Functions

-overloaded functions

-function templates

-recursive functions

overloaded functions

Int a[5]={3,2,1,9,8};

String p[4]={“Mon”,”Tues”,”Wed”,”Thurs”};

display(a, 5); ~~~~~~~~~~~~~

void display(int x[], int nSize)

//arrays are always passed by reference

{for(int I =0; i<nSize; ++i)

{cout<<x[i]<<’\t’;}

cout<<endl;

}

display(pz 4);

void display(string x[], int nSize)

{

for(int i=0; i<nSize; ++i)

{cout<<x[i]<<’\t’;}

cour<<endl;

}

//^ those are overloaded functions

Function Templates

void display( x[], int nSize)

{

for(int i=0; i<nSize; ++i)

{cout<<x[i]<<’\t’}

cout<<endl;

}

//to give power to that statement above we need to make the ff changes

void display(T x[], int nSize) //manmade type so I’ll use capital letter T

Recursive Functions

Math

***f***(n){n+f(n-1) if n>1

called recursive definitions

***f*** (4)=4+ ***f*** (~~3~~)

^3+ ***f(~~2~~)***

***^2+f~~(~~*~~1~~*)***

***^***1

***f***(4)=10 //forget it

c++ version

int f(int n)

{

if (n==1)

{return 1;}

else

{return n+f(n-1);}

}

calling statement:

*f(3);*

void f(int n)

{

cout<<n;

if(n>=1) f(n-1);

}//refer to table below

in recursive functions

//stack is involved

n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_f(n)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_output

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

3>=1 3 2 1 0

3210

2>=1

1>=1

0

calling statement

cout<< ***f*** (4)

^ 4+*f(3) //is 3=1? nope*

*^3+f(2) //is 2=1? nope*

*^2+ f(1)* //is = to 1

^ 1

cout<<f(10) //bec 4+3+2+1=10

Example: trace the following

Calling statement:

*f(3);*

void f(int n)

{

if(n>=1) *f(*n-1);

cout<<n;

}

//stack is involved

n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_f(n)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_output

|  |
| --- |
|  |
|  |
|  |
|  |
| ~~cout<<1~~ |
| ~~cout<<2~~ |
| ~~cout<<3~~ |

3>=1

0 1 2 3

2>=1

1>=1

0

void *f(int n)*

*{*

cout<<n;

if(n>=1) *f*(n\*1);

cout<<n;

*}*

//stack is involved

f(n)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_output

|  |
| --- |
|  |
|  |
|  |
|  |
| ~~cout<<1~~ |
| ~~cout<<2~~ |
| ~~cout<<3~~ |

f(3) 3>=1 //yes

3 2 1 0 0 1 2 3

f(2) 2>=1 //yes

f(1) 1>=1 //yes

f(0) 0>=1 //no

//fastest way to trace recursive function

//learn how to trace recursive functions

Calling stack

*f*(3);

cout<<3 f(2) cout<<3

cout<<2 *f*(1) cout<<2

cout<<1 f(0) cout<<1

cout<<0 \_ cout<<0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 2 1 0 0 1 2 3

//just learn how to trace recursive functions

Fibonacci numbers

1 1 2 3 5 6 13 21. . . . //the next number is the sum of the last 2 numbers

Fib(1) Fib(2)

Fib(n)=Fib(n-1)+Fib(n-2)

if n=1, 2

Fib(n)={1

{Fib(n-1)+Fib(n-2)

long Fib(int n) //long bec it’s a long number!

{

if(n==1 || n==2) return 1;

else return Fin(n-1)+Fib(n-2);

}

fib(7)

fib(6) + fib(5)

fib(5) + fib(4 fib(3)+fib(2)

…… etc… , //the total time of calculation for that is called runtime

//this is a slow function

\*compute the total of the following with recursive function

2^2+2^3+2^3+. . . .+2^n

//in c++ we need a recursive formula first

//to find a formula for the expression

1. f(n)=2^2+2^3+. . .+2^n
2. let n=5: f(5)= 2^2+2^3+…+2^5

f(~~5~~n)=\_\_\_f(~~4~~ n-1)\_\_\_+2^~~5~~n

f(n)=f(n-1)+2^n

f(n) {2^2=4 if n=2

{f(n-1)+2^n if n>2

//function on next page

int f(int n)

{ if(n==2) return 4;

else return pow(2,n)+f(n-1);

}

//in mathematics this is called math modeling, I want to die

trace the following function and show its output

calling statement: f(5)

void f(int n)

{

if(n>=1

{

f(n-1)

cout<<n;

f(n-2);

}

}

f(5);

f(~~5 minus 1~~ = 4) 5 f(5-2)

f(3) 4 f(2)

f(2) 3 f(1)

//next page

f(1) 2 f(0)

f(0) 2 f(-1)

- 2 - 2 - 3 2 4 22 4 2232=223242252232