كلية العلوم

القسم : **الرياضيات**

السنة : **الثانية**

المادة : لغات البرمجة٢

المحاضرة : السادسة/عملي/

{{ مكتبة A to Z }}

Facebook Group : مكتبة A to Z

الجمهورية العربية السورية
جامعة طرطوس
كلية العلوم قسم الرياضيات
السنة الثانية
المادة:لغات برمجة٢_عملي

المحاضرة السادسة

العودية

Recursion

2025-2026

مفهوم العودية(Recursion):

هي أسلوب في البرمجة حيث تستدعي الدالة نفسها لحل جزء من المشكلة حتى الوصول إلى حالة بسيطة تعرف باسم حالة القاعدة (Base Case) التي تنهي الاستدعاء.

مراحل عمل العودية:

A. تجزئة المشكلة: في البداية يتم تحويل المشكلة إلى مشكلة أصغر من نفس النوع.
B. الاستدعاء الذاتي: تستدعي الدالة نفسها لمعالجة النسخة الأصغر من المشكلة. هذا الاستدعاء يجب أن يكون متجهاً نحو الحل، أي يقلل حجم العمل في كل مرة.
C. الوصول إلى حالة القاعدة: وهي المرحلة التي تتوقف عندها الاستدعاءات. حالة القاعدة تمثل أبسط شكل للمشكلة، ويمكن حلها مباشرة بدون أي استدعاء إضافي. وجودها ضروري لمنع التكرار اللانهائي.
D. إعادة النتائج: بعد الوصول إلى حالة القاعدة، تبدأ الدالة بإرجاع النتائج خطوة بخطوة إلى الاستدعاءات السابقة. في هذه المرحلة يبنى الحل النهائي من خلال تجميع نتائج الاستدعاءات.
E. الحل الكامل: عند عودة آخر استدعاء، تكون المشكلة قد حُلّت بالكامل، ويظهر الناتج النهائي الذي اعتمد على سلسلة الاستدعاءات السابقة.

الفرق بين العودية والحلقات التكرارية:

يمكن توضيح الفروقات بين العودية والحلقات التكرارية من خلال النقاط التالية:

- طريقة العمل: العودية تعتمد على استدعاء الدالة نفسها، بينما الحلقات تعتمد على تكرار أوامر معينة ضمن حدود محدودة.
- وضوح الحل: العودية تكون أوضح في المشكلات القابلة للتقسيم (مثل العاملي) ، بينما الحلقات تكون أوضح في التكرارت البسيطة.
- استهلاك الذاكرة: العودية تستهلك مكدس الذاكرة (stack) بسبب الاستدعاءات، بينما الحلقات أخف وأوفر.

- **إنهاء التنفيذ: العودية تنتهي عند الوصول إلى حالة القاعدة، بينما الحلقات تنتهي عند تحقق شرط الإيقاف.**
- **الاستخدام الأنسب: نستحدم العودية للمشكلات التي يمكن تقسيمها إلى أجزاء أصغر، ونستخدم الحلقات للتكرارات المباشرة والواضحة.**

أمثلة على التوابع باستخدام العودية:

مثال 1: اكتب تابع يطبع عبارة نصية n مرة بشكل عودي:

هذا المثال يوضح مبدأ العودية بطريقة سهلة، حيث يقوم التابع print بطباعة كلمة hello عدة مرات اعتماداً على قيمة n. في كل مرة يستدعي التابع نفسه مع تقليل العدد n بمقدار 1 حتى الوصول إلى شرط التوقف. العملية تتوقف تلقائياً عندما يصبح n أصغر من 1 وذلك لمنع الاستدعاءات اللانهائية.

```
1  #include<iostream.h>
2  void print (int n){
3   if (n<1)
4   return;
5   else
6   cout <<"hello"<<endl;
7   print(n-1);
8     }
9   int main(){
10     int n;
11     cout<<"Enter a number : ";
12     cin>>n;
13     print(n);
14    }
```

شرط التوقف:n<1 أي لا يوجد عدد مرات طباعة

استدعاء ذاتي للتابع مع تقليل n بمقدار 1

الخرج:

عند تمرير القيمة 3 الى التابع سيتم التحقق من شرط التوقف فسيكون غير محقق ، وبالتالي سيطبع كلمة hello ثم نعيد استدعاء التابع بقيمة جديدة للوسيط print(3-1)، ثم تعاد نفس العملية التحقق من شرط التوقف ومن ثم طباعة الكلمة واستدعاء التابع حتى الوصول إلى نقطة التوقف وعندها نتوقف عن الطباعة.

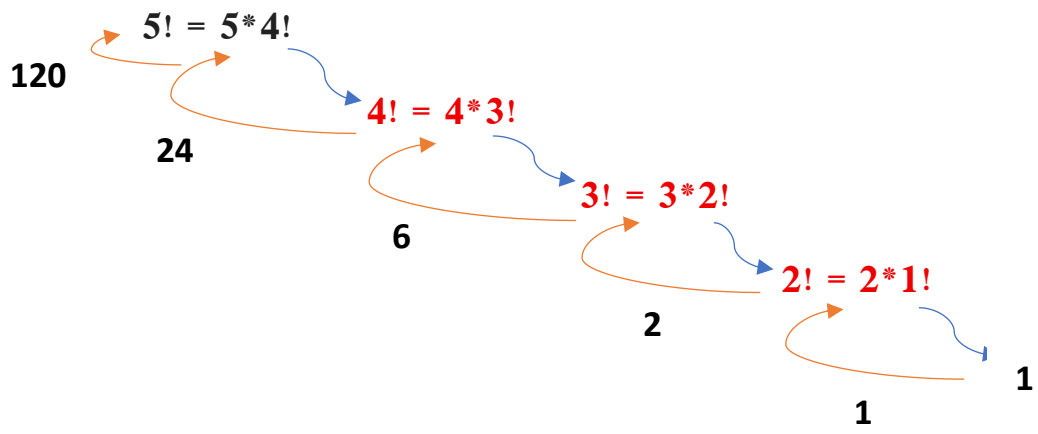
```
Enter a number : 3
hello
hello
hello

[Program finished]
```

مثال 2: تابع حساب العاملي لعدد n بشكل عودي:

في هذا المثال التابع factorial يستخدم مبدأ الاستدعاء الذاتي لحساب حاصل ضرب جميع الأعداد من 1 حتى n ، حيث كل رقم n يتم ضربه في نتيجة العاملي للعدد الذي يسبقه (n−1) ، وتتوقف العملية عند الوصول إلى 0 أو 1 حيث 1=!1=1 0!=1 والتي تمثل حالة القاعدة.

توضيح كيفية الحل:

n=5



```
1  #include <iostream.h>
2  int factorial(int n) {
3     if (n == 0 || n == 1)
4        return 1;
5     else
6        return n * factorial(n - 1);
7  }
8  int main() {
9     int num;
10    cout << "Enter a number : ";
11    cin >> num;
12    cout << factorial(num) << endl;
13 }
```

شرط التوقف:
0!=1!=1

ضرب n مع
استدعاء تابع
العاملي للرقم
السابق له

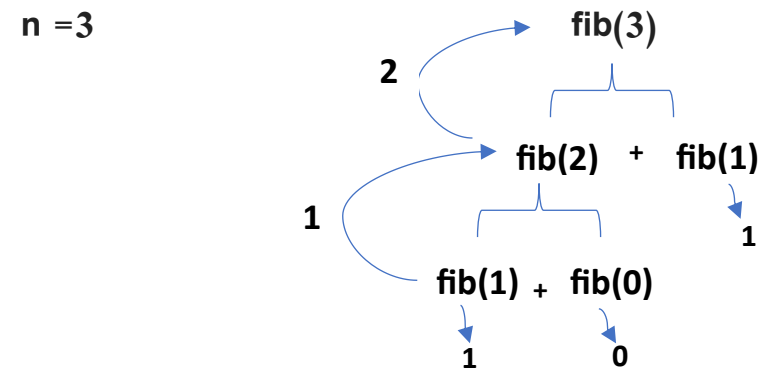
الخرج:

```
Enter a number : 5
120

[Program finished]
```

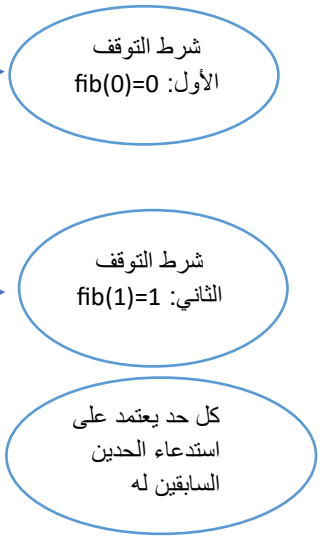
مثال 3: تابع حساب الحد النوني من سلسلة فيبوناتشي بشكل عودي:

في هذا المثال التابع fib يستخدم مبدأ الاستدعاء الذاتي لحساب الحد رقم n في السلسلة، حيث كل حد هو مجموع الحدين السابقين له، وتتوقف العملية عند الحدين الأولين 0 و 1 حيث fib(0)=0 و fib(1)=1.

توضيح كيفية الحل:



```
1  #include <iostream.h>
2  int fib(int n) {
3     if (n == 0) return 0;
4     if (n == 1) return 1;
5     else
6     return fib(n - 1) + fib(n - 2);
7  }
8
9  int main() {
10    int n1;
11    cout<<"Enter a number : ";
12    cin >> n1;
13    cout <<fib(n1) << endl;
14 }
```



الخرج:

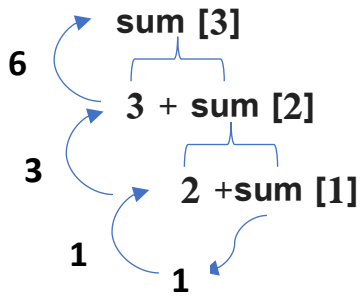
```
Enter a number : 6
8

[Program finished]
```

مثال 4: تابع حساب مجموع الأعداد من 1 إلى n بشكل عودي:

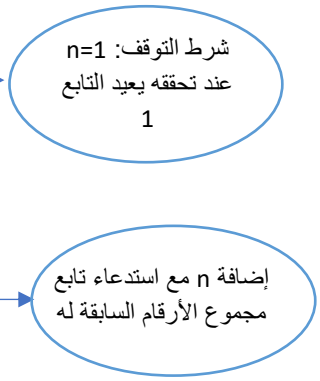
في هذا المثال التابع sum يستخدم مبدأ الاستدعاء الذاتي لحساب مجموع جميع الاعداد من 1 حتى n، حيث نجمع n مع نتيجة مجموع الأرقام السابقة له(n−1)، وتتوقف العملية عندما n يساوي 1 .

توضيح كيفية الحل:

n =3



```
1  #include <iostream.h>
2  int sum(int n) {
3     if (n == 1) return 1;
4     return n + sum(n - 1);
5  }
6
7  int main() {
8     int a;
9     cout << "Enter a number : ";
10    cin >> a;
11    cout << sum(a) << endl;
12 }
```



الخرج:

```
Enter a number : 5
15

[Program finished]
```

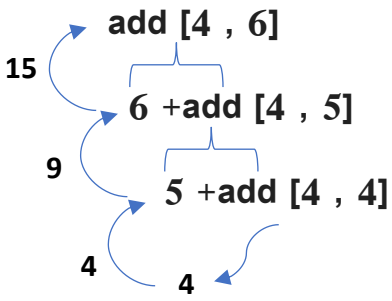
مثال 5: تابع حساب مجموع الأعداد الصحيحة ضمن مجال معين من x إلى y بشكل عودي:

الحالة الأولى: تبدأ العودية من y وتتوقف عند x (من y إلى x) :

في هذه الحالة التابع add يحسب مجموع الأعداد بين x وy باستخدام مبدأ الاستدعاء الذاتي من الحد الأعلى إلى الحد الأدنى، حيث كل رقم يضاف إلى مجموع الأرقام الأقل منه حتى الوصول إلى الحد الأدنى x ، وتتوقف العملية عند تساوي الحدين أي عندما x=y .

توضيح كيفية الحل:

x=4 y=6




```
1  #include <iostream.h>
2  int add(int x, int y) {
3     if (x ==y)
4     return x;
5     else
6     return y + add(x,y-1);
7  }
8  int main() {
9     int x, y;
10    cout << "Enter the first number : ";
11    cin >> x;
12    cout << "Enter the second number : ";
13    cin >> y;
14    cout << add(x,y)<<endl;
15 }
```

قمنا بتمرير قيمة طرفي المجال كوسيطين للتابع

شرط التوقف: عند تساوي قيمة طرفي المجال.

الخرج:

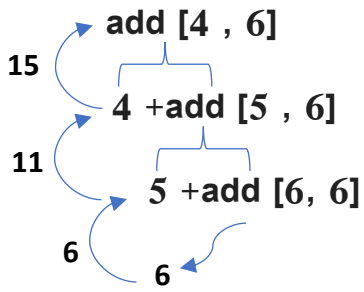
```
Enter the first number : 4
Enter the second number : 6
15

[Program finished]
```

الحالة الثانية: تبدأ العودية من x وتتوقف عند y (من x إلى y) :

في هذه الحالة التابع add يحسب مجموع الأعداد بين x وy باستخدام مبدأ الاستدعاء الذاتي من الحد الأدنى إلى الحد الأعلى، حيث كل رقم يضاف إلى مجموع الأرقام الأعلى منه حتى الوصول إلى الحد الأعلى y، وتتوقف العملية عند تساوي الحدين أي عندما x=y .

توضيح كيفية الحل:

x=4 y=6




```
1  #include <iostream.h>
2  int add(int x, int y) {
3     if (x ==y)
4     return y;
5     else
6     return x + add(x+1,y);
7  }
8  int main() {
9     int x, y;
10    cout << "Enter the first number : ";
11    cin >> x;
12    cout << "Enter the second number : ";
13    cin >> y;
14    cout << add(x,y)<<endl;
15 }
16
```

الخرج:

```
Enter the first number : 4
Enter the second number : 6
15

[Program finished]
```

وظيفة: اكتب تابع يقوم بحساب مجموع الأعداد الفردية فقط ضمن المجال من x ل n بشكل عودي.