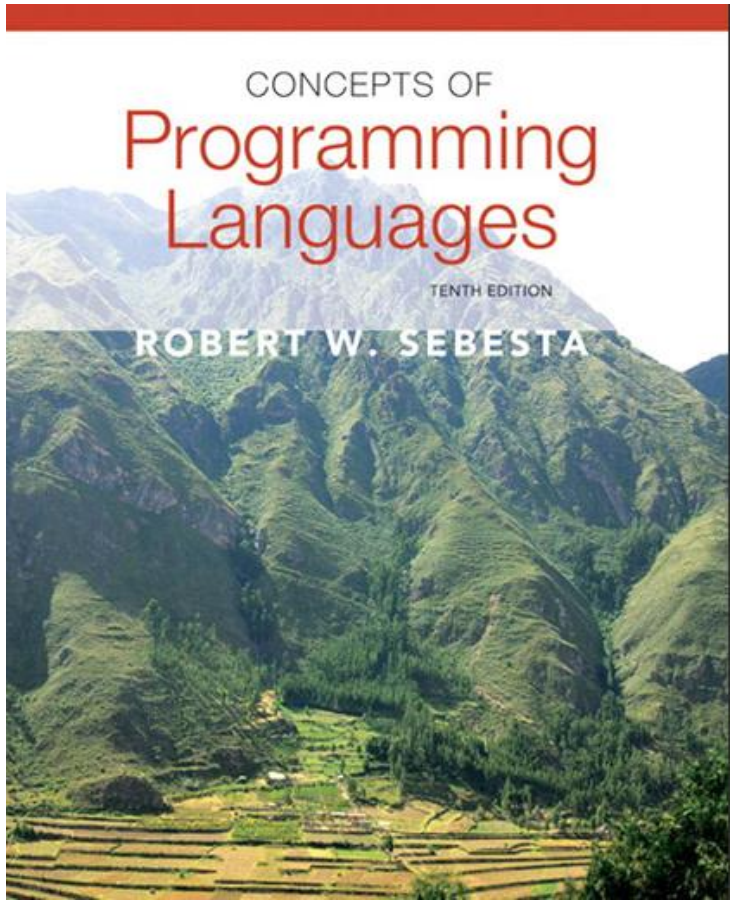


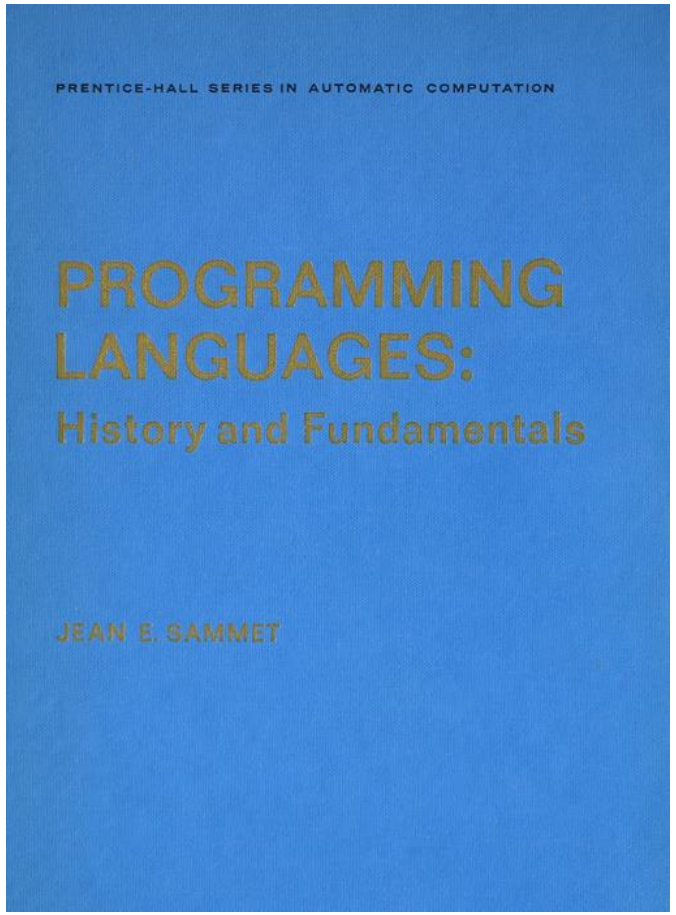


روابط خارجية

عنوان الفيديو	الرابط

المراجع





- Concepts Of Programing Language
- Programing Language History and Fundamentals

شكرا لكم