EP578 Computing for Physicists

Topic 4

Functions

Department of Engineering Physics

University of Gaziantep

Course web page
`www.gantep.edu.tr/~bingul/ep578`

Oct 2011

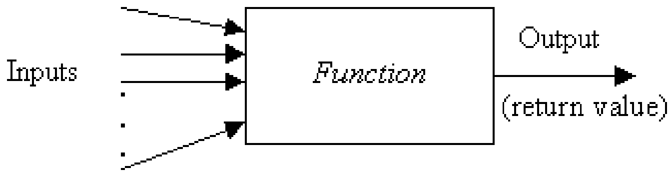
1. Introduction

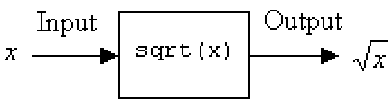
This lecture covers the following topics:

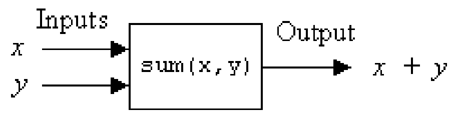
- Function Concept
- Defining & Using Functions
- `void` Functions
- Arguments passed by value and by reference
- Default Parameters
- Overloading Functions (Polymorphism)
- Macro Functions
- Examples

2. Functions (sub-programs)

- A function is an object that accepts some inputs and then outputs a result depending on the inputs.







3. Defining and Using Functions

- General form of a function declearation

```
type name(parameter 1, parameter2, ...){
  ...
    statements
  ...
}
```

- Example function to compute the area of a triangle

```
double area(double base, double height)
{
   double ar;
   ar = base * height / 2.0;
   return ar;
}
```

```
#include <iostream>
using namespace std;

double area(double base, double height)
{
    double ar;
    ar = base * height / 2.0;
    return ar;
}

int main()
{
   double a, b = 3.0, h = 4.0;

   a = area(b, h);
   cout << "area = " << a << endl;

   return 0;
}
```

```
area = 6
```



```
#include <iostream>
using namespace std;

// function prototype (fonksiyon örneği veya kalıbı)
double area(double, double);

int main(){
   double a , b = 3.0 ,h = 4.0;

   a = area(b, h); // function call
   cout << "area = " << a << endl;

   return 0;
}

// Function decleration
double area(double base, double height){
    double ar;
    ar = base * height / 2.0;
    return ar;
}
```

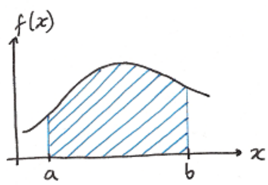
```
area = 6
```

```
#include <iostream>
using namespace std;

// function prototype (fonksiyon örneği veya kalıbı)
double area(double, double);

int main(){
  double a , b = 3.0 ,h = 4.0;

  a = area(b, h); // function call
  cout << "area = " << a << endl;

  return 0;
}

// Function decleration
double area(double base, double height){
   return base * height / 2.0;
}
```

```
area = 6
```


Example: Numerical Integration

- The integral of a function f(x) between the limits a and b is simply the area under the curve between a and b.



- Finding the integral analytically may be difficult and so a numerical solution may be necessary or simply convenient.

- **Problem**: Write a program to evaluate the integral

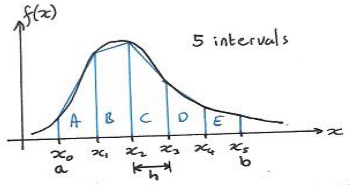
$$\int_0^2 e^{-x} dx$$

using the trapezoidal method for *n* = 100 parts.

Note that analytical solution is: `0.8646647`

We want the integral $\int_a^b f(x)\,dx$.

First consider approximating with five trapezoids:



$A = \frac{h}{2}(f(x_0)+f(x_1))$

$B = \frac{h}{2}(f(x_1)+f(x_2))$

$C = \frac{h}{2}(f(x_2)+f(x_3))$

$D = \frac{h}{2}(f(x_3)+f(x_4))$

$E = \frac{h}{2}(f(x_4)+f(x_5))$

$h = \frac{x_5 - x_0}{5} = \frac{b-a}{5}$

let $f_i \equiv f(x_i)$

$A+B+C+D+E$ = Extended Trapezoidal Formula (ETF).

$h(f_0/2 + f_1 + f_2 + f_3 + f_4 + f_5/2)$

For n intervals

$ETF = h\left(\frac{f_0}{2} + f_1 + f_2 + f_3 + \cdots + f_{n-1} + \frac{f_n}{2}\right)$

with $h = \frac{b-a}{n}$ $\quad x_i = a + ih, \; i = 0,1,2,\ldots,n$


```
// Numerical integration via the Extended Trapezoidal Formula (ETF)
#include <cmath>
#include <iostream>
using namespace std;

// integrant function
double f(double x) {
  double y = exp(-x);
  return y;
}

int main() {
  const int n=100;
  const double a=0.0, b=2.0;
  const double h = (b-a)/n;

  double etf = (f(a)+f(b))/2;

  for (int i=1; i<n; i++){
    etf += f(a+i*h);
  }
  etf *= h;

  cout.precision(7);
  cout << "The integral = " << etf << endl;
}
```

```
The integral = 0.8646935
```

```
// Numerical integration via the Extended Trapezoidal Formula (ETF)
#include <cmath>
#include <iostream>
using namespace std;

// integrant function
double f(double x) {
  double y = exp(-x);
  return y;
}
// integrate function
double integrate(double a, double b){
  const int n = 100;
  const double h = (b-a)/n;
  double etf = (f(a)+f(b))/2;

  for (int i=1; i<n; i++)
    etf += f(a+i*h);
  etf *= h;
  return etf;
}

int main() {
  double i = integrate(0.0, 2.0); // function call
  cout.precision(7);
  cout << "The integral = " << i <<
}
```

```
The integral = 0.8646935
```

4. void Functions

```
#include <iostream>
using namespace std;

// no value is returned
void printDouble(int a)
{
  cout << "Double of a:" << 2*a;
}

int main()
{
  printDouble(5);
}
```

```
Double of a: 10
```

```
#include <iostream>
using namespace std;

// no value is returned
void Message(void)
{
 cout << "I am a function";
}

int main()
{
  Message();
}
```

```
I am a function
```

5. Arguments pass by value & pass by ref.

```
#include <iostream>
using namespace std;

// arg. Pass by value
void Decrease(int a, int b){
   a--;
   b--;
}

int main()
{
  int x = 3, y = 8;

  cout << x << "   " << y << endl;

  Decrease(x, y);

  cout << x << "   " << y << endl;
}
```

```
3   8
3   8
```

```
#include <iostream>
using namespace std;

// arg. Pass by reference
void Decrease(int& a, int& b){
   a--;
   b--;
}

int main()
{
  int x=3, y=8;

  cout << x << "   " << y << endl;

  Decrease(x,y);

  cout << x << "   " << y << endl;
}
```

```
3   8
2   7
```



```
#include <iostream>
using namespace std;

void Convert(float, int& ,float&);

int main()
{
  float rx, x = 3.2;
  int   ix;

  Convert(x, ix, rx);

  cout << " x = " << x  << endl;
  cout << " ix= " << ix << endl;
  cout << " rx= " << rx << endl;
}

void Convert(float num, int& ip, float& rp)
{
  ip = num;
  rp = num - int(num);

}
```

6. Default Arguments

C++ allows a function to have a variable number of arguments.
Consider the second order polynomial function: $a + bx + cx^2$

```
#include <iostream>
using namespace std;

// -- optional parameters must all be listed last --
double p(double x, double a, double b =0, double c =0)
{
  return a + b*x + c*x*x;
}

int main(){
  double x = 1.0;

  cout <<  p(x, 7)       << endl;
  cout <<  p(x, 7, 6)    << endl;
  cout <<  p(x, 7, 6, 3) << endl;

  return 0;
}
```

```
7
13
16
```


Example: Newton-Raphson Method for f(x)=0

- For a function f(x), we can find the root satisfying f(x)=0 iteratively:

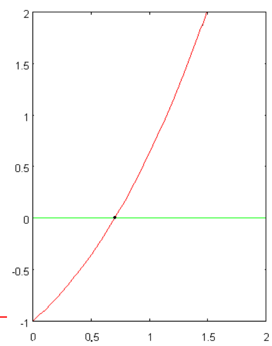
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

where x_0 is the initial root estimate
$E = f(x) / f'(x)$ is the error estimate
Iterations are terminated when E < 'predefined tolerance'

- **Problem**: Write a program to evaluate the root of the function:

$$f(x) = x^2 - e^{-x} = 0$$

Note that there is no analytical solution!



```cpp
// Finding the root of the equation f(x) = x*x - exp(-x)=0
#include <cmath>
#include <iostream>
using namespace std;

double f(double x) {  // the function
  double y = x*x - exp(-x);
  return y;
}
double d(double x){  // the derivative
  return 2*x+exp(-x);
}
double findRoot(double x=1.0){ // the root finder
  while(1){
     double err = f(x) / d(x);
     x = x - err;
     if( fabs(err)<1.e-6 ) break;
  }
  return x;
}

int main() {
  cout << findRoot() << endl; // function call
}
```


7. Overloading Functions (Polymorphism)

```cpp
#include <iostream>
using namespace std;

int topla(int x, int y){
   return x + y;
}

int topla(int x, int y, int z){
  return  x + y + z;
}

double topla(double x, double y){
 return x + y;
}

int main(){
  cout <<"topla(9,7)     = " << topla(9, 7)      << endl;
  cout <<"topla(3,6,2)   = " << topla(3, 6, 2)   << endl;
  cout <<"topla(3.1,4.7)= " << topla(3.1, 4.7) << endl;
  return 0;
}
```

```
topla(9,7)     =  17
topla(3,6,2)   =  11
topla(3.1,4.7)=  7.8
```

8. Macro Functions

A macro function, which is actually not a function, is defined by using `#define` preprocessor command.

The following macro function definitions are valid in C/C++:

```
#define square(x)  (x)*(x)
#define hypotenus(x,y)  sqrt((x)*(x)+(y)*(y))
#define delta(a,b,c)  ((b)*(b)-4*(a)*(c))
#define max(a,b)  ((a>b) ? a:b)
```



```
// Finding Pythagorian Triples
#include <iostream>
#include <cmath>
using namespace std;

// macro function definition
#define hypotenus(x,y) sqrt((x)*(x) + (y)*(y))
// symbolic constant definition
#define N 50

int main (){
  int a, b, c;

  cout << "Pythagorean Triples less than N = 50" << endl;

  for (a=1; a<=N; a++)
  for (b=a; b<=N; b++)
  for (c=1; c<=N; c++)
      if( c == hypotenus(a,b) )
          cout << "("
               << a << ", " << b << ", " << c
               << ")" << endl;
  return 0;
}
```

```
Pythagorean Triples less than N = 50
(3, 4, 5)
(5, 12, 13)
(6, 8, 10)
(7, 24, 25)
(8, 15, 17)
(9, 12, 15)
(9, 40, 41)
(10, 24, 26)
(12, 16, 20)
(12, 35, 37)
(14, 48, 50)
(15, 20, 25)
(15, 36, 39)
(16, 30, 34)
(18, 24, 30)
(20, 21, 29)
(21, 28, 35)
(24, 32, 40)
(27, 36, 45)
(30, 40, 50)
```


Homeworks

Solve the following problems. You have to prepare a pdf document and sent it to me until next lecture.
E-mail: bingul[at]gantep.edu.tr (*replace* [at] *with* @)

1. Write a function named `double perimeter(double r)` which returns the circumference of a circle of radius *r*.
   Use the function in a main program.

2. Write a macro function named `deBroglie(p)` which returns the de Broglie wavelength in meter of a particle whose momentum is *p* in kg.m/s. Use the function in a main program.

3. Write your own **`pow(x,y)`** by using **`log()`** and **`exp()`** functions.

4. Write only a function whose prototype is `double sum(int n)` that returns the sum first n terms of the series
`1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/32 + ...`

5. In mathematics the Stirling's approximation (or Stirling's formula) is an approximation for large factorials. The formula is written as:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Write a function named sfact(n) to calculate n! according to Stirling approximation. Note that return value of the function must be `double` since the formula generates real values. Use this function in a main program to output the integer numbers *k* = 1, 2, …, 100 and their factorials.

6. Using Plank's formula for a black-body radiator, write a C++ program to derive Wein law:

$$k_B T \lambda_{\max} = 0.2014$$

or

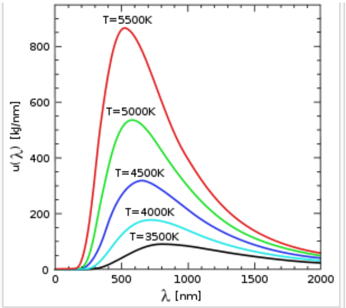
$$\lambda_{\max} T = 0.0029 \ \mathrm{m \cdot K}$$

Hint: Plank formula is given by:

$$u(\lambda) = \frac{8\pi hc}{\lambda^5}\frac{1}{\exp(hc/k_B T\lambda) - 1}$$

use dimensionless variable: $x = \dfrac{hc}{k_B T \lambda}$

and solve x satisfying: $\dfrac{du}{dx} = 0$



This diagram shows how the peak wavelength and total radiated amount vary with temperature according to Wien's displacement law. Although this plot shows relatively high temperatures, the same relationships hold true for any temperature down to absolute zero. Visible light is between 380 and 750 nm.

http://en.wikipedia.org/wiki/Thermal_radiation

7. Modify the second integration program to normalize ground state the wave-function of the electron in an infinite square well problem for *L* = 1 nm. That is find the constant *A* given below:

$$\Psi(x) = A\sin(\frac{\pi x}{L})$$