E85 Digital Design & Computer Engineering

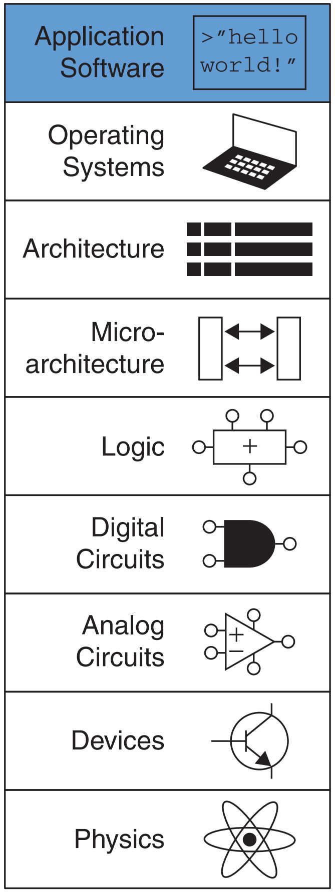
Lecture 12:

C Programming

HARVEY MUDD COLLEGE

Lecture 12

- Overview
- Programming Constructs
  - Comments
  - Constants
  - Variables
  - Primitive Data Types
  - Function Calls
  - Operators
  - Control Flow
  - Loops
  - Arrays and Strings



Overview

- C programming language developed at Bell Labs around 1973
- Capable of controlling a computer to do nearly anything, including directly interacting with the hardware
- Suitable for generating high performance code
- Relatively easy to use
- Available from supercomputers to microcontrollers
- Closely related to other important languages including C++, C#, Objective C, Java, Arduino

C is Libertarian

- Lets you do just about anything
- Interacts directly with the hardware
- Does NOT protect you from your own stupidity
- Assumes YOU know the size of arrays and variables
- Unless sandboxed will write ANYWHERE in memory

Example

```c
// factorial.c
// David_Harris@hmc.edu 22 October 2019

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

void main(void) {
       int result;
       result = fact(4);
}
```

Steps to C Programming

- Write code
- Compile code
- Execute code
- Debug code

Comments

- Single-line comments begin with "//" and continue to the end of the line.

```
x += 2; //This is a single-line comment.
```

- Multi-line comments begin with "/*" end with "*/".

```
/* You can hide or disable a section of
code such as this block with a multi-line
comment
   x = bob ? x : y;
   y -= 5;
*/
```

Constants, Defines, or Macros

- Constants are named using the `#define` directive

```
#define MAXGUESSES 5
#define PI 3.14159
```

- The # indicates that this line in the program will be handled by the preprocessor.
- Before compilation, the preprocessor replaces each occurrence of the identifier MAXGUESSES in the program with 5.
- By convention, `#define` lines are located at the top of the file and identifiers are written in all capital letters.

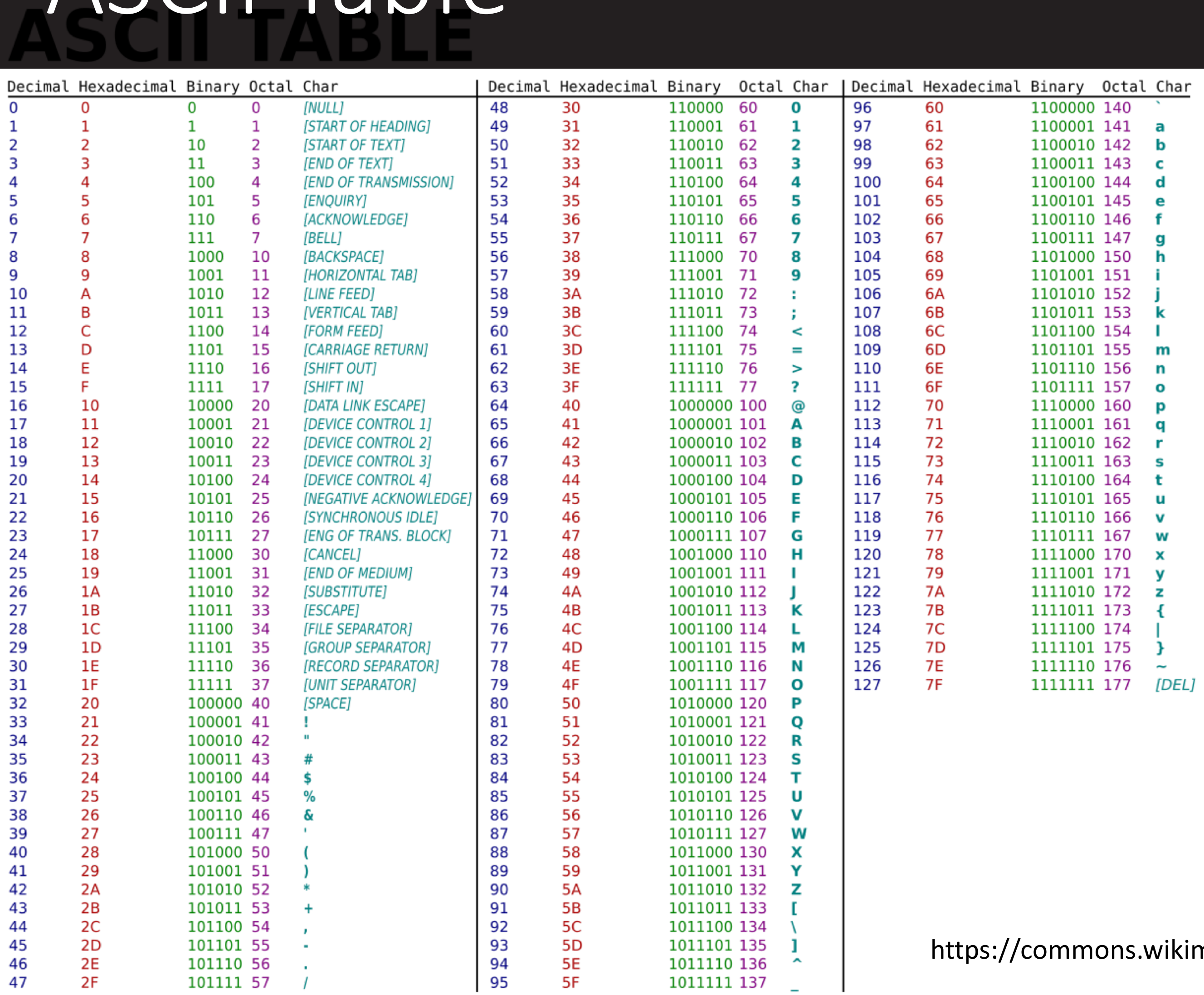
Global and Local Variables

- Global variables often lead to hard-to-debug code and should be avoided
- Global variables are declared outside of any function
- Local variables are declared inside a function
- Local variables should be your go-to type

Primitive Data Types

Type	Size (bits)	Minimum	Maximum
`char`	8	$-2^{-7} = -128$	$2^{7} - 1 = 127$
`unsigned char`	8	0	$2^{8} - 1 = 255$
`short`	16	$-2^{15} = -32{,}768$	$2^{15} - 1 = 32{,}767$
`unsigned short`	16	0	$2^{16} - 1 = 65{,}535$
`long`	32	$-2^{31} = -2{,}147{,}483{,}648$	$2^{31} - 1 = 2{,}147{,}483{,}647$
`unsigned long`	32	0	$2^{32} - 1 = 4{,}294{,}967{,}295$
`long long`	64	-2^{63}	$2^{63} - 1$
`unsigned long`	64	0	$2^{64} - 1$
`int`	machine-dependent		
`unsigned int`	machine-dependent		
`float`	32	$\pm 2^{-126}$	$\pm 2^{127}$
`double`	64	$\pm 2^{-1023}$	$\pm 2^{1022}$

ASCII Table



ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

https://commons.wikimedia.org/wiki/File:ASCII-Table.svg

Functions

- Curly braces {} enclose the body of the function, which may contain zero or more statements
- A function can return (or output) at most one value
- The type of returned value is declared in the function declaration
- The return statement indicates the value that the function should return to its caller
- A function can receive inputs
- The type of the inputs is declared in the function declaration
- Functions pass variables by *value* not *reference*
- A function must be either declared BEFORE it is used or a function prototype declared BEFORE it is used

Function Example

```
// Return the sum of the three input variables

int sum3(int a, int b, int c) {
  int result = a + b + c;
  return result;
}
```

Function Prototypes

```c
// sum3example.c
// David Harris@hmc.edu 22 October 2019

////////////////////////////////
// Prototypes
////////////////////////////////
int sum3(int, int, int); // needed because sum3 is called before declared

////////////////////////////////
// main
////////////////////////////////

void main(void) {
  int answer;
  answer = sum3(6, 7, 8);
}

////////////////////////////////
// other functions
// prototype not needed if thsse were moved before main
////////////////////////////////

int sum3(int a, int b, int c) {
  int result = a + b + c;
  return result;
}
```

Prototypes are Sometimes Unavoidable

```
// Prototypes needed for f1 and/or f2 because they
// can’t both be declared before each other

int f1(int);
int f2(int);

int f1(int n) {
  return f2(n-1) + 1;
}

int f2(int n) {
  return f1(n-1)*2;
}

void main(void) {
  int answer;
  answer = f1(5);
}
```

Includes

- The function prototypes for the standard libraries are included at the top of a file with the `#include` directive:

  e.g., `#include <stdio.h>` or `#include <math.h>`

- Your own function prototypes (or anything else you want to include) is done with quotes instead of brackets for relative or absolute path:

  e.g., `#include "other/myFuncs.h"`

Boolean (True/False) in C

- A variable or expression is considered FALSE if its value is 0
- A variable is considered TRUE if it has any other value
  - 1, 42, and -1 are all TRUE for C
- Logical operators assign FALSE as 0 and TRUE as 1

Operators and Precedence

Category	Operator	Description	Example
Unary	++	post-increment	a++; // a = a+1
	--	post-decrement	x--; // x = x–1

	&	memory address of a variable	x = &y; // x = the memory
			// address of y

	~	bitwise NOT	z = ~a;
	!	Boolean NOT	!x
	-	negation	y = -a;
	++	pre-increment	++a; // a = a+1
	--	pre-decrement	--x; // x = x–1

	(type)	casts a variable to (type)	x = (int)c; // cast c to an
			// int and assign it to x

	sizeof()	size of a variable or type in bytes	long int y;
			x = sizeof(y); // x = 4

Operators Continued

Multiplicative	*	multiplication	y = x * 12;
	/	division	z = 9 / 3; // z = 3
	%	modulo	z = 5 % 2; // z = 1
Additive	+	addition	y = a + 2;
	–	subtraction	y = a - 2;
Bitwise Shift	<<	bitshift left	z = 5 << 2; // z = 0b00010100
	>>	bitshift right	x = 9 >> 3; // x = 0b00000001
Relational	==	equals	y == 2
	!=	not equals	x != 7
	<	less than	y < 12
	>	greater than	val > max
	<=	less than or equal	z <= 2
	>=	greater than or equal	y >= 10

Operators Continued

Table eC.3 Operators listed by decreasing precedence—Cont'd

Category	Operator	Description	Example
Bitwise	&	bitwise AND	y = a & 15;
	^	bitwise XOR	y = 2 ^ 3;
	|	bitwise OR	y = a | b;
Logical	&&	Boolean AND	x && y
	||	Boolean OR	x || y

Ternary	? :	ternary operator	y = x ? a : b; // if x is TRUE,
			// y=a, else y=b

Operators Continued

Assignment	=	assignment	x = 22;	
	+=	addition and assignment	y += 3;	// y = y + 3
	−=	subtraction and assignment	z −= 10;	// z = z - 10
	*=	multiplication and assignment	x *= 4;	// x = x * 4
	/=	division and assignment	y /= 10;	// y = y / 10
	%=	modulo and assignment	x %= 4;	// x = x % 4
	>>=	bitwise right-shift and assignment	x >>= 5;	// x = x>>5
	<<=	bitwise left-shift and assignment	x <<= 2;	// x = x<<2
	&=	bitwise AND and assignment	y &= 15;	// y = y & 15
	|=	bitwise OR and assignment	x |= y;	// x = x | y
	^=	bitwise XOR and assignment	x ^= y;	// x = x ^ y

Control Flow Statements

if

```
if (expression)
  statement;
```

if/else

```
if (expression)
  statement1;
else
  statement2;
```

switch/case

```
switch (variable) {
  case (expression1): statement1; break;
  case (expression2): statement2; break;
  case (expression3): statement3; break;
  default: statement4;
}
```

Don’t forget “break” or “default”

If example

```
if (n <= 1) return 1;
```

Compound Statements

- When a statement has more than one line, enclose it in {}

```
if (answer == 42) {
  ultimateQuesiton = 1;
  hitchhikersGuide = 1;
}
```

If/else example

```
if (n <= 1) return 1;
else        return fact(n-1);
```

Case example

```
switch (state) {
  case (0): if (ta) state = 0; else state = 1; break;
  case (1): state = 2; break;
  case (2): if (tb) state = 2; else state = 3; break;
  case (3): state = 0; break;
  default:  state = 0;
}
```

Loops

while

```
    while (condition)
      statement;
```

do/while

```
    do {
      statement;
    } while (condition);
```

for

```
    for (initialization; condition; loop operation)
      statement;
```

While example

```
int fact(int n) {
  int result = 1;
  while (n > 1) {
    result = result * n; // or write result *= n;
    n = n - 1;           // or write n—
  }
  return result;
}

// Alternative code is shorter but less clear

while (n > 1) result *= n--;
```

Do/while example

```
int fact(int n) {
  int result = 1;
  do {
    result *= n;
  } while (n-- > 1);
  return result;
}
```

- Do always executes the statement at least once.
- Longer and not preferred for this example

For example

```
int fact(int n) {
  int result = 1;
  int i;

  for (i=1; i <= n; i++)
    result *= I;
  return result;
}
```

- First do initialization (I = 1)
- Then check condition (i<=n)
  - If satisfied, do body (result *= i)
  - Then do loop operation (i++)
- Then repeat from checking condition

Data Types: Arrays

- Array contains multiple elements

```
float accel[3];
```

- The elements are numbered from 0 to N–1, where N is the length of the array
- Initialize your arrays.
  - An uninitialized array can contain anything
- Arrays can be multidimensional

```
#define NUMSTUDENTS 120
#define NUMLABS 11
int grades[NUMSTUDENTS][NUMLABS];
```

Array Example

```
#include <math.h>

double mag(double v[3]) {
  return sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
}
```

Data Types: Strings

- A string is an array of characters
- Last entry is zero to indicate end (”NULL terminated”)

```
char name[20] = "BOB";
```

- Stored as:

```
name[0] = 66; // ASCII value for B
name[1] = 79; // ASCII value for O
name[2] = 66; // ASCII value for B
name[3] = 0;   // NULL termination
```

other entries are junk, ignored

Examples: String Handling

```
#define MAXLEN 80

int strlen(char str[]) {
  int len=0;

  while (str[len] && len < MAXLEN) len++;
  return len;
}

void strcpy(char dest[], char src[]) {
  int i = 0;

  do {
    dest[i] = src[i];
  } while (src[i++] && i < MAXLEN);
}
```

Examples: Using Strings

```c
#include <string.h>
#define MAXLEN 80

void main(void) {
  char name[80];
  int len;
  char c;

  strcpy(name, "BOB"); // copy BOB into name
  len = strlen(name);  // len = 3
  c = name[1];         // c = 'O' (79)
}
```