Functions

Function-definition

type name (*parameter-declaration-list*)
{
  statements
}

return *expression*; ~ Java

Function-declaration/ Function prototype

type name (parameter-declaration-list)

In C++ a global function must be declared before use !!

```
#include <iostream>
bool odd(int n);      //declaration of odd, function-prototype

bool even(int n)        //definition of even
{  if (n==0) return true; else return odd(n-1);
}

bool odd(int n)   // definition of odd
{  if (n==0) return false; else return even(n-1);
 }

int main()
{  int n; cin>> n;
   cout << (even(n)? "even" : "odd") << endl;
}
```

A function may be defined

- at the top level (in file scope)
- in a class as a method (in class scope)

Function definitions in local scope not allowed, i.e. no nested functions.

Functions

Default Arguments

Not in Java !!

```
int F(int a, int b, int c=10, int d=20){return a+b+c+d;}

cout << F(1,2,3,4);     //                    prints: 10
cout << F(1,2,3);       // ~ F(1,2,3,20)      prints: 26
cout << F(1,2);         // ~ F(1,2,10,20)     prints: 33
```

C-Array's as Arguments

a C-array
  is NOT an object
  does not contain its own size

```
double SUM(double A [], int size)
{  double s=0.0;
   for (int i=0; i< size; ++i){s += A[i];}
   return s;
}
```

Functions as Arguments

Not in Java !!
Functions only as methods

```
double f(double x) {return (x*x + 1.0/x);}

void plot(double fcn(double), double x0, double incr, int n)
{    for (int i = 0; i < n; ++i){
     cout << " x :" << x0 << "   f(x) : " << fcn(x0) << endl;
     x0 += incr;
     }
}

int main()
{  cout << "mapping function x*x + 1.0/x " << endl;
   plot(f, 0.01, 0.01, 100); return(0);
}
```

Array and Function types: read from inside out:

double A [] : A is an array of double's

double fcn(**double**): fcn is a function taking a double and giving a double result

Inline Functions

```
inline double cube(double x){return x*x*x;}
double y= 5.2;
… + cube(y) + …    compiled as  … + (y*y*y) + …
```

The compiler will not generate code for a function call to cube, but will replace the call cube(y) with (y*y*y)

Overloading Functions

~ Method Overloading in Java

```
void Print(int x){cout<< " integer= " << x; }
void Print(double x){ cout<< " double= " << x;}
…
Print(5); Print(5.2);     //prints " integer= 5 double= 5.2"
```


A Quick Introduction to Classes

```
class Vector2
{ public:
    Vector2(float a, float b){ x = a; y = b;}
    float inner(Vector2 a){return(x * a.x + y * a.y);}
  private:
    float x, y;
};
```

Methods become inline

Notice the Java dot notation for member selection

Notice the semicolon !!

Separation of Class-declaration and Implementation

```
class Vector2
{ public:
    Vector2(float a, float b);
    float inner(Vector2 a);
  private:
    float x, y;
};
```

Methods not inline unless specified in implementation

```
//Method Implementation:
Vector2::Vector2(float a, float b)
{  x = a;
   y = b;
}
float Vector2::inner(Vector2 a)
  {return(x * a.x + y * a.y);}
```

Use of Vector2:

```
Vector2 V1(2.1f, 3.4f);
Vector2 V2(3.3f, 6.8f);
float IP= V1.inner(V2);
```

File Scope / Local Scope

File Scope

```
#include <iostream>

double a =12.34;

int F (int n)
{ ...   n  ...
  ...   a  ...
}

double G (double x)
{  cout << a << "\n";
   ...F...
   int a= 10;
   cout << a << "\n";

   if (a > 0)
     {cout << a ;

     char a = 'z';

     cout << a << "\n";
     }
   cout << a << "\n"
}

void main()
{ a   F   G  }
```

Local Scope

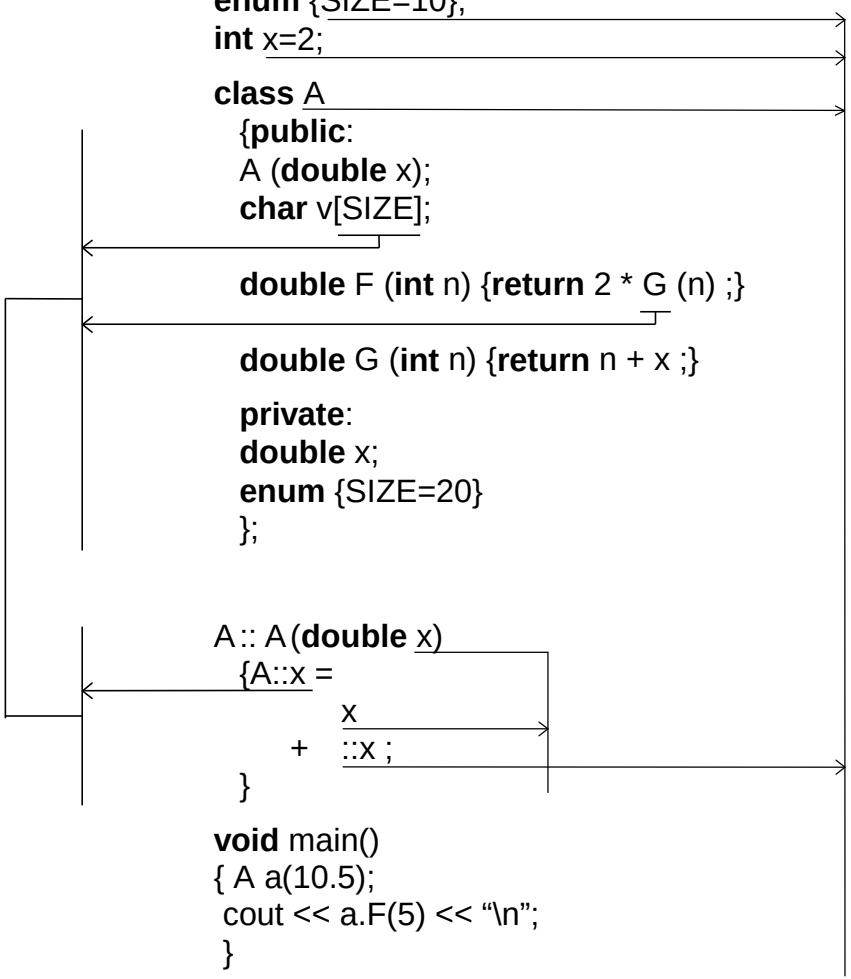
Class Scope

class scope operator:
Vector2 ::

```
class Vector2
{ public:
    Vector2(float a, float b);
    float inner(Vector2 a);

    private:
    float x, y;
};

class Vector3
{ public:
    Vector3(float a, float b, float c);
    float inner(Vector3 a);

    private:
    float x, y, z;
};

Vector2::Vector2(float a, float b)
{  x = a;
   y = b;
}

Vector3::Vector3(float a, float b, float c)
{  x = a;
   y = b;
   z= c;
}

float Vector2::inner(Vector2 a)
   {return(x * a.x + y * a.y);}

float Vector3::inner(Vector3 a)
   {return(x * a.x + y * a.y + z * a.z);}
```

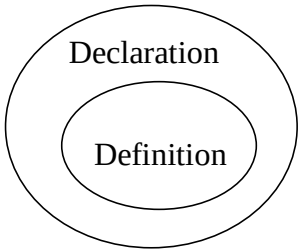
Class Scopes

Class / File / Local Scope



Class scope operator: A ::	File scope operator: ::

Declarations / Definitions



- A **declaration**
  introduces one or more *names* into a program.
  It *provides* the necessary *information* about types etc. so that the compiler can generate the required code.
- A **definition**
  is a declaration which causes *storage allocation* for a variable or constant or
  specifies a function body or class body.
  At most 1 definition in a program
  per variable, constant, function and class.

```
double G (int);              // function declaration / prototype

double F (int n)             // function definition
  {return 2 * G (n) ;
  }

double G (int n)             // function definition
  {int a = 1;                    // local variabel definition
   extern int gb;                // global variabel declaration
   extern int z [ ];             // global array declaration
   return n + a + gb + z[n] ;
  }

enum{SIZE=20, EX=5};         // definition of SIZE and EX

int gb = 100;                // global variabel definition

int z[SIZE];                 // global array definition

void main()
{for (int i=0; i<SIZE; i++) z[i]= i*i;
 cout << F(EX) << "\n";
}
```

Declarations / Definitions

```
class Vector2                          // class definition
{ public:
    Vector2(float a, float b);         // member function declaration
    float inner(Vector2 a);            // member function declaration
    int nonzero(){x !=0.0|| y !=0.0;}  // member function definition
  private:
    float x, y;                        // member data definition
};
Vector2::Vector2(float a, float b)     // member function definition
  { x = a;
    y = b;
  }


class B;                 // class declaration
class A                  // class definition
  { public:
      A * left;
      B * right;
      int val;
  };
class B                  // class definition
  { public:
      A * next;
      int val;
  };
void main()
  {.......
  }
```


Storage Allocation/Storage Classes

Static storage variables (belong to the static storage class):

- Global variables (with internal or external linkage)
- Local variables with the **static** specifier

Static storage variables are initialised to zero by default. They retain their storage location during the whole program execution.

Automatic storage variables (belong to the automatic storage class):

- Local variables in functions and blocks (not explicitly declared static)

Automatic storage variables exist during the execution of the function/block in which they are local. Allocated in a stack.

```
double R= 3.2;              //R: static
int F(int n)                // n: automatic
  { int m= n+2;             // m: automatic
    static int count=0;     // count: static
    ++count;
    if (n>1){return m*F(n-1);}
      else {cout<< count<< endl; return 1;}
  }
double A[100];              //A: static
int main()
{ cout << F(10) << endl;
…
}
```

Compilation Units, Internal/External Linkage

Saccnt.C :

```
int acct=0;

static double acct_bal [1000];

static double sum()
  {double s=0.0;
   for(int i=0;i<1000; i++){s+=acct_bal[i];}
   return s;
  }

void deposit(double amt)
  {acct_bal[acct] += amt;
  }

double balance()
  {return acct_bal[acct];
  }

double total()
  {return sum();
  }
```

Internal Linkage

Client.C:

```
extern int acct;
void deposit(double);
double balance(void);
double total(void);

#include <iostream.h>

void main()
  {  acct= 225; deposit(4536.5); deposit(2000);
     acct= 899; deposit(325.0);  deposit(85.25);
     acct=225; cout << balance() << "\n";
     acct=899; cout << balance() << "\n";
     cout << "total=" << total() << "\n";
     }
```

External Linkage

Compilation Units, Separate Compilation

Saccnt.C :

```
int acct=0;

static double acct_bal [1000];

static double sum()
{double s=0.0;
 for(int i=0;i<1000; i++)
   {s+=acct_bal[i];}
 return s;
}

void deposit(double amt)
{acct_bal[acct] += amt;
}

double balance()
{return acct_bal[acct];
}

double total()
{return sum();
}
```

g++ -c Saccnt.C (compile)

Saccnt.o:

myacct.exe:

g++ -o myacct.exe Saccnt.o Client.o (link)

Client.C:

```
extern int acct;
void deposit(double);
double balance(void);
double total(void);

#include <iostream.h>

void main()
{   acct= 225;
    deposit(4536.5); deposit(2000);
    acct= 899;
    deposit(325.0);  deposit(85.25);
    acct=225;
    cout << balance() << "\n";
    acct=899;
    cout << balance() << "\n";
    cout << "total=" << total() << "\n";
    }
```

Client.o:

g++ -c Client.C (compile)

Compilation Units

Server.C:

```
int acct=0;

static double acct_bal [1000];

static double sum()
   {double s=0.0;
    for(int i=0;i<1000; i++)
      {s+=acct_bal[i];}
    return s;
   }

void deposit(double amt)
   {acct_bal[acct] += amt;
   }

double balance()
   {return acct_bal[acct];
   }

double total()
   {return sum();
   }
```

Client1.C:

```
extern int acct;
void deposit(double);
double balance(void);
double total(void);

.....

{

... deposit..

... balance ...

    total ..

... acct ...

}
```

Client2.C:

```
extern int acct;
void deposit(double);
double balance(void);
double total(void);

.....

{

... deposit..

... balance ...

    total ...

... acct ...

}
```


Compilation Units, Header Files

Server.h:

```
extern int acct;
void deposit(double);
double balance(void);
double total(void);
```

Server.C:

```
#include "Server.h"

int acct=0;

static double acct_bal [1000];

static double sum()
   {double s=0.0;
    for(int i=0;i<1000; i++)
      {s+=acct_bal[i];}
    return s;
   }

void deposit(double amt)
   {acct_bal[acct] += amt;
   }

double balance()
   {return acct_bal[acct];
   }

double total()
   {return sum();
   }
```

Client1.C:

```
#include "Server.h"

.....

{

... deposit..

... balance ...

    total ..

... accnt ...

}
```

Client2.C:

```
#include "Server.h"

.....

{

... deposit..

... balance ...

    total ...

... accnt ...

}
```

Separation of Class-declaration and Implementation

```
/////File: vector2.h

class Vector2
{ public:
     Vector2(float a, float b);
     float inner(Vector2 a);

  private:
     float x, y;
};
```

```
/////File vector2.cpp, Method Implementation:

#include "vector2.h"

Vector2::Vector2(float a, float b)
{  x = a;
   y = b;
}

float Vector2::inner(Vector2 a)
  {return(x * a.x + y * a.y);}
```

```
///// File client.cpp

#include <iostream>
…
#include "vector2.h"

Vector2 V1(2.1f, 3.4f);
Vector2 V2(3.3f, 6.8f);

float IP= V1.inner(V2);

cout << IP << endl;
```