Functions
Recursion

C++ functions

- Declare/prototype
  ```cpp
  int myfunction (int x);
  ```

- Define
  ```cpp
  int myfunction (int x){
     int y = x*x;
     return y;
  }
  ```

- Call
  ```cpp
  int a;
  a = myfunction (7);
  ```
function call flow

```cpp
void first()
{
    cout << "I am now inside the function first.\n";
}

void second()
{
    cout << "I am now inside the function second.\n";
}

int main()
{
    cout << "I am starting in function main.\n";
    first();
    second();
    cout << "Back in function main again.\n";
    return 0;
}
```

types

- type of function (of the return value)
  - double myfunction (....)

- type of arguments
  - double myfunction (int a, double b, char c)

- the types have to be consistent between declaration, definition and call
function arguments

CALL
area = PI * square(radius);

DEFINITION

double square(double number
  {
    return number * number;
  }

arguments by value

• value is copied to parameter/argument
• parameters have the scope the function
  • same as a local variable
**return**

- returns the function output value to the call instruction
  - has to match function output type

```c
int myfunction (int x){
    int y = x*x;
    return y;
}
```

- terminates the function
  - even if there are more statements to execute

---

**Argument default value**

- if no argument is given at the call, use a default value
  - default value given in function definition

```c
double log5(double x=125){
...
    ...
}
```
Scope: local and global

- **global**: define outside any function
  - visible everywhere (preserve value)

- **local**: define inside a function (or block)
  - invisible outside the definition block

Static variables

- **static local variables** do not get erased when function/block terminates

  - the next time the function is called, a static variable still has the previous value
    - initialized only one time

```c
int function (int param){
    static double myvar=0;//initialization happens only at the first function call
    ... do something ...
}
```
Overload function names

myfunction does \( y = 2x_1 - 3x_2 \)

- I want it to work for doubles and int types

- int myfunction (int, int)
- double myfunction (double, double)
- double myfunction (int, double)
- double myfunction (double, int)

Arguments by reference

usually (call by value), if the argument passed to the function changes value inside the function, the variable used as argument does not.

- to modify the variable used as argument at the call, pass the argument by reference

```cpp
//call
int a=0,b=0;
b = f1(a);//now a=0, b=1

//function definition
int f1 (int x){
    x = x +1;
    return x;
}
```

```cpp
//call
int a=0,b=0;
b = f2(a);//now a=1, b=1

//function definition
int f2 (int & x){
    x = x +1;
    return x;
}
```
Recursive calls

Recursion of a function

- A function that calls itself
  - OR cyclic: function f calls function g; function g calls function f
- Creates a stack of calls
- Calls terminate in the reverse order of calling
- Local variables are defined independently for each call
Recursion: flow

void message(int times) {
    if (times > 0) {
        cout << "call t=" << times << "\n";
        message(times - 1);
    }
}

Solving a problem recursively

- recognize recursive/inductive nature
  - many problems easier to solve with a loop
- build up the recursion mechanism
- follow the principle of mathematical induction
- most often, find an “invariant” operation
  - can be a math formula
  - can be an inductive form
  - do one step of it, then call the recursion (or the other way)
- look carefully at the base cases
Sum of first n integers

- \( S(n) = 1 + 2 + 3 + 4 + \ldots + n = \frac{n(n+1)}{2} \)
- induction : \( S(n) = S(n-1) + n = \frac{(n-1)n}{2} + n \)
  \( = \frac{n(n+1)}{2} \)

recursion

```c
int sum (int n){
    if (n<0) {
        cout<<“ERROR, negative”;
        return -1;
    }
    if (n==0) return 0;
    return n + sum(n-1);
}
```
Factorial

- $n! = 1 \times 2 \times 3 \times \ldots \times n$
- induction: $n! = n \times (n-1)!$
  - $1!=0!=1$
- can be very very large
  - $10! = 3628800$
  - $50! \approx 3.0414 \times 10^{64}$

```cpp
long factorial (long n){
    cout<< "call: factorial(\"<<n<<\")\n";
    int out;
    if(n<=1) out=1;
    else out = n*factorial(n-1);
    cout<<"return: factorial(\"<<n<<\")\n";
    return out;
}
```
Tower of Hanoi

- three towers/rods A, B, C
- A contains pegs 1 to n, in order, n at the bottom
- B, C empty
- TASK: move all pegs to A such that
  - a peg at a time
  - only top peg of a tower can move
  - peg can “sit” only on higher value pegs

Original setup.  First move: Move disc 1 to peg 3.
Tower of Hanoi

1. Original setup.
2. Second move: Move disc 2 to peg 2.
3. Third move: Move disc 1 to peg 2.
5. Fifth move: Move disc 1 to peg 1.
function $f$: moves top $k$ pegs from tower $X$ to tower $Y$
- leaves all pegs existing on $Z$ and $Y$ unmoved
- leaves all pegs on tower $X$ below top $k$ unmoved

function $f$ is recursive
- moves top $k-1$ pegs from $X$ to $Z$ (recursive call)
- moves $k$ peg from $X$ to $Y$
- moves top $k-1$ pegs from $Z$ to $Y$ (recursive call)
Euclid GCD

- given positive integers $a$ and $b$
  - find $d = \gcd(a, b)$
  - find integers $m, n$ such that $a \cdot m + b \cdot n = d$. Do they always exist?

- recursion: if $a > b$ and $a = q \cdot b + r$ then
  - $\gcd(a, b) = \gcd(b, r)$
  - what about $m$ and $n$?

Euclid GCD: find linear coefficients

- given positive integers $a$ and $b$
  - find $d = \gcd(a, b)$
  - find integers $m, n$ such that $a \cdot m + b \cdot n = d$. Do they always exist?

- recursion: if $a > b$ and $a = q \cdot b + r$ then
  - $\gcd(a, b) = \gcd(b, r)$
  - $m(a, b) = m(b, r)$
  - $n(a, b) = m(b, r) - q \cdot n(b, r)$
Fibonacci numbers

- Problem defined with recursion
- \( F(n+2) = F(n) + F(n+1) \)
  - \( F(0) = 0; F(1) = 1 \)

```c
int Fibonacci (int n){
    if (n<=1) return n;
    else return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Count characters

- preview the notion of ARRAY
- char s[200]; //array of 200 characters
  - different type than class string
- can be accessed as s[0], s[1], ..., s[199]
  - char a = s[3];
- works for any type
  - double d[10]; int i[100];
- C++ does not check for array bounds !!
Count characters

- start counting at position 1
  - record 1 if character find,
  - keep looking at next position
- can be a loop
- can be a recursion

Binary search

- Find a specific value V in a sorted array A[ ]
- Start with array indices i=0, j=last, m=middle
- Compare A[m] to V and decide where in the array to look next
  - recursive call
  - or a loop
- Why binary search and not simply check all elements?
Binary search

How long is going to take? (worst case)

In algorithms, how long means how many steps/instructions
  - as a function of input $n = \text{size of array}$

we dont want an exact time/value
  - "linear" = like $n = \text{about CONSTANT} \times n$
  - "quadratic" = like $n^2 = \text{about CONSTANT} \times n^2$
  - $\text{CONST}\log n$, $\text{CONST}\times n\log n$, etc

Binary Search takes $\text{CONST}\times\log(n)$ steps, in worst case