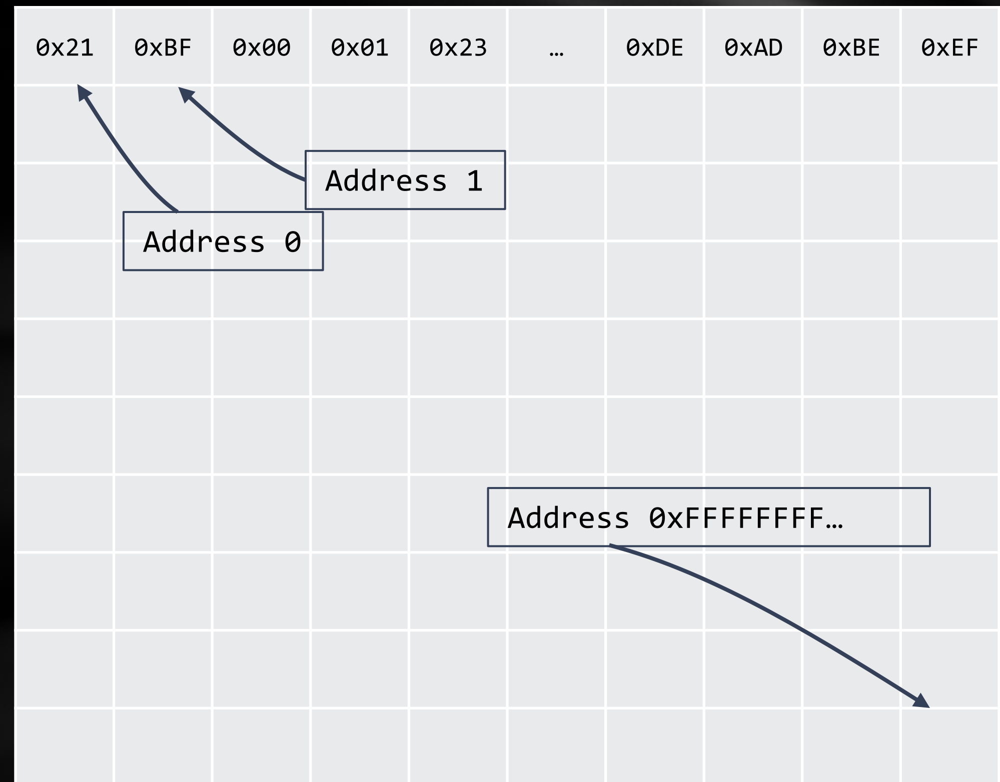
Data Structures and Algorithms

Lecture 3 slides

To understand recursion, you must first understand recursion.

C Language Review - Memory

- *The memory is a very large array of bytes*
- *It is made up of cells, or locations of one byte each (the memory is byte-addressed)*
- *Each location has an address*
- *The number of addresses depends on the size of the memory*



C Language Review - Pointers

- *Pointers are variables that don't hold values, but addresses;*
- *The operator used to obtain the address of a variable is "&";*
- *The operator used to read the data from an address is "*"; NOTE: This is different from the operator used to define a pointer;*

```
char * c;
c = (char*)16;
char d = *c;
// d = ?
```

0	1	2	3	4	5	6	7	8	9
0x21	0xBF	0x00	0x01	0x23	…	0xDE	0xAD	0xBE	0xEF
10	11	12	13	14	15	16	17	18	19
0x10	0x83	0x11	0xA9	0x50	0x1F	0xFF	0x4D	0xCC	0x1A
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

C Language Review – Pointers (cont'd)

- *If a pointer is declared for a data type larger than one byte, subsequent bytes are considered to belong to the same variable*
- *Little endian convention: **least significant byte goes first**!*

```
int *pInt = (int*)6;
int myInt = *pInt;
// myInt == ?
```

0	1	2	3	4	5	6	7	8	9
0x21	0xBF	0x00	0x01	0x23	…	0xEF	0xBE	0xAD	0xDE
10	11	12	13	14	15	16	17	18	19
0x10	0x83	0x11	0xA9	0x50	0x1F	0xFF	0x4D	0xCC	0x1A
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

C Language Review – Pointers (cont'd)

Pointer arithmetic: adding a constant integer to a pointer will actually increment the address with that value multiplied with the size of the data type, in bytes.

```
short  *pShort = (short*)10;
pShort++;
printf("%d", (long)pShort); //will get 12
```

```
float *pFloat = (float*)16;
pFloat++;
printf("%d", (long)pFloat); //will get 20
```

Using the "[]" operator is always equivalent with using pointer arithmetic:

```
int *pInt = (int*)20;
// pInt[5] === *(pInt + 5)
```

C Language Review – Pointers (cont'd)

Practice:

```
*((short*)4 + 2)

*((char*)((int*)((char*)10 – 2) – 1))

*((char*)(*((char*)((int*)0 + 1) - 1)))
```

0	1	2	3	4	5	6	7	8	9
0x21	0xBF	0x00	0x01	0x23	…	0xEF	0xBE	0xAD	0xDE
10	11	12	13	14	15	16	17	18	19
0x10	0x83	0x11	0xA9	0x50	0x1F	0xFF	0x4D	0xCC	0x1A
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

Algorithms and Complexity

- An algorithm is series of steps required to be performed in order to achieve a goal.
- A software or computer algorithm is a series of operations performed in order to process a set of input data and obtain a required set of output data.
- In most cases, there is more than one way to solve a problem => there is more than one possible algorithm.
- The quality of an algorithm depends on how fast it runs on a set of input data, how well it scales with the increase of the size of the input data, and how many computer resources it uses during runtime.

Algorithms and Complexity – Big O Notation

- The Big O notation is used to represent the way an algorithm scales with the increase of input data; E.g.: a search algorithm in an random content array is always O(n), meaning that the algorithm scales linearly with the number of elements in the array.
- You can think of *n* as the number of operations required to solve the problem.

Examples:

<table>
<tr><th>O(n) complexity</th><th>O(n²) complexity</th><th>O(√n) complexity</th></tr>
<tr>
<td><pre>for(i=0; i<n; i++){
    sum += v[i];
}</pre></td>
<td><pre>for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            sum += v[i][j];
        }
}</pre></td>
<td><pre>int isPrime(int n){
    int i;
    if(n == 1) return 0;
    for(i=2; i<=sqrt(n); i++){
        if(n % i == 0) return 0;
    }
    return 1;
}</pre></td>
</tr>
</table>

Algorithms and Complexity – Big O Notation (cont'd)

In theory, the constant before the *n* value in big O notation doesn't matter: $O(kn) = O(n)$

In practice, it matters, and sometimes it matters a lot; There are theoretical algorithms that are of lower complexity, but because they have such a large constant in front of the *n*, they require huge values for n in order to be efficient.

Example: Which is faster? $O(10n)$ or $O(n^2)$?

For n = 2?
How about n = 3?
How about n = 10?
And how about n = 12?

Recursion

- Just like in math, recursion is the definition of a function through itself;
- Just like in math, a stop condition is a mandatory requirement, or the stack will overflow (remember lecture 2).

<table>
<tr><th>Formal definition</th><th>C definition</th></tr>
<tr><td>$f: N^* \rightarrow N^*$
$f(x) = f(x-1) + 1$
$f(1) = 1$</td><td><pre>unsigned f(unsigned n){
    if(n == 1) return 1;
    return f(n-1) + 1;
}</pre></td></tr>
</table>

Recursion - Example

1. Compute the factorial of a number

```
unsigned long long factorial(int n){
    if(n == 1) return 1;
    return factorial(n – 1) * n;
}
```

1. Find the maximum in an array

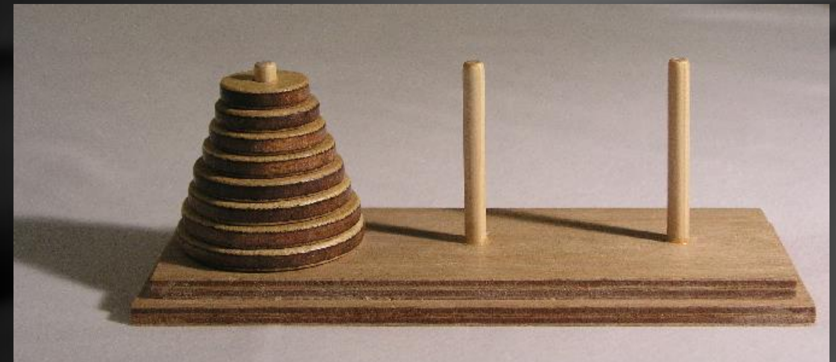
```
int max(int* v, int start, int stop){
    if(start == stop) return v[start];
    int maxFirstHalf = max(v, start, (start + stop) / 2);
    int maxSecondHalf = max(v, (start + stop) / 2 + 1, stop);
    return maxFirstHalf > maxSecondHalf ? maxFirstHalf : maxSecondHalf;
}
```

Recursion –Divide et Impera

- Divide et Impera (*divide and conquer* in Latin), is a programming technique that tries to solve a problem by splitting it into smaller problems of the same type.
- Most recursive functions are particular divide-et-impera implementation.
- Example of divide-et-impera algorithm, Tower of Hanoi:

Having three rods and a number of discs placed in ascending order on the first rod (like the image on the right), the objective is to move all discs to the third rod, obeying three simple rules:

1. *Only one disk can be moved at a time.*
2. *Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.*
3. *No disk may be placed on top of a smaller disk.*



Thank you!

* For next lecture, required notions are: arrays and *for* statements