

بسم الله الرحمن الرحيم

درس استخدام التقريبات في حساب الدوال.

من إعداد محمد أمين طق.

في هذا الدرس سنتطرق إلى طريقة البرمجة واستخدام الدوال التقريبات و الجودة.

التقريبات مهمة جداً في عمليات الحساب المنطقي، فبرمجة دالة الجذر $\sqrt{1+x}$ فيمكن استخدام عدة طرق فمنها ما يدرس في الجامعات مثل مقياس التحليل العددي لحساب قيمة جذر x ولكن هنا سنطبق الطرق العملية في البرمجة أي نعتمد على سرعة البرمجة.

في هذا العدد سأنتقل إلى طريقة حساب لو غارتم باستخدام الطرق العملية فقط في البرمجة.

مثال الدالة اللوغاريتمية $\log(x)$:

أول شيء نطلب من المستخدم قيمة المتغير x الذي يريد حسابه هكذا:

```
double x ;  
  
cout<<"Enter the value of x: "<<endl ;  
  
cin>>x;
```

أول ما نتطرق له هو الاحتمالات التي يمكن للمستخدم أن يضعها لحساب $\log(x)$ ، وهي

1 - إذا كان x سالب:

نحن نعلم أن الدالة $\log(x)$ لا تقبل أعداد سالبة لذلك يجب أن نظهر جملة للمستخدم وهي على سبيل المثال:
"خطأ يجب أن يكون x موجب".
ونبرمجها كما يلي:

```
if(x<0)  
cout<<"Error ! " ;
```

2 - الاحتمال الثاني وهو إذا كان x يساوي 0:

نحن نعلم أن $\log(0) = -\text{Infinity}$ لذلك نظهر للمستخدم عبارة وهي:

```
if(x==0)  
cout<<"-Infinity" ;
```

3 - المرحلة الثالثة إذا كان $x \geq 1$

نحن نعلم أن نشر تايلور للدالة $\log(1+x)$ حيث x صغير هو كما يلي

$$\frac{\log(1+x)}{x} = 1 - \frac{x}{2} + \frac{x^2}{3} - \frac{x^3}{4} + \dots + \frac{(-x)^n}{n+1} + \dots$$

لذلك فإنه لبرمجة هذه نستعمل ما يلي:

لدينا أن:

$$\frac{\log(1+x)}{x} = 1 + \sum_{i=1}^n \frac{\text{term}(i)}{i+1}$$

حيث:

$$term(i) = (-x)^i$$

ومنه فإن العلاقة بين الحد $i+1$ ($term(i+1)$) والحد i هي بقسمة الحد $i+1$ على الحد i أي أن:

$$\frac{term(i+1)}{term(i)} = -x$$

الآن نأتي لطريقة الحساب:

نضع مايلي:

$$\frac{\log(1+x)}{x} = sum$$

$$term(i) = term$$

ومنه فإن كلما تغير i فإن $term$ يتغير وكذلك sum يتغير ومنه:

لدينا أن i عدد طبيعي يبدأ من ال 1 ومنه فإننا نعلن عنه أنه متغير صحيح يبدأ من الصفر، هكذا:

```
int i=1 ;
```

ونحن نعلم أن المتغيرين $term$ و sum هما عددان حقيقيان يبدأان من 1 نعلن عنهم هكذا:

```
double term=1, sum=1;
```

وبما أننا لدينا جمع نستعمل حلقة $while$ أو for أو $do while$ حسب الرغبة:

أن سأسعمل الدالة $while$ هكذا:

```
while(condition)
```

```
{
```

```
term=term*(-x) ;
```

```
sum=sum+term/i+1 ;
```

```
i++;
```

```
}
```

الآن كما تلاحظون فإن كلما أخذ i قيمة فإن $term$ تأخذ قيمة و sum يأخذ قيمة هكذا:

كما نعلم أنه عند الدخول إلى $while$ فإن المتغيرات تدخل بقيمها الابتدائية أي هكذا:

ومنه عند بداية الحلقة فإن $i=1$ إذن $term=1*(-x)$ و $sum=1+(-x)/1+1$ و $i=1+1$. نهاية الدورة الأولى للحلقة.

وإذا كان الشرط condition مازال صحيحاً فإنه ندخل إلى الدورة الثانية و المتغيرات هنا تتبدل وتأخذ قيمها من الدورة الأولى أي أننا ندخل إلى الدورة الثانية بالمتغيرات التالية:

$i=2$ و $term=-x$ و $sum=1-x/2$ وما يحدث في الدورة الثانية هو:

$term=(-x)*(-x)$ و $sum=1-x/2+(-x)*(-x)/2+1$ و $i=2+1$

وهكذا في الدور الأولى حصلنا على تقريب ل الدالة اللوغاريتمية هكذا: $sum = 1 - x/2$

بينما في الدورة الثانية حصلنا على تقريب للدالة اللوغاريتمية هكذا: $sum = 1 - \frac{x}{2} + \frac{x^2}{3}$

ولكن متى ستتوقف الدالة while:

الشرط condition

هنا نضع الشرط وهو المهم، كما يلي:

إذا كان آخر حد صغير جداً فيا أيتها الدالة while توقفي وأعطيني قيمة المجموع.

لقد وضعنا الحدود على الشكل $term/(i+1)$ أي أن شرطنا يصبح هكذا:

إذا كان الحد $term/(i+1)$ أكبر من عدد صغير فلا تتوقفي يا while أي هكذا:

$while(term/(i+1)>epsilon)$

حيث $epsilon$ عدد صغير نعرفه كما يلي على سبيل المثال:

`double epsilon = 0.00001 ;`

أي أن الدالة while لا تتوقف إلى إذا كان آخر حد نحسبه أقل من $epsilon$ ،

ولكن كما لاحظنا أن قيم $term$ تتغير أي أنها تارة موجبة وتارة أخرى سالبة حيث أنه في الدورة الأولى قيمتها سالبة $term = -x$ وفي الدورة الثانية قيمتها موجبة $term = x^2$ وبذلك فهذه مشكلة أخرى.

ولكن هناك حل، وهو استخدام دالة القيمة المطلقة.

وهي سهلة جداً وهذه الدالة يمكن تعريفها كما يلي:

`if(x>=0)`

`x=x ;`

`else`

`x=-x ;`

نستخدم هنا تعريف الدوال في البرمجة وهو التعريف التالي:

`type function(type variable1, type variable2,...)`

```
{
consyructions ;
return TheValueOfFunction ;
}
```

ونضعها قبل بداية دالة الواجهة *main()*

أي أن دالة القيمة المطلقة ل عدد معين يمكن كتابتها كما يلي:

```
double absolu(double x)
{
if(x>=0)
x=x ;
else
x=-x ;
return x;
}
```

ومنه فإننا شرطنا يصبح كما يلي: $\text{absolu}(\text{term}/(i+1)) > \text{epsilon}$

ونستخدمه في الدالة *while* هكذا: $\text{while}(\text{absolu}(\text{term}/(i+1)) > \text{epsilon})$
الآن أنهينا الشرط،

ولكن نحن نريد أن نحسب $\log(x)$ وليس $\frac{\log(1+x)}{x}$

الحل هنا سهل وهو تحويل المتغير $1+x$ ب x أي أن:

$$\frac{\log(x+1)}{x} = \text{sum}(x) \Rightarrow \frac{\log(x)}{x-1} = \text{sum}(x-1)$$

أي أنه كلما نجد x نستبدله ب $x-1$ ومنه:

$$\log(x) = (x-1) \cdot \text{sum}(x-1)$$

ويصبح برنامجنا مؤقتاً هكذا:

```
#include <iostream>

using namespace std ;
```

```

double absolu(double x)
{
    if(x>=0)
    x=x ;
    else
    x=-x ;
    return x;
}

int main()
{
    double x;
    cout<<"Enter the value of x: "<<endl ;
    cin>>x;

    if(x<0)
        cout<<"Error ! " ;
    if(x==0)
        cout<<"-Infinity" ;

    if(x>=1)
    {
        double epsilon = 0.0001 ;
        double term=1, sum=1;
        int i=1;
        while(absolu(term/(i+1))>epsilon)
        {
            term=term*(1-x) ;
            sum=sum+term/(i+1) ;
            i++;
        }
    }
}

```

```
}
```

```
cout<<"The value of Log(x) is : "<<(x-1)*sum;
```

```
}
```

```
return 0;
```

```
}
```

الآن قم بتنشيط البرنامج.

ماذا ستلاحظ

نعم من أجل إعطاء قيمة 0 أو سالبة أو من أجل $x=1$ البرنامج ناجح.

وغير دقيق من أجل قيم $x=2$ و يا مصيبتاه من أجل $x=3$ أو أكثر فالبرنامج لا يتوقف !!!!

وكذلك أين القيم المحصورة بين 0 و 1 فالبرنامج لا يستجيب أصلاً.

نقولكم بالشوي عليا، إنني نسيت أن أبرمج القيم بين 0 و 1. وكذلك نسيت أن أقل لكم البرنامج صالح فقط من أجل القيم x الأكبر من 1 و الأقل من 2 و 0 و أي عدد سالب !!!!!.

طيب سوف أبرمج القيم المحصورة بين 0 و 1 هكذا:

سأستخدم حالة مهمة في الرياضيات وهي التالية:

$$\log\left(\frac{1}{x}\right) = -\log(x)$$

أي أنه إذا أعطيتني قيمة بين ال 0 و 1 سأقلب هذه القيمة وأضيف إشارة سالب فقط أي هكذا:

وبذلك فتحتم عليا أن أعلن عن دالة أسميها $\ln(x)$ مثل دالة القيمة المطلقة وأضعها قبل بداية البرنامج.

سأقوم بأخذ كل ما يتعلق ب while وأضعه في تصرف الدالة $\ln(x)$ التي أعلنت عنها فقط وأقول للدالة $\ln(x)$ عندما تكمل الحساب أعطيني النتيجة $(x-1)*sum$ أي هكذا:

ويصبح البرنامج هكذا:

```
#include <iostream>
```

```
using namespace std ;
```

```
double absolu(double x)
```

```
{
```

```
if(x>=0)
```

```

x=x ;

else

x=-x ;

return x;

}

double Ln(double x)

{

double epsilon = 0.0001 ;

double term=1, sum=1;

int i=1;

while(absolu(term/(i+1))>epsilon)

{

term=term*(1-x) ;

sum=sum+term/(i+1) ;

i++;

}

return (x-1)*sum ;

}

int main()

{

double x;

cout<<"Enter the value of x: "<<endl ;

cin>>x;

    if(x<0)

        cout<<"Error !" ;

    if(x==0)

        cout<<"-Infinity" ;

```

```

if(x>=1)
cout<<"The value of Log(x) is : "<<Ln(x);
if(x>0 && x<1)
cout<<"The value of Log(x) is : "<<-Ln(1/x);
return 0;
}

```

ولكن البرنامج لا يتوقف عندما أضع $x=3$ فما فوق أو أضع $x=0.22$ فالبرنامج لا يتوقف.

لقد قلت لك أن البرنامج صالح من أجل القيم الصغيرة لـ x أي من أجل:

$$1 \leq x < 2 \text{ ومنه } 0 \leq x-1 < 1$$

لذلك سننتقل إلى المرحلة المهمة وهو كيفية تصغير x حتى يصبح محصور بين القيمتين 1 و 2.

تصغير x

أنا قلت تصغير x أي يرد في ذهنك أنه عند إعطاء قيم كبيرة لـ x فيجب تصغيرها ولكن كيف.

أقول لك ببساطة قسمة x على عدد معين عدد مرات، وبما أننا سنحصر x بين 1 و 2 فما رأيك بقسمة x على 2 عدد مرات.

كلام جميل ولكن كيف،

أقول لك بإدخال دالة while وفي كل مرة أقسم x على 2 أي هكذا:

```

while(condition) {
x=x/2 ;
}

```

والشرط هنا هو القسمة على 2 إلى أن يصبح أقل من 1

أي هكذا:

```

while(x>1) {
x=x/2 ;
}

```

ولكن بعدها كيف نعرف عدد المرات التي قسم بها x ، الحل هنا هو إضافة متغير صحيح j بحيث في كل مرة نقسم على 2 نضيف للقيمة j القيمة 0 أي هكذا:

```

int j=0 ;

```

```
while(x>1) {
x=x/2 ;
j++ ;
}
```

وبهذا كتبنا x على الشكل التالي:

$$x = r \cdot 2^j$$

حيث r أقل من 1.

ومنه في إدخال اللوغارتم على x يصبح كما يلي:

$$\log(x) = \log(r \cdot 2^j) = \log(r) + j \cdot \log(2)$$

حيث r صغير ومحصور بين 0 و 1 ولذلك نستطيع تطبيق عليه نشر تايلور:

ولكن هنا سأعطيه قيمة $\log(2)$ وذلك لتسريع الحساب فقط، ويمكن الحصول عليها باستخدام الطريقة القديمة.

أي سأعطيه قيمة $\ln(2)$ وتساوي إلى:

```
double v= 0.693147180559945309417232121458176568075500134360255254120678;
```

أي بعد إكمال القسمة وتصغير x سأدخل نتيجة القسمة وهي r في دالتنا السابقة ونقول للدالة $\ln$ أن تعطينا النتيجة التالية:

$$(x-1)*sum+j*v$$

الآن نذهب إلى مرحلة التطبيق ونضع دالة `while` هذه في الدالة `Ln`

```
#include <iostream>

using namespace std ;

double absolu(double x)
{
if(x>=0)
x=x ;
else
x=-x ;
```

```

return x;

}

double Ln(double x)

{

double v= 0.693147180559945309417232121458176568075500134360255254120678;

double epsilon =0 .000001 ;

double term=1, sum=1;

int i=1;

int j=0 ;

while(x>1)

{

x=x/2 ;

j++ ;

}

while(absolu(term/(i+1))>epsilon)

{

term=term*(1-x) ;

sum=sum+term/(i+1) ;

i++;

}

return (x-1)*sum+j*v ;

}

int main()

{

double x;

cout<<"Enter the value of x: "<<endl ;

```

```

    if(x<0)
        cout<<"Error !" ;
    if(x==0)
        cout<<"-Infinity" ;

    if(x>=1)

        cout<<"The value of Log(x) is : "<<Ln(x);

    if(x>0 && x<1)

        cout<<"The value of Log(x) is : "<<-Ln(1/x);

    return 0;

}

```

الآن قم بعملية التطبيق:

نعم كل شيء جيد ولكن أريد أن أزيد من دقة الحساب؟

هذا ليس مشكل إذهب إلى epsilon وغير من قيمته.

أنا سأخذها كما يلي:

```
double epsilon = 0.000000000000000001 ;
```

أي بدقة 10^{-16} ولاحظ الفرق الآن.

جيد الآن كل شيء يعمل جيداً.

مارأيكم أن أجعل دقة الحساب تصل إلى أكثر من ذلك؟

أنا أريد أن أصلها إلى أكثر من ذلك، أي سأجعل قيمة epsilon تصل إلى الحد $(-60)^{10}$ ، أي هكذا:

[illegible]

أو باختصار هكذا:

```
epsilon=1e-60 ;
```

الدونامج مازال شغال 100 بالمئة.

ومنه نهاية الدرس الأول حول حساب الدالة $\log$ ويبقى فقط التجميل والاختصارات أى الجودة:

أولاً بما أن القيمتين v و ϵ ثابتتين فمن الأفضل وضعهما في بداية البرنامج وذلك لتتفحص عدد المتغيرات أي على الشكل التالي:

```
#define epsilon 1e-60
```

```
#define v 0.693147180559945309417232121458176568075500134360255254120678
```

الدالة القيمة المطلق ل x يمكن اختصارها إلى مايلي:

```
double absolu(double x) {return (x>=0?x:-x);}
```

من الأفضل الإعلان على المتغيرات كلها قبل بداية أي عملية وإعطائها قيم بدايتها وقت الحاجة إليها، أي في الدالة Ln هكذا:

```
double Ln(double x){
```

```
double term, sum ;
```

```
int i, j ;
```

```
j=0;
```

```
while(x>1)
```

```
{
```

```
x=x/2 ;
```

```
j++ ;
```

```
}
```

```
i=1 ;
```

```
sum=1;
```

```
term=1;
```

```
while(absolu(term/(i+1))>epsilon)
```

```
{
```

```
term=term*(1-x) ;
```

```
sum=sum+term/(i+1) ;
```

```
i++;
```

```
}
```

```
return (x-1)*sum+j*v ;
```

```
}
```

من الأفضل استخدام المؤشر inline قبل بداية أي دالة أي على هذا الشكل:

```
inline double absolu(double x) {return (x>=0?x:-x);}
```

هذه العملية ستأخذ الدالة كلها وتضعها أمام العملية التي نريدها أي تسمح للبرنامج عدم الذهاب إلى موقع الحالة ولكن مباشرة يجد الدالة أمامه، أي أنه مثلاً من أجل الدالة `while(absolu(term/(i+1))>epsilon)` فإنه لا يذهب إلى موقع الدالة `absolu` فقط يجدها أمامه ويطبق مباشرة.

ومن مسلوئها أنها تزيد من حجم البرنامج ولكن ليس بالمقدار الذي نتصوره ولكن بضع من البيئات.

لذلك سأضع inline أمام الدالة `absolu` فقط لأنني أحتاجها عدة مرارة وذلك بسبب الدالة `while` ولا أضعها أمام `Ln` لأنني أحتاجها مرتين فقط.

التصريح ; `using namespace std` ; أحتاج منه فقط الدوال `cout` و `cin` و `endl` لذلك سألغي `using namespace` ; `std` ; وسأستخدم فقط مايلي: `std::cout` و `std::cin` و `std::endl`.

لذلك البرنامج يصريح كما يلي:

```
#include <iostream>
```

```
#define epsilon 1e-60
```

```
#define v 0.693147180559945309417232121458176568075500134360255254120678
```

```
inline double absolu(double x) {return (x>=0?x:-x);}
```

```
double Ln(double x){
```

```
double term, sum ;
```

```
int i, j ;
```

```
j=0;
```

```
while(x>1)
```

```
{
```

```
x=x/2 ;
```

```
j++ ;
```

```
}
```

```
i=1 ;
```

```
sum=1;
```

```
term=1;
```

```

while(absolu(term/(i+1))>epsilon)
{
term=term*(1-x) ;
sum=sum+term/(i+1) ;
i++;
}
return (x-1)*sum+j*v ;
}

int main()
{
double x;
std::cout<<"Enter the value of x: "<<std::endl ;
std::cin>>x;

    if(x<0)
        std::cout<<"Error ! " ;
    if(x==0)
        std::cout<<"-Infinity" ;

    if(x>=1)

std::cout<<std::endl<<"The value of Ln("<<x<<") is : "<<Ln(x);

    if(x>0 && x<1)

std::cout<<std::endl<<"The value of Ln("<<x<<") is : "<<-Ln(1/x);

return 0;
}

```

في الدرس القادم بإذن الله سيكون حول برمجة الدوال الأخرى واستخدام الهيدر والكلاس وهكذا.