CSC231

Week 11 — Introduction to C
Fall 2019

Dominique Thiébaut
dthiebaut@smith.edu
Outline

• Recursion: Towers of Hanoi
• C
Towers of Hanoi... in Assembly

- In Python first
- In Assembly next

https://media-cdn.tripadvisor.com/media/photo-s/0f/00/ee/18/ulun-danu-bratan-temple.jpg
What's the relationship between Towers of Hanoi and Serpinski?

https://youtu.be/2SUvWfNJSsM
Write it in Python!

```python
# towers.py
# D. Thiebaut
# CSC231

def moveDisks(n, source, dest, extra):

    def main():
        n = int(input("> "))
        moveDisks(n, 'A', 'B', 'C')

    main()
```

getcopy hanoi0.py
moveDisk( 5, Source, Dest, Extra )

moveDisk( 4, Source, Extra, Dest )

moveDisk( 1, Source, Dest, Extra )

moveDisk( 4, Extra, Dest, Source )
moveDisk( 5, Source, Dest, Extra )
moveDisk( 4, Source, Extra, Dest )
moveDisk( 1, Source, Dest, Extra )
moveDisk( 4, Extra, Dest, Source )
moveDisk( 5, Source, Dest, Extra )

moveDisk( 4, Source, Extra, Dest )

moveDisk( 1, Source, Dest, Extra )

moveDisk( 4, Extra, Dest, Source )
section .data

prompt db "> ", 0

section .text

extern _getInput
extern _println
extern _printlnString

global _start

_start:

;;; get N from user
mov ecx, prompt
mov edx, 2
call _printlnString

call _getInput

;;; define the 3 pegs and pass them in bl, cl, and dl:
mov bl, 'A'
mov cl, 'B'
mov dl, 'C'

;;; moveDisks( N, 'A', 'B', 'C' )
call moveDisks

;;; exit
mov ebx, 0
mov eax, 1
int 0x80

moveDisks:
pushad

;;; if n=1:

;;; print( source, dest )
cmp eax, 1
jg recurse
mov al, bl
; print source
call printChar
mov al, ','
; print space
call printChar
mov al, cl
; print dest
call printChar
call _println
; print 

popad
; done! return
ret

recurse:

;;; moveDisks( n-1, source, temp, dest )
dec eax
xchg cl, dl
; swap cl & dl
call moveDisks
xchg cl, dl
; swap them back

;;; print( source, dest )
mov al, bl
; print source
call printChar
mov al, ','
; print space
call printChar
mov al, cl
; print dest
call printChar
call _println
; print 

;;; moveDisks( n-1, temp, dest, source )
popad
pushad
xchg bl, dl
dec eax
call moveDisks
popad
; makes sense, but not needed
ret

getcopy hanoi.asm
;; printChar: prints the character in al
;; does not modify any other register

section .data
    db 'A'
section .text

printChar:
    pushad
    mov byte[char],al
    mov ecx, char
    mov edx, 1
    call _printString
    popad
    ret
C for Assembly Language Programmers
- Dennis Ritchie
- 1969 to 1973
- AT&T Bell Labs
- Close to Assembly
- Unix
- Standard
- Many languages based on C. (C++, Obj. C, C#)
- Many influenced by C (Java, Python, Perl)
Dennis Ritchie
1969 to 1973
AT&T Bell Labs
Close to Assembly
Unix
Standard
Many languages based on C. (C++, Obj. C, C#)
Many influenced by C (Java, Python, Perl)
Missing from C...

- Exceptions
- Garbage collection
- OOP
- Polymorphism
- But...
Missing from C…

- Exceptions
- Garbage collection
- OOP
- Polymorphism
- But… it is **usually faster!**
Example: N-Queens Problem

http://www.science.smith.edu/dftwiki/index.php/CSC231:_N-Queens_Problem:_interpreted_vs_compiled
Example: N-Queens Problem

http://www.science.smith.edu/dftwiki/index.php/CSC231:_N-Queens_Problem:_interpreted_vs_compiled

<table>
<thead>
<tr>
<th>Language</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>48 sec</td>
</tr>
<tr>
<td>C</td>
<td>1.4 sec</td>
</tr>
<tr>
<td>C with -O3 optimization</td>
<td>0.4 sec</td>
</tr>
</tbody>
</table>
Good Reference

• Essential C, by Nick Parlante, Stanford U.  
  http://cslibrary.stanford.edu/101/EssentialC.pdf
(simplification)
Hello World!

```c
#include <stdio.h>

int main() {
    printf("\nHello World\n");
    return 0;
}
```
Hello World!

- Library
- Strings
- Block-structured language
- main()

```c
#include <stdio.h>

int main() {
    printf("\nHello World\n");
    return 0;
}
```
Hello World!

- Library
- Strings
- Block-structured language
- main()

```c
#include <stdio.h>

int main() {
    printf("\nHello World\n");
    return 0;
}
```
• `gcc` Gnu compiler
• `man gcc` for help

```
[~/handout]$ gcc hello.c
[~/handout]$ ./a.out
Hello World
```

```
[~/handout]$ gcc -o hello hello.c
[~/handout]$ ./hello
Hello World
```
Files

```
[cs231a@marax:~/handout/C$ ls -ltr | tail -n 3
-rw-r--r-- 1 cs231a cs231a 22 Nov 18 09:46 hello.txt
-rw-r--r-- 1 cs231a cs231a 66 Nov 18 09:46 hello.c
-rwxrwxr-x 1 cs231a cs231a 8600 Nov 18 09:46 hello
]
```
• Write your own Hello World! program

• Make it print something like:

```
************
* C Rocks! *
************
```

```
[~/handout]$ gcc -o hello hello.c
[~/handout]$ ./hello
Hello World!
```
The GNU C Reference Manual

Table of Contents

• The GNU C Reference Manual

• Preface

• Credits

• Introduction

• Lexical Elements

• Identifiers

• Keywords

• Constants

• Character Constants

• Real Number Constants

• String Constants

• Operators

• Separators

• White Space

• Data Types

• Integer Types

• Real Number Types

• Complex Number Types

• File Handling

• Memory Management

• Structures

• Pointers

• Functions

• Dynamic Memory Allocation

• Arrays

• Strings

• Errors

• Bugs

Good Reference on C Compiler

• `printf( "string with %-operators", list of vars);`
  • `%d` int
  • `%f` float
  • `%s` string

`printf( "%3d %1.3f %s\n", 2, 3.14159, "CSC231 Assembly" );`
Variables
Variables

- Simple types
- No string type!
- No booleans (only 0 for \textit{false} and \texttt{!0} for \textit{true})
- No classes, no objects, no OOP!

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer variable</td>
</tr>
<tr>
<td>short</td>
<td>short integer</td>
</tr>
<tr>
<td>long</td>
<td>long integer</td>
</tr>
<tr>
<td>float</td>
<td>single precision real (floating point) variable</td>
</tr>
<tr>
<td>double</td>
<td>double precision real (floating point) variable</td>
</tr>
<tr>
<td>char</td>
<td>character variable (single byte)</td>
</tr>
</tbody>
</table>
// sizeOfSimpleTypes.c
// D. Thiebaut
// Prints the size of various types for the gcc compiler
// on the current platform.

#include <stdio.h>

void main() {

    printf(" short int is %2d bytes \n", (int) sizeof(short int));
    printf(" int is %2d bytes \n", (int) sizeof(int));
    printf(" long int is %2d bytes \n", (int) sizeof(long int));
    printf(" signed int is %2d bytes \n", (int) sizeof(signed int));
    printf(" unsigned int is %2d bytes \n", (int) sizeof(unsigned int));
    printf("\n");
    printf(" float is %2d bytes \n", (int) sizeof(float));
    printf(" double is %2d bytes \n", (int) sizeof(double));
    printf(" long double is %2d bytes \n", (int) sizeof(long double));
    printf("\n");
    printf(" signed char is %2d bytes \n", (int) sizeof(signed char));
    printf(" char is %2d bytes \n", (int) sizeof(char));
    printf(" unsigned char is %2d bytes \n", (int) sizeof(unsigned char));

}
cs231a@marax:~/handout/C$ gcc sizeOfSimpleTypes.c

cs231a@marax:~/handout/C$ ./a.out

    short int is  2 byte(s)
        int is  4 byte(s)
    long int is  8 byte(s)
    signed int is  4 byte(s)
    unsigned int is  4 byte(s)

    float is  4 byte(s)
    double is  8 byte(s)
    long double is 16 byte(s)

    signed char is  1 byte(s)
        char is  1 byte(s)
    unsigned char is  1 byte(s)

cs231a@marax:~/handout/C$
/* hello.c
   D. Thiebaut
   Hello World! in C
*/
#include <stdio.h>

int main() {
   // this line is a comment
   printf("\nHello World!\n");

   return 0;
}
We stopped here

Last Time
Strings
(and String Functions)

Use "…" for strings
Use 'x' for characters
```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    return 0;
}
```
```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    strcpy( sentence, hello ); // sentence <- "hello"

    return 0;
}
```
Strings

```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    strcpy( sentence, hello ); // sentence <- "hello"
    strcat( sentence, " " );    // sentence <- "hello "

    return 0;
}
```
```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    strcpy( sentence, hello );  // sentence <- "hello"
    strcat( sentence, " " );   // sentence <- "hello "
    strcat( sentence, world ); // sentence <- "hello world!"

    return 0;
}
```
```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    strcpy( sentence, hello );       // sentence <- "hello"
    strcat( sentence, " " );         // sentence <- "hello "
    strcat( sentence, world );       // sentence <- "hello world!"

    printf( "sentence = %s\n", sentence );

    return 0;
}
```
Strings

```c
#include <stdio.h>
#include <string.h>

int main() {
    char hello[] = "hello";
    char world[] = "world!";
    char sentence[100] = "";

    strcpy( sentence, hello );  // sentence <- "hello"
    strcat( sentence, " " );    // sentence <- "hello 
    strcat( sentence, world );  // sentence <- "hello world!"

    printf( "sentence = %s\n", sentence );

    return 0;
}
```

[~/handout/C]$ gcc strings.c
[~/handout/C]$ ./a.out

sentence = hello world!
[~/handout/C]
A Few String Functions

\textbf{strcpy( dest, src )}
\begin{itemize}
  \item \textit{copies the string pointed to by src to dest}
\end{itemize}

\textbf{strcat( dest, src )}
\begin{itemize}
  \item \textit{concatenate the string pointed to by src to dest}
\end{itemize}

\textbf{strlen( src )}
\begin{itemize}
  \item \textit{returns the number of characters in the string}
\end{itemize}

more at \url{https://www.tutorialspoint.com/c_standard_library/string_h.htm}
A Few String Functions

`strstr(haystack, needle)`

returns a pointer to the needle (string) in the stack (string), or `NULL` if not found.

`strchr(haystack, char-needle)`

returns a pointer to the needle (char) in the stack (string), or `NULL` if not found.

more at https://www.tutorialspoint.com/c_standard_library/string_h.htm
Strings end with ‘\0’

msg

'h' 'e' 'l' 'l' 'o' '\0'
Strings end with `\0`

- msg
  - 'h' 'e' 'l' 'l' 'o' \0

- msg2
  - 'h' 'e' 'l' 'l' 'o' ' ' 't' 'h' 'e' 'r' 'e' \0
Strings end with ‘\0’

msg

' h  \
e  \
'  \
'  \
o  \
\0

msg2

' h  \
e  \
'  \
'  \
o  \\
'  \\
t  \
h  e  \
'  \
r  e  \
\0

msg3

' h  \
e  \
'  \
'  \
o  \\
\0  \\
t  \
h  e  \
'  \
r  e  \
\0
Strings end with `'\0'`

- msg: 
  {'h', 'e', 'l', 'l', 'o', '\0'}

- msg2: 
  {'h', 'e', 'l', 'l', 'o', ' ', 't', 'h', 'e', 'r', 'e', '\0'}

- msg3: 
  {'h', 'e', 'l', 'l', 'o', '\0', 't', 'h', 'e', 'r', 'e', '\0'}

- msg4: 
  {'\0', 'e', 'l', 'l', 'o', '\0', 't', 'h', 'e', 'r', 'e', '\0'}
Strings end with `\0`
Strings end with ‘\0’

```c
#include <stdio.h>
#include <string.h>

int main() {
    char sentence[100] = "Hello world!"
    printf( "sentence = %s\n", sentence );
    sentence[5] = '\0';
    printf( "sentence = %s\n", sentence );

    return 0;
}
```

~/handout/C]$ ./a.out
sentence = Hello world!
sentence = Hello
[~/handout/C]$
Strings end with `\0`

```c
#include <stdio.h>
#include <string.h>

int main() {
    char sentence[100] = "Hello world!";
    printf("sentence = %s\n", sentence);
    sentence[5] = '\0';
    printf("sentence = %s\n", sentence);
    return 0;
}
```

~/handout/C]$ ./a.out
sentence = Hello world!
sentence = Hello
[~/handout/C]$
• make the program compute your email account as the first letter of your first name and the first 8 letters of your last name.
For-Loops
For-Loops

```c
#include <stdio.h>

int main() {
    int sum = 0;

    // compute the sum of all the numbers from 1 to 100
    for (int i=1; i<=100; i++) {
        sum += i;
    }
    printf("\nsum = %d\n\n", sum);
    return 0;
}
```
```c
#include <stdio.h>

int main() {
    int i;
    int sum = 0;

    // compute the sum of all the numbers from 1 to 100
    i = 1;
    while ( i <= 100 ) {
        sum += i; // could have also used i++
        i += 1;
    }

    printf( "\nsum = %d\n\n", sum );
    return 0;
}
```

For information about GNU Emacs and the GNU system, type C-h C-a.
Infinite Loops

```c
#include <stdio.h>

int main() {
    // infinite loop #1
    while ( 1 ) {
        printf( "hello!\n" );
    }

    // infinite loop #2
    for ( ; ; ) {
        printf( "hello!\n" );
    }

    return 0;
}
```
• Write a program that displays the first 10 \textit{fibonacci} terms. Use a loop!

• You may use this code to get started. You can start printing 1, 2, 3 or 1, 1, 2, 3... Both solutions are fine.

\begin{verbatim}
int fibn=1, fibn1=1, fibn2=0;
fibn = fibn1 + fibn2;
...
printf( "\%d\n", fibn );
\end{verbatim}
• Take the solution for the previous exercise (getcopy C/fibonacci0.c) and replace the for-loop with a **while loop** that prints all the terms that less than 1000.
Symbolic Constants
Symbolic Constants

```c
// symConstants.c
// D. Thiebaut
#include <stdio.h>

#define NAME     "Mickey"
#define HEIGHT   5
#define YEARBORN 1928

int main() {
    printf("%s is %d inches high, and was created in %d\n\n", NAME, HEIGHT, YEARBORN);

    return 0;
}
```
Symbolic Constants

After preprocessing

```c
#include <stdio.h>
#define NAME "Mickey"
#define HEIGHT 5
#define YEARBORN 1928

int main() {
    printf("%s is %d inches high, and was created in %d\n\n", "Mickey", 5, 1928);
    return 0;
}
```
// conditionals.c
#include <stdio.h>

int main() {
    int a = 5;
    int b = 3;
    int c = 7;

    if ( a <= b && a <= c )
        printf( "%d is the smallest\n\n", a );
    else if ( b <= a && b <= c )
        printf( "%d is the smallest\n\n", b );
    else
        printf( "%d is the smallest\n\n", c );

    /*--F1 condicionals.c Top L1 (C/l Abbrev)-----------------------------*/
### Conditionals

<table>
<thead>
<tr>
<th>&amp;&amp;</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not</td>
</tr>
</tbody>
</table>
Conditionals (cont’d)

```
switch ( ordinal_expression ) {
  case ordinal_value: {
    // ...
    break;
  }
  case ordinal_value: {
    // ...
    break;
  }
  default: {
    // ...
  }
}
```
Conditionals (cont’d)

```c
switch ( ordinal_expression ) {
    case ordinal_value: {
        // ...
        break;
    }
    case ordinal_value: {
        // ...
        break;
    }
    default: {
        // ...
    }
}
```

ints or chars, something countable
Pointers
float x = 6.5;
float *px = &x;
Example: Initialize an Array

Using indexing
Example: Initialize an Array

```c
#include <stdio.h>
#define SIZE 10

int main() {
    float A[SIZE];
    int i;

    for ( i=0; i<SIZE; i++ )
        A[i] = i;

    for ( i=0; i<SIZE; i++ )
        printf( "A[%d] = %.2f\n", i, A[i]);

    return 0;
}
```
Example: Initialize an Array

Using pointers

A

\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
3 & 4 & 1 & 9 & 3 & 4 & 6 & 6 & 1 & 0
\end{array}
Example: Initialize an Array

```c
// initArray1.c
// D. Thiebaut
#include <stdio.h>
#define SIZE 10

int main() {
    float A[SIZE];
    float* p;
    int i;

    p = A;
    for ( i=0; i<SIZE; i++ ) {
        *p = i;
        p++;
    }

    p = A;
    for ( i=0; i<SIZE; i++ ) {
        printf( "p=%p A[%d] = %1.2f *p = %1.2f\n",
                p, i, A[i], *p );
        p = p + 1;
    }

    return 0;
}
```


```bash
./a.out
p=0x7fff88d54560 A[0] = 0.00 *p = 0.00
p=0x7fff88d54564 A[1] = 1.00 *p = 1.00
p=0x7fff88d54568 A[2] = 2.00 *p = 2.00
p=0x7fff88d5456c A[3] = 3.00 *p = 3.00
p=0x7fff88d54570 A[4] = 4.00 *p = 4.00
p=0x7fff88d54574 A[5] = 5.00 *p = 5.00
p=0x7fff88d54578 A[6] = 6.00 *p = 6.00
p=0x7fff88d5457c A[7] = 7.00 *p = 7.00
p=0x7fff88d54580 A[8] = 8.00 *p = 8.00
p=0x7fff88d54584 A[9] = 9.00 *p = 9.00
```
Arrays

Arrays are a fundamental data structure in computer science. They allow for the storage and manipulation of a collection of elements of the same data type.

In C, arrays are defined using the `TYPE` keyword to declare the type of the array elements and the array name.

```c
TYPE v[DIM];
TYPE* pv;

pv = v;
```

In the diagram, we see the array `v` with its elements `v[0]`, `v[1]`, `v[2]`, and `v[3]` each occupying a specific memory location starting from `0FFF1F00`. The elements are accessed using the pointer `pv`. The diagram illustrates how the memory addresses increment by 4 for each subsequent element, consistent with the `sizeof(TYPE)` constraint.

For example, `pv[0]` points to the first element `v[0]`, `pv[1]` points to `v[1]`, and so on. The values in the diagram are illustrative and do not represent actual memory addresses or values stored in these locations.

The diagram also shows that array elements are stored sequentially in memory, with each element's address being the previous address plus a fixed offset determined by the `sizeof(TYPE)`.
Arrays

- The name of an array is a pointer to the first cell of the array.

```c
char name[DIM];
```

- `name` is the same as `&(name[0])`
* and &

- * has two meanings, depending on context
  - “Pointer to”
  - “Contents of”

- & means “the address of”
* and &

```c
int A[DIM];
int* p = A;   // "int pointer p"
int *q = &A[0];  // "int pointer q"
*p = 3;       // what p is pointing to gets 3
*(q+1) = 5;   // what q is pointing to gets 5
```
• Write a C program that copies Array A into Array B using indexing first, and then using pointers.

```c
#include <stdio.h>

#define DIM 10

int main() {
    int A[DIM];
    int B[DIM];

    // init array A
    for ( int i=0; i<DIM; i++ ) {
        A[i] = 13*i % 11;
        printf( "A[%d] = %d\n", i, A[i] );
    }
    // copy A into B (put code here!)

    // print array B
    for ( int i=0; i<DIM; i++ )
        printf( "B[%d] = %d\n", i, B[i] );

    return 0;
}
```

• Write a C program that finds the largest integer in Array A, using pointers.

```c
#include <stdio.h>

#define DIM 10
int main() {
    int A[DIM];

    for ( int i=0; i<DIM; i++ ) {
        A[i] = 13*i % 11;
        printf( "%d " , A[i] );
    }
    printf( "\n" );

    // put your code here

    printf( "max = %d\n" , *max );
    return 0;
}
```
Functions

- Functions are *normally* declared before they are used.
- Functions can return values of simple types (int, char, floats), and even pointers.
- Functions get parameters of simple types, and pointers.
- Passing by value is automatic. Passing by reference requires passing a pointer.
#include <stdio.h>

int sum(int a, int b) {
    return a + b;
}

int main() {
    int x = 10;
    int y = 20;
    int z;

    z = sum(x, y);
    printf("z = %d\n", z);

    z = sum(3, 8);
    printf("z = %d\n", z);

    printf("sum(11, 22) = %d\n", sum(11, 22));
    return 0;
}

Example 1

z = 30
z = 11
sum(11, 22) = 33
Example 2

(Incomplete code... Add missing elements!)

```c
#include <stdio.h>

// warning: incomplete code!

void sum2( int a, int b, int c ) {
    c = a + b;
}

int main() {
    int x = 10;
    int y = 20;
    int z;

    sum2( x, y, z );
    printf( "z = %d\n", z );

    sum2( 3, 8, x );
    printf( "x = %d\n", x );

    return 0;
}
```

Pass by Reference!

- `z = 30`
- `x = 11`
Exercise

Complete the code shown here

```c
void decrement( int* A, int N ) {
    int *p =
    for ( ) {

    }
}

int main() {
    int A[DIM];
    // initialize A
    for ( int i=0; i<DIM; i++ )
        A[i] = i*23 % 17;
    display( A, DIM );

    // decrement all values of A
    decrement( A, DIM );
    display( A, DIM );
    return 0;
}
```

getcopy C/decrementArray.c
Exercise

• Write a C program that uses functions to find the largest integer in A. The result is passed back using a return statement.

```c
#include <stdio.h>
#include <stdlib.h>

#define N 10

void main() {
    int A[N] = { 3, 2, 1, 0, 6, 5, 9, 8, 7, -3 };  
    for ( int i=0; i<N; i++ )
        printf( "%d ", A[i] );

    printf( "\nmax of A = %d\n", max1( A ) );
}
```
Exercise

• Write another program that does the same thing but the max is *passed* back via a parameter *passed by reference.*