

# COL 100 Lecture 11

Note Title

30-08-2018

Review:

Declaration order

Pass by value

Pass by reference

- can't pass-in non-variables
- 

```
void addFive (int &x)
{
    cout << res x + 5;
}

int main()
{
    int y = 10;
    addFive (y);
    cout << y; // 15
    return 0;
}
```

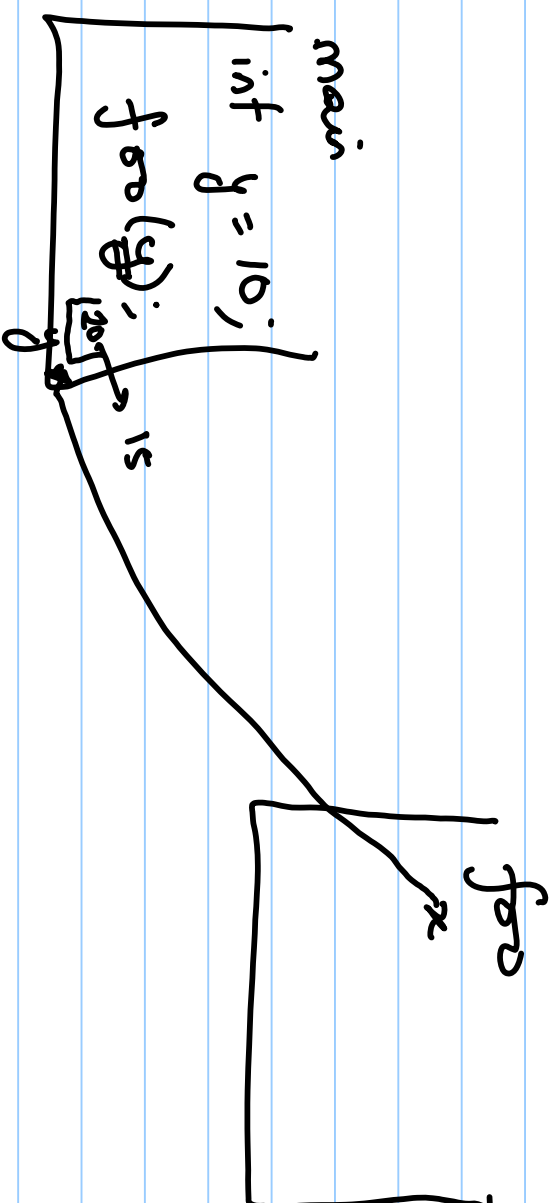
caller  
caller

void foo (int x)



Call/Pass by reference

void foo (int &x)



# Efficient?

→ If the passed value is large, pass-by-reference is more efficient

→ If the passed value is small (e.g. primitive types like char, int, etc.), then pass-by-value is more efficient

Advantage of pass-by-value:

calls can be over  
that variable is  
not modified

## Const parameters

```
void foo ( const string & s )
```

the variable  
will not be  
modified

pass by reference

s = "hello"; error

```
}
```

```
int main()
```

```
{    string s = "This is a really long string";  
    for (int i = 0; i < s.length(); i++)  
        cout << s[i]; // unmodified!  
}
```

```
}
```

string t = "I am a very long string";

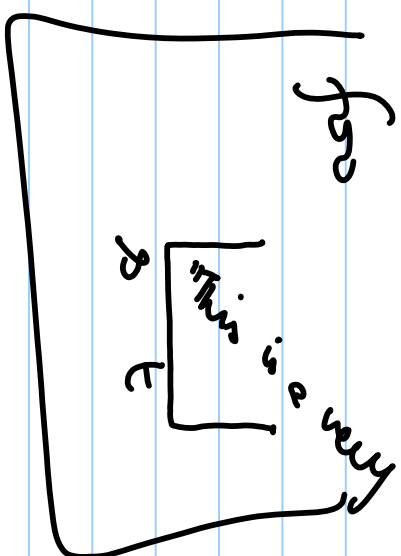
```
void foo(const string &s)
```

```
{  
    // both s & t refer/bind to the  
    t = "hello" variable  
    cout << s;  
}
```

```
{  
    int main()
```

```
{  
    foo(t);  
}
```

"نہیں ہوا" very long for

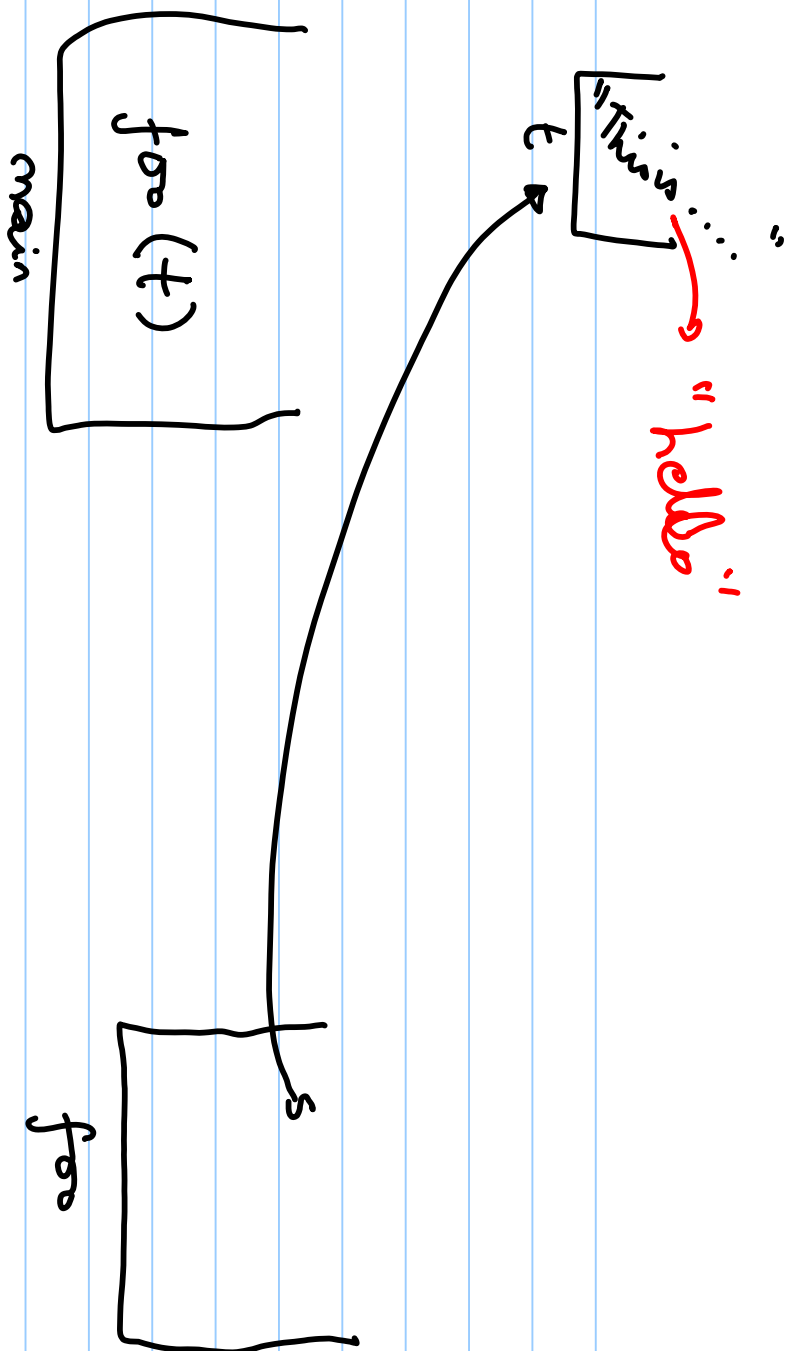


for

this is

E





```
1 string t = "I am a . . . .";
```

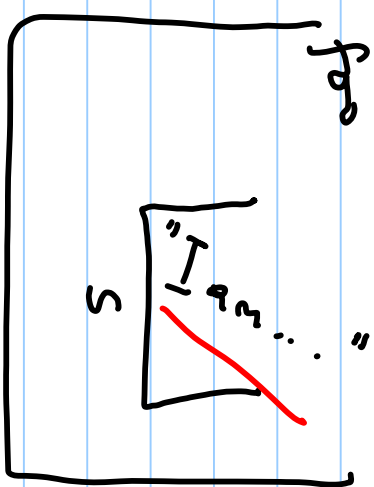
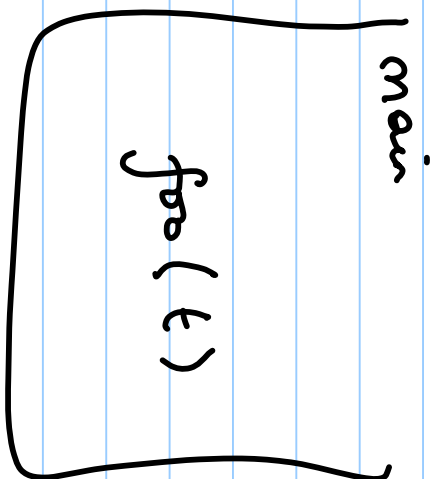
```
2 void foo (string s)
```

```
3     s t = "hello";  
4     cout << s;
```

```
5
```

```
6 int main()
```

```
7 {  
8     foo(t);  
9 }
```



```
void for (const string & s)
```

```
{
```

```
{
```

```
{ int main()
```

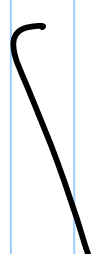
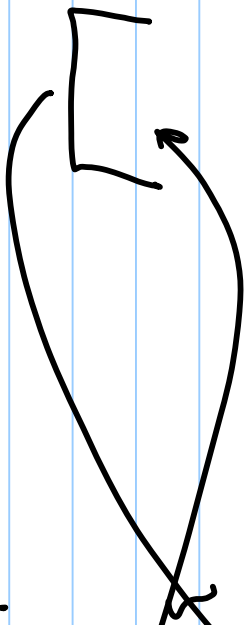
```
{ for ("hello");
```

```
}
```

```
✓
```

void bar (int &i)

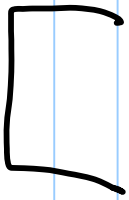
bar ( 3 + 4 \* x - 7 );



void bar ( int &x) <sup>the var get modified</sup>  
}

...

}



bar (10); X  
bar (y+3); X

int  $\infty$  sumDigits (int n)

## Output parameters

```
void sumAndProductDigits (int n, int &sum, int &prod)
```

```
{
```

```
    sum = ...;
```

```
    product = ...;
```

```
}
```

```
int x, y, z; x = 346;
```

```
sumAndProductDigits (x, y, z);
```

→



write a function to "quadratic" to find roots of quadratic equation:

$$ax^2 + bx + c = 0$$

$$|| \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

void quadratic( int a, ~~bx~~ int b, int c,  
double & root1,  
double & root2 )

double d = sqrt( b\*b - 4\*a\*c );

root1 = (-b + d) / (2\*a);

root2 = (-b - d) / (2\*a);

}

double quadratic (int a, int b, int c, double &root1, double &root2)

{

double d = ...

double x =  $(-b + d) / (2 * a);$

root1 =  $(-b - d) / (2 * a);$

return x;

}

# Decomposition

Properties of good function:

- Fully performs a single coherent task
- Does not do too large a share of work
- Not unnecessarily connected to other functions.
  - eg: no chaining of functions

✓

subtract 7 ( )

addFive ( )

{  
int main()

addFive(x)  
subtractSeven(x);

}

Chairing  
subtract 7

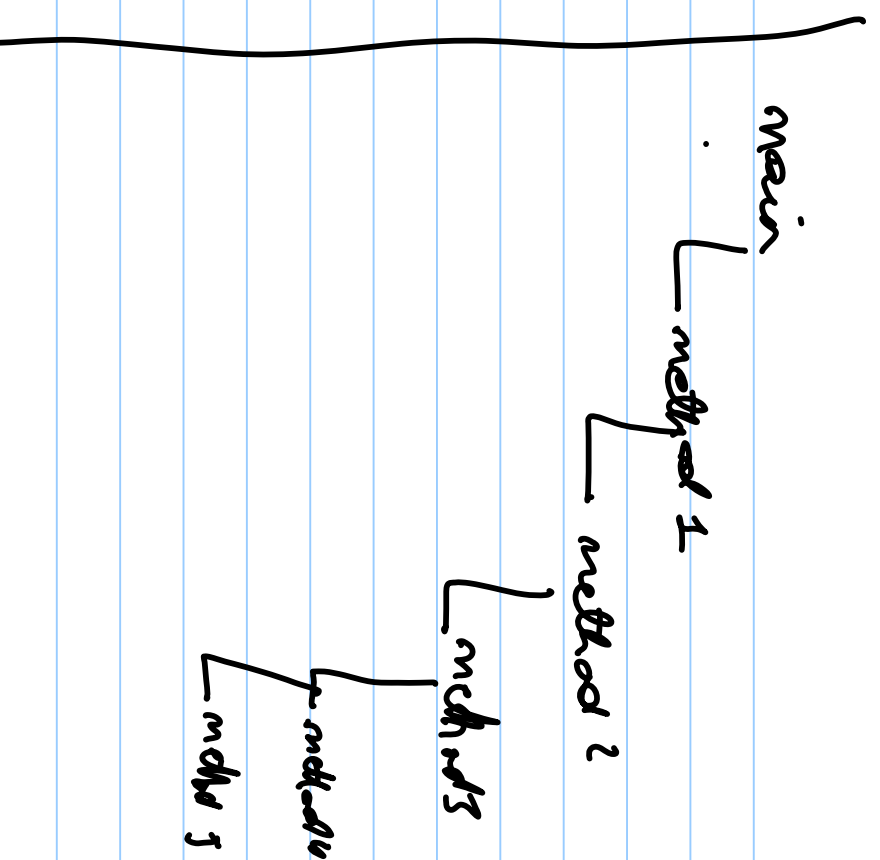
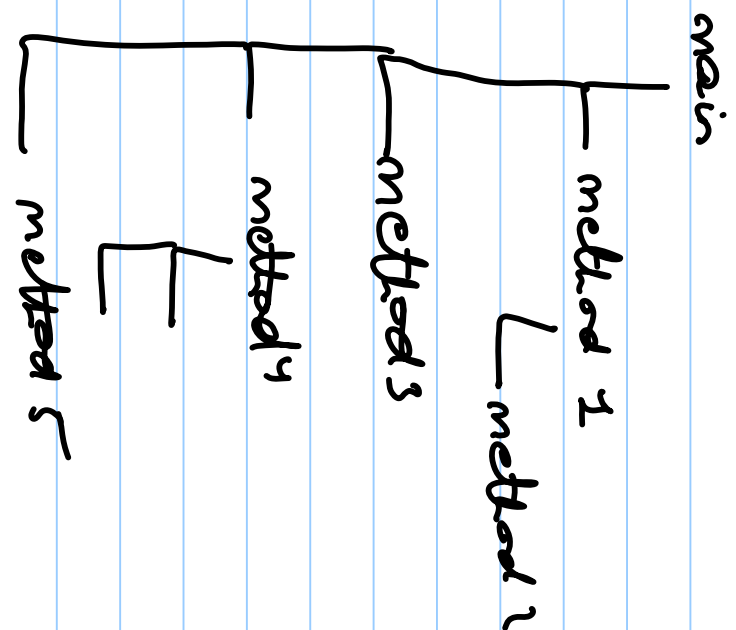
{  
addFive And Subtract 7()

{  
x = x - 5  
subtract 7(x)

{  
int main()

addFive And Subtract 7()

}



# Strings

→ C++ string  
string s = "hello";

string s = "Hi COL100!";

String is a sequence of characters:  
1 types of strings in C++

8  

0	1	2	3	4	5	6	7	8
H	i		C	O	L	1	0	0

!

square brackets → represents a character at index 3 is a.  
↓  
'c'

char c1 = s[3]; // 'c'  
char c2 = s.at(3); // 'c'

member function  
of strings

at(3) X

→ C string

("abc") . at(2); X

string("abc") . at(2); ✓

"abc"

:

C string

do not have junction  
member  
also separate of character  
but a different  
representation  
internally



char a[100]; // C string with at most 2, 100 characters

~~a~~ a[2] = ' ';

a.at(2) X

string("abc")

## Converting between C-strings and C++ strings

```
char cs[]; // C  
string cpps; // C++
```

```
string (cs) } C-strings to  
string ("abc") } C++ strings
```

```
char c[] = cpps.c_str()
```

char c1 = string("abc").at(2); ✓  
// c1 == 'c'

~~"abc"[2]~~

## C++ strings

string s = "Hi COL100!";

char c1 = s.at(3); // c1 == 'C'

What are checks?

'0', '1' ... '9'

'a', 'b' ... 'z', 'A' ... 'Z',

'!', ...

'\_'

...'

'\n' ...

# ASCII representation of Characters

...	:	'C'
72	:	'B'
71	:	
...	:	

}

255  
ASCII  
values

:'

```
char c1 = 'a';
int i1 = c1;
cout << i1;
```

ASCII value of  
first character

char c = '72';

cout << c;

bool isAlpha (int i) : i represents  
bool isDigit (int i) : i is a digit