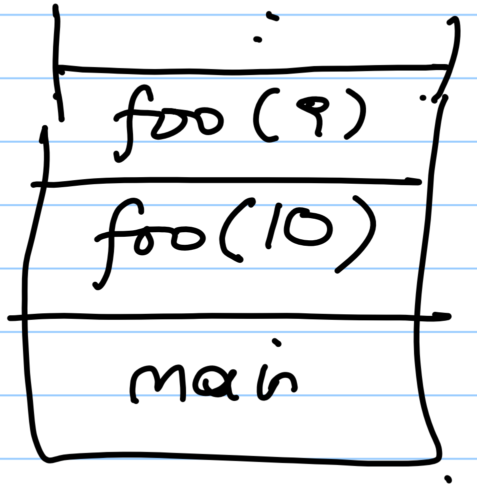
COL100 Lecture 28

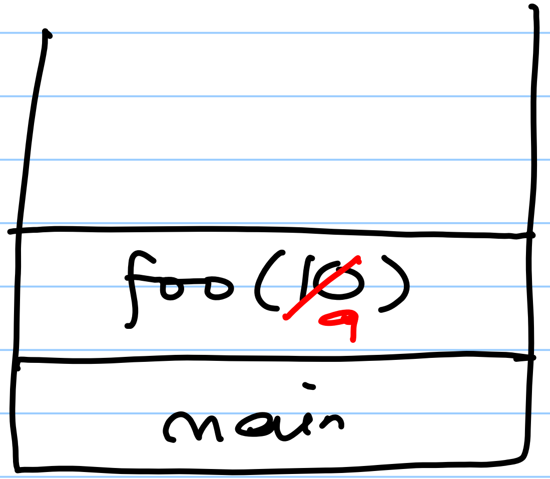
Note Title 15-11-2018

(last lecture, yay!)

Tail recursion:



Recursion



```
int  fact ( int  n)
{
    if  (n == 0)
        return 1 ;
    else {
        return  n * fact (n-1);
    }
}
```

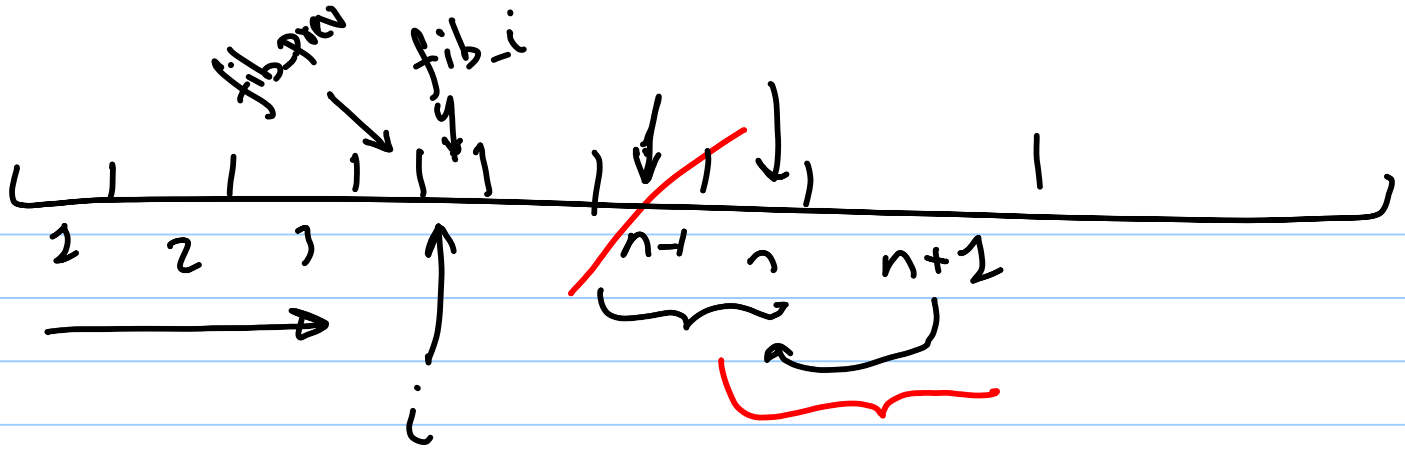
not tail recursive

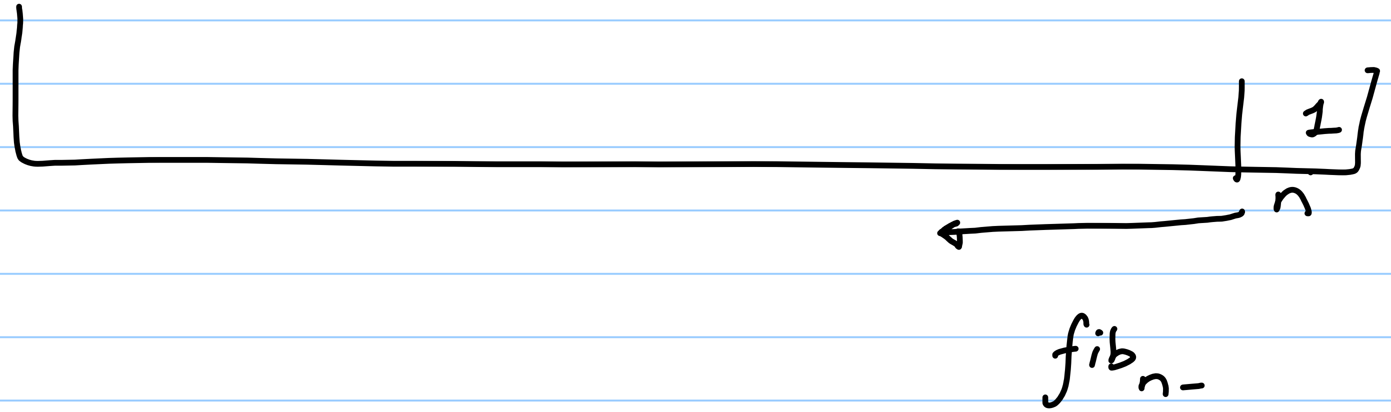
```
// computes accum * n! → by computing
                         (accum * n) x (n-1)!
                                       recursive
                                       call
int   fact_helper (int n, int accum)
{
      if   (n == 0) {
             return  accum;
      } else {
      return fact_helper(n-1, accum * n);
      }
}
```

no loops

```
int   fact (int  n)
{
      return   fact_helper(n, 1);
}
```

```
int fact (int n)
{
    int accum = 1;
          while (n != 0)
recursive {
case          int new_n = n - 1;
              int new_accum = accum * n;
              n = new_n;
              accum = new_accum;
          }
    base case : return accum;
}
```


fib_prev
fib_i
1
2
3
i
n-1
n
n+1


1
n
fib_{n-}

```
int    fib_helper(int n, int i, int fib_i,
                                 int fib_prev)
{
      if   (i == n)  {
             return  fib_i;
      } else  {
            return fib_helper(n, i+1, fib_i+
                                       fib_prev,
                                       fib_i);
      }
}

int   fib (int  n)
{
      return  fib_helper (n, 1, 1, 0);
}
```

tail recursive

```
int    fib (int   n)
{
            int   i =  1;
            int   fib_i = 1;
             int  fib_prev = 0;
            while  ( i  != n)
            {    int    new_i =   i+1;
                 int    new_fib_i = fib_i+fib_prev;
                  int    new_fib_prev = fib_i
                   i = new_i;
                   fib_i = new_fib_i ,
                   fib_prev =  new_fib_prev;
            }
            return  fib_i ;
    }
```

i == N - n + 1

```
int   fib_helper (int  n,   int  fib_i,  int fib_prev)
{
        if   (n == 1)   {
                 return   fib_i;
        } else   {
        return  fib_helper (n - 1, fib_i + fib_prev,
                                            fib_i ) ;
        }
}
int   fib (int   n)
{
        return   fib_helper (n, 1 , 0) ;
}
```

```
int   fib (int   n)
{
        int    fib_i = 1;
        int    fib_prev = 0;
        while (n != 1)
        {
            int   new_n = n - 1;
            int   new_fib_i = fib_i + fib_prev;
            int   new_fib_prev = fib_i;
            n = new_n;
            fib_i = new_fib_i;
        }   fib_prev = new_fib_prev;

        return fib_i;
}
```

Exhaustive Search

- Exploring every possible combination from a set of choices

General pseudo-code algorithm for exhausti search

Choosing

→ we iterate over "decisions". what are we iterating over here?

→ what are the "choices" for each decision?

Exploring

→ How can we "represent" the choice and the decisions made so far

Un-choosing

→ How do we "un-modify" the choices in the previous step

or copy

Base case

printAllBinary(2):

```
0 0
0 1
1 0
1 1
```

printAllBinary(3):

```
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

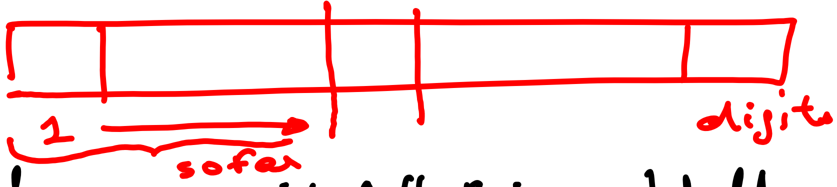
decision? character at ith position

choices? 0 or 1

representation?
stringSoFar

Unchoosing?
Create new strings for every call

Base case?
when all decisions have been made
print stringSoFar

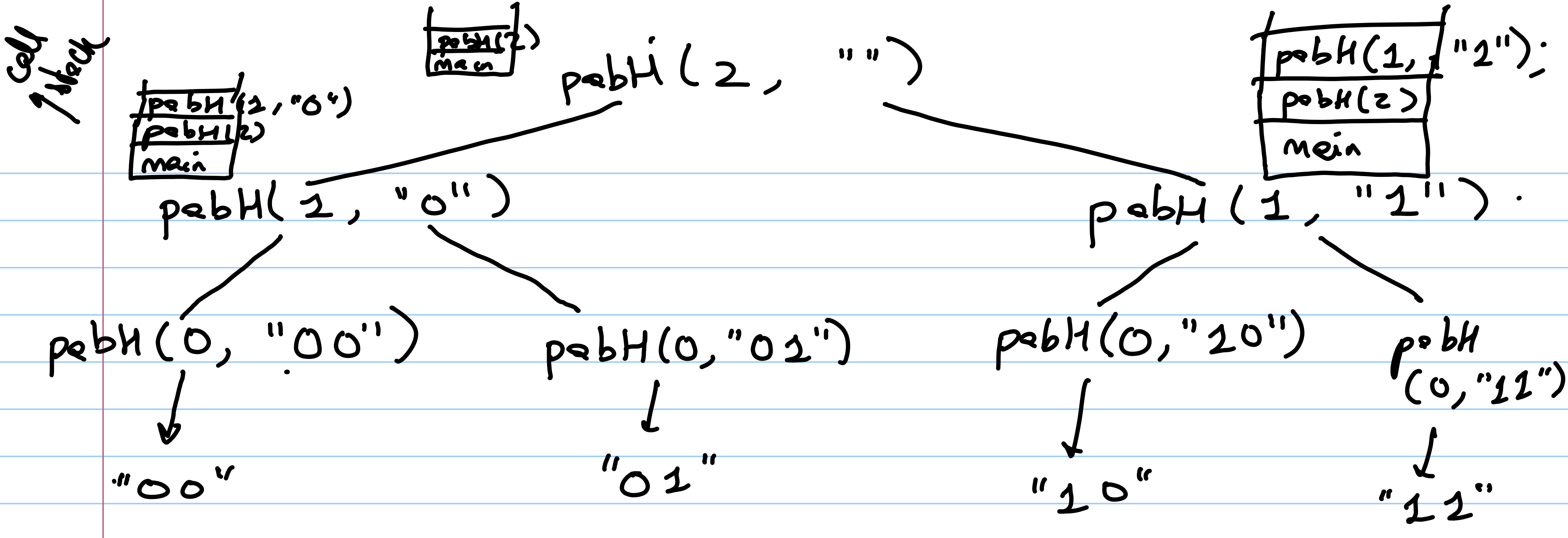


```
void    printAllBinaryHelper (int  digits,
                                   string   sofar)
{
        if   ( digits == 0)  {
                  cout << sofar << endl;
        } else {
              printAllBinaryHelper (digits - 1,
                                        sofar + "0");
              printAllBinaryHelper (digits - 1,
                                        sofar + "1");
        }
}
void   printAllBinary (int   digits)
{
   printAllBinaryHelper (digits,   "");
}
```

copy
because
pass-by-value

don't
need
un-choosing

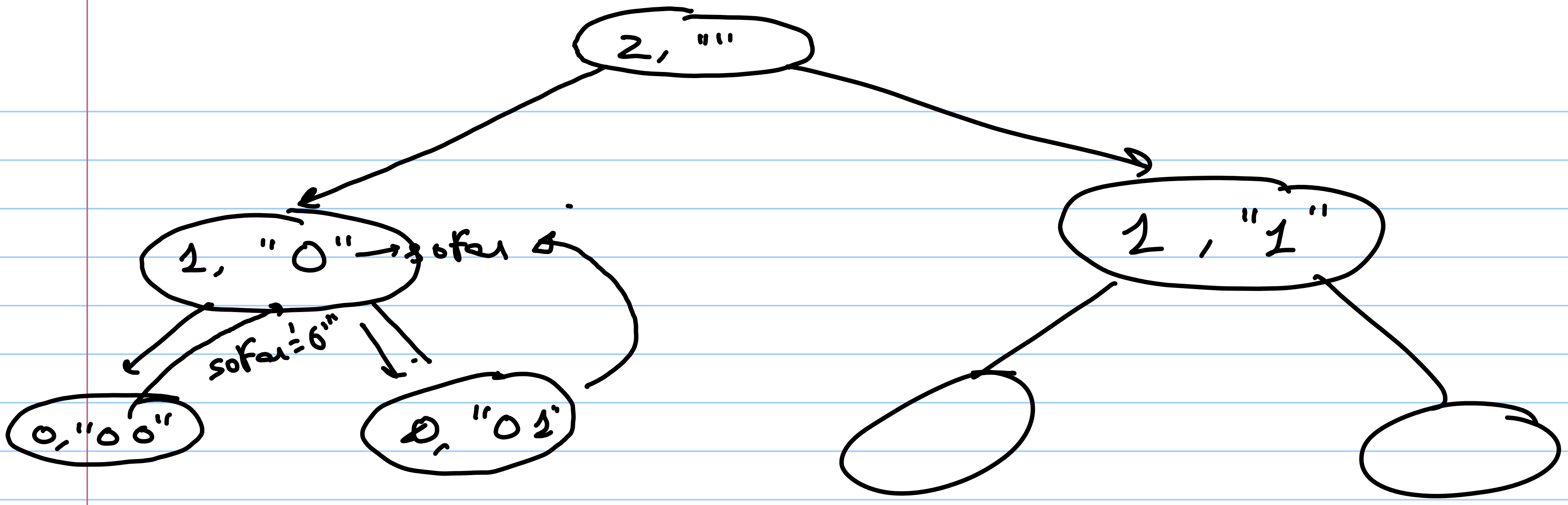
empty
string



call tree (tree of decisions)

pass-by-reference

```
void    printAllBinaryHelper ( int digits ,
                               string & sofar)
{
        if    (digits == 0)  {
                  cout <<  sofar << endl ;
        } else   {
              sofar = sofar + "0";
              printAllBinaryHelper(digits-1, sofar);
              sofar = sofar.substr(0, sofar.length()
                                            -1);
              sofer = sofar + "1";
              printAllBinaryHelper(digits -1,
                                         sofar);
               sofar = sofar . substr(0, sofar.length()
                                            -1);
        }
}
```


2, ""
1, "0"
sofar
sofar = "0"
1 , "1"
0, "00"
0, "01"