Getting Started in C

ITSC 2181: Introduction to Computer Systems
UNC Charlotte
College of Computing and Informatics

COLLEGE OF COMPUTING
AND INFORMATICS

Why C?

- Developed to build Unix operating system
- Main design considerations:
  - Compiler size: needed to run on PDP-11 with 24KB of memory (Algol60 was too big to fit)
  - Code size: needed to implement the whole OS and applications with little memory
  - Performance
  - Portability
- Little consideration (if any) to the following:
  - Security, robustness, maintainability
  - Legacy Code

Why C? (cont’d)

- Simple to write compiler
  - Programming embedded systems, often only have a C compiler
- Performance
  - Typically 50x faster than interpreted Java
- Smaller, simpler, lots of experience
- One of the most popular programming languages
  - For the latest numbers, see https://www.tiobe.com/tiobe-index/

What’s Your Priority?

Priority	Language Choices
Speed of execution, minimum memory “footprint”	Assembly, C
Safer, easier to develop large (hundreds of files) programs	Java, C++
Easier / faster to code, higher level operations, richer libraries	Python, Ruby, PHP, Perl
Integrate with the web	Web application frameworks, JavaScript

C Strengths

- It’s a **procedural** language (like many others
- It’s **efficient** (binary code size, execution speed)
- **Simple**, **clean** language design
- There is an **international standard**, currently C17
- It has a decent **standard library** of useful functions

COLLEGE OF COMPUTING
AND INFORMATICS

Examples of C or C++

- **Linux**: Assembly, C
- **MS Windows**: Assembly, C, C++
- **Firefox Web Browser**: C++, Javascript
- **GNU Compiler** (GCC): C
- **MySQL**: C, C++
- **Embedded Systems** (cars, appliances, etc.)
- **High performance** (science/engineering) applications

C Weaknesses

- **Little** consideration for **security** or **safety**
- **Less modular** than Java and other OO languages (but C++ fixes that)
- **More programming effort** required than PHP/Python/Perl/Ruby and other scripting languages
- **Not** usually written in C or C++: **web apps, business apps, GUIs, simple utility programs**

Types of Programming Languages

- **Declarative**: focus on what the computer should do
  - *Functional:* Scheme, Haskell
  - *Dataflow*
  - *Logic-* or *constraint-based: Prolog*
  - *Markup languages:* HTML, CSS, subset of SQL
- **Imperative:** focus on how the computer should do something
  - *Procedural* : **C**
  - *Object-oriented* : **Java**

There are no objects in C. Oftentimes this makes programming very different from Java and Python.

COLLEGE OF COMPUTING AND INFORMATICS

Procedural vs. Object-Oriented

- **Procedural**: programming as procedures that modify variables
  - Emphasis on actions that must take place
  - Analogy: following a recipe
- **Object-Oriented**: programming as objects that interact (each with internal state, and methods to manage that state)
  - Emphasis on the state of objects
  - Analogy: operating a car

Getting Started….

Common Platform for This Course

- Different platforms have different conventions for end of line, end of file, tabs, compiler output, …
- Solution (for this class): **compile and run** all programs consistently **on one platform**
- Our common platform will be a Virtual Machine (VM) that runs the Ubuntu Linux operating system.
  - See *Canvas* for more details.

COLLEGE OF COMPUTING AND INFORMATICS

Other Alternatives

- Use **ReplIt (**repl.it)**,** a web-based virtual computing environment (https://replit.com/)
- Use a CCI Lab Computer
- Use Mac OS X (Xcode + developer tools)
- Use MS Windows + **`cygwin`** or Visual Studio
- Use Linux on your PC (dual boot or virtualized)

Note:
The only platform supported by the course staff is the VM that we provide.

COLLEGE OF COMPUTING AND INFORMATICS

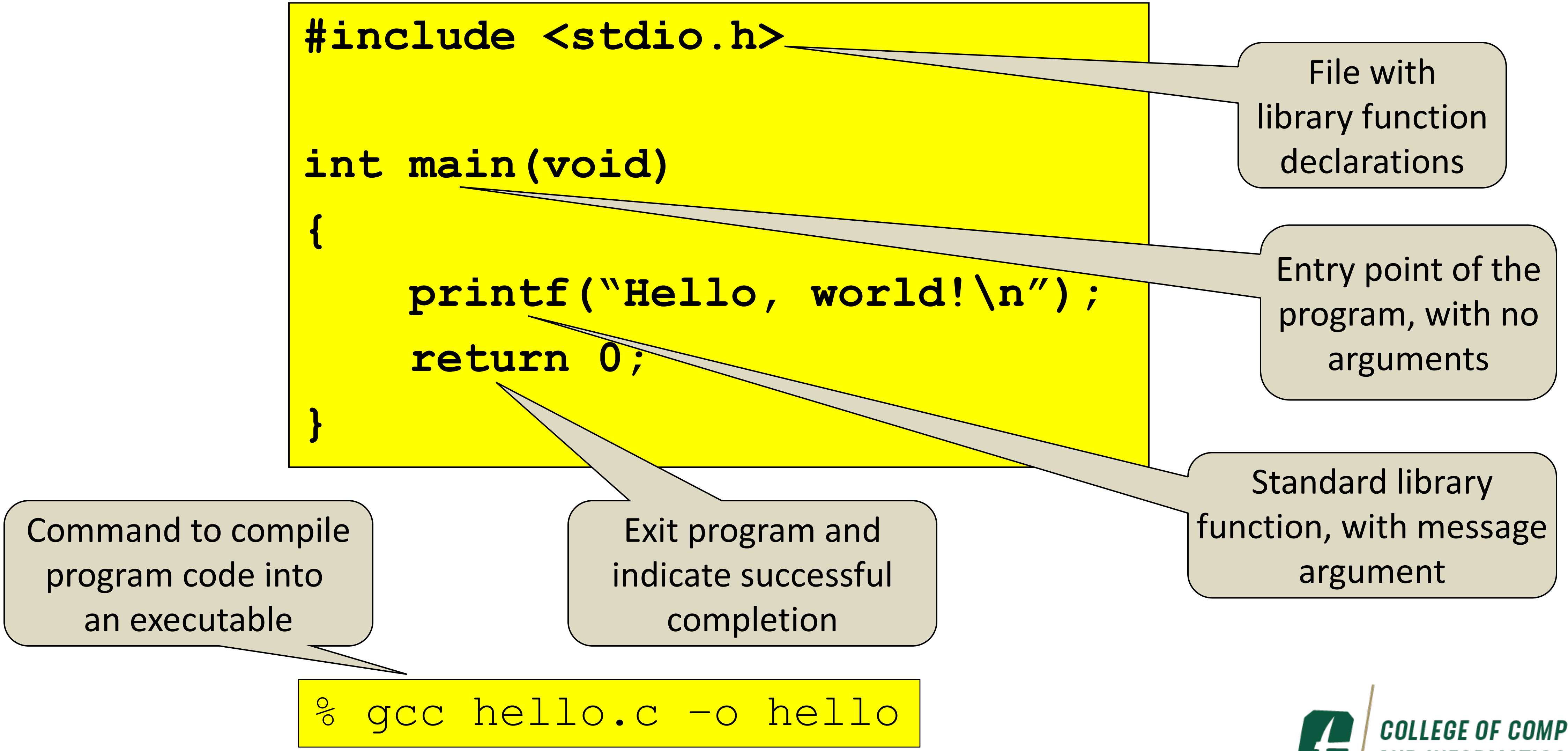
Common Platform Questions

- If you want to develop locally, that’s fine, but you must ensure that it works on the Common Platform
  - You should always test on the Common Platform before submitting
  - The Instructional Assistants will use the common platform to grade your work
  - **No, really, you should test on the Common Platform**

Common Platform Questions

- There are differences between the C compilers for different architectures that may cause your program (that runs locally) to fail on the Common Platform
- C is not architecture neutral!

Your First C Program



```
#include <stdio.h>

int main(void)
{
    printf(“Hello, world!\n”);
    return 0;
}
```

File with library function declarations

Entry point of the program, with no arguments

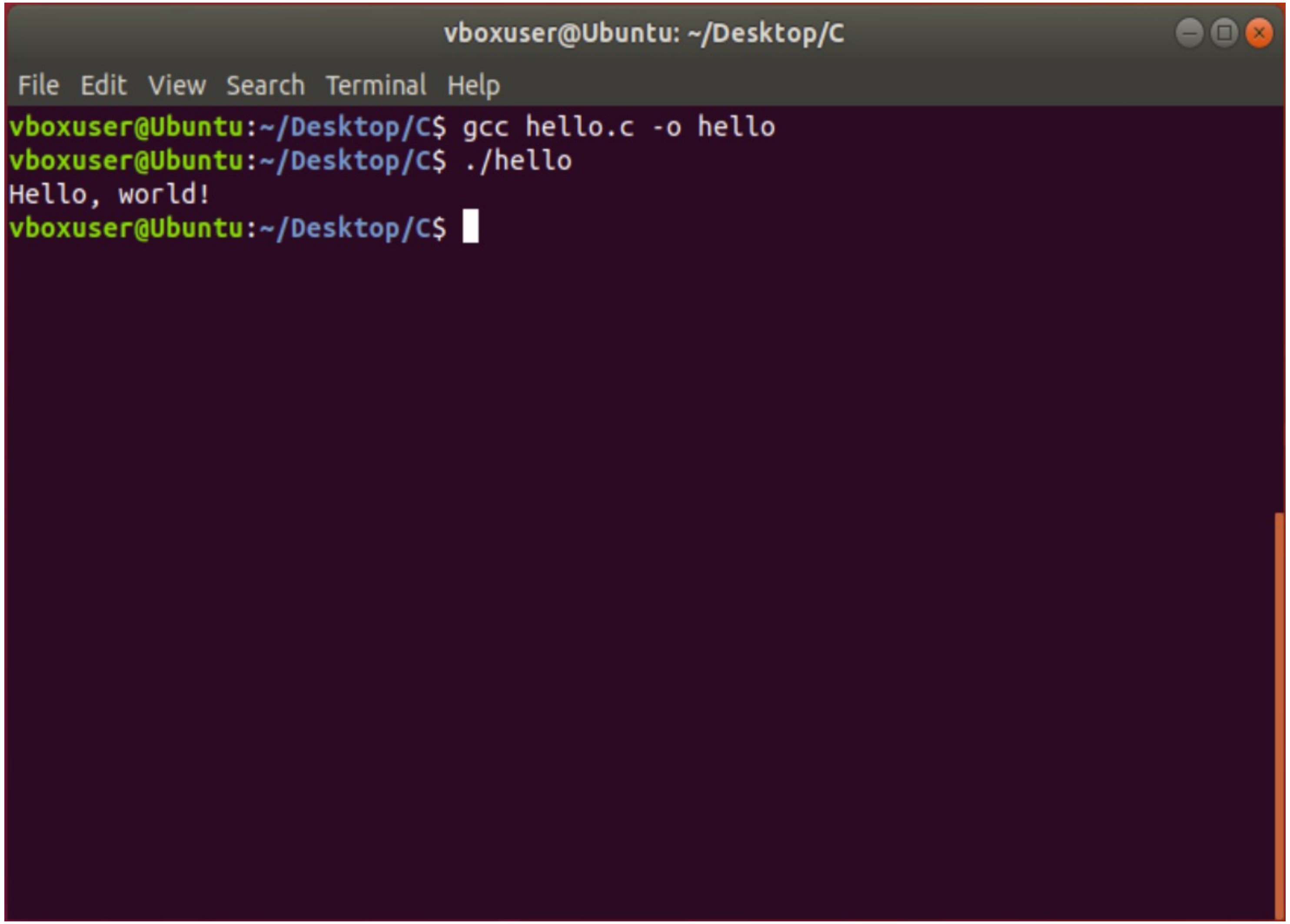
Standard library function, with message argument

Exit program and indicate successful completion

Command to compile program code into an executable

```
% gcc hello.c -o hello
```

Compiling and Running the Program


vboxuser@Ubuntu: ~/Desktop/C
File Edit View Search Terminal Help

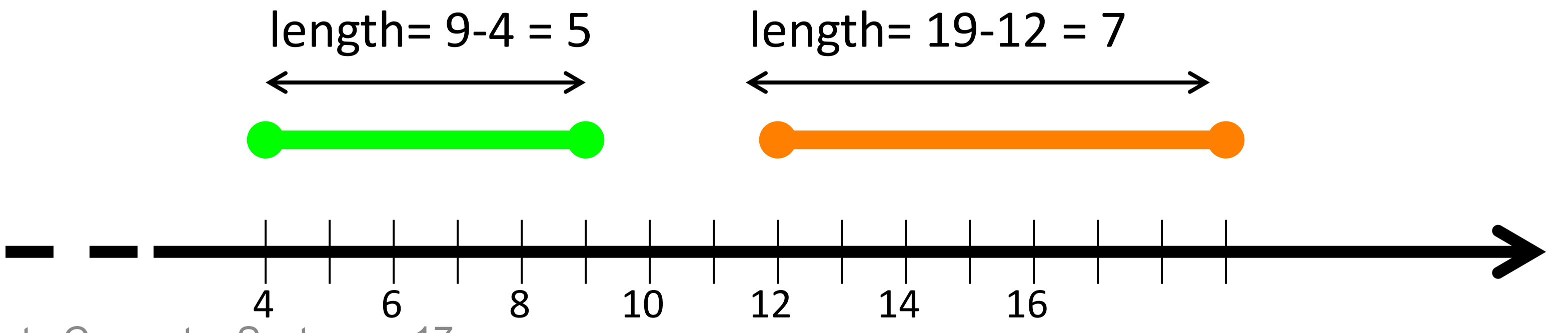
```
vboxuser@Ubuntu:~/Desktop/C$ gcc hello.c -o hello
vboxuser@Ubuntu:~/Desktop/C$ ./hello
Hello, world!
vboxuser@Ubuntu:~/Desktop/C$ 
```

COLLEGE OF COMPUTING AND INFORMATICS

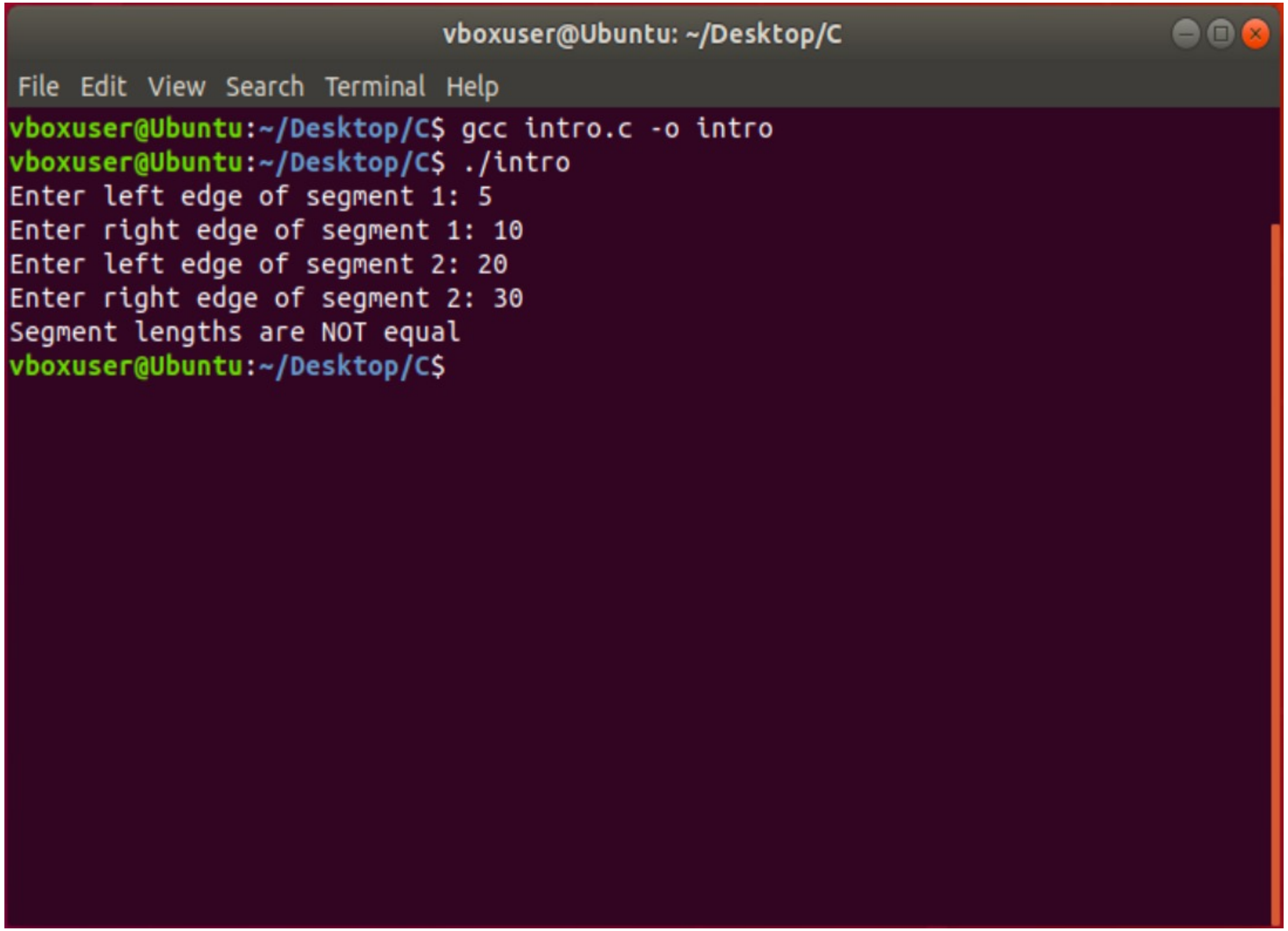
A Sample Program (For Illustration)

Specification:

1. Two line segments are created
2. The user is asked to enter the left and right edges of the two line segments, as integer values
3. The length of each segment is computed as (*right edge – left edge*)
4. The two lengths are compared to determine if they are the same, and a message is displayed



Compiling and Running the Program


vboxuser@Ubuntu: ~/Desktop/C
File Edit View Search Terminal Help

```
vboxuser@Ubuntu:~/Desktop/C$ gcc intro.c -o intro
vboxuser@Ubuntu:~/Desktop/C$ ./intro
Enter left edge of segment 1: 5
Enter right edge of segment 1: 10
Enter left edge of segment 2: 20
Enter right edge of segment 2: 30
Segment lengths are NOT equal
vboxuser@Ubuntu:~/Desktop/C$
```

Sample *C* Program (part 1)

- The following slides show sample program code to implement a solution to the problem described earlier.

- We will study each of the elements used in the *C* code throughout the term.

- By the end of the C programming module you will be able to write programs such as the one used in this example.

COLLEGE OF COMPUTING AND INFORMATICS

Sample C Program (part 2)

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

static int  compute_length (int, int);

int main (void)
{
    typedef struct {
        int         left;
        int         right;
        int         length;
    } seg_t;

    seg_t *seg1, *seg2;
```

library function definitions

main routine, procedure #1

data structure definition

declaration of references to data structure instances

Sample *C* Program (part 3)

```
    seg1 = (seg_t *) malloc (sizeof (seg_t));
    seg2 = (seg_t *) malloc (sizeof (seg_t));

    printf ("Enter left edge of segment 1: ");
    scanf ("%d", &(seg1->left));
    printf ("Enter right edge of segment 1:");
    scanf ("%d", &(seg1->right));
    printf ("Enter left edge of segment 2: ");
    scanf ("%d", &(seg2->left));
    printf ("Enter right edge of segment 2:");
    scanf ("%d", &(seg2->right));

    seg1->length = computelength (seg1->left,
          seg1->right);
    seg2->length = computelength (seg2->left,
          seg2->right);
```

create instances of data structure, and associate with references

input / output, store result in data structure

call a subroutine, store result in data structure

Sample *C* Program (part 4)

```
 if (seg1->length == seg2->length)
  printf("Segment lengths are equal\n");
 else
  printf("Segment lengths are NOT equal\n");

  return 0;
}

int compute_length (int left, int right)
{
  return (right-left);
}
```

subroutine, procedure #2

Variables and Datatypes

Identifiers (Names, Labels)

- Consist of letters, ‘_’, and digits

    cannot start with a digit (**`2_B_or_not_2_B`**) 🚫

- Case sensitive!

    `myVar` is not the same as **`myvar`**

- Unlimited length (advice: stop at 32)

    `gnome_memmgt_insert_into_heap_I_modified_this_because_I_can`

Reserved Keywords

- **Do not** use reserved keywords as identifiers, such as:

```
auto, break, case, char, const, continue,
default, do, double, else, enum, extern, float,
for, goto, if, int, long, register, return,
short, signed, sizeof, static, struct, switch,
typedef, union, unsigned, void, volatile, while,
_Bool, _Complex, _Imaginary, inline, restrict
```

C Variables

- A ***variable*** = a **location** in memory + its ***interpretation***
- Interpretation of a variable is based on its
  1. storage class and
  2. data type

COLLEGE OF COMPUTING AND INFORMATICS

Data **Types**

- The **data type** of a variable defines its interpretation

- Ex: suppose a 32-bit binary value stored in memory is
  01000001010000100100001101000100
  - if type **`float`**, interpreted to be numerical value 781.03521728515625
  - if type **`unsigned int`**, interpreted to be numerical value 1145258561
  - if type **`char`**, interpreted to be the ASCII string value ABCD

COLLEGE OF COMPUTING
AND INFORMATICS

Static or Dynamic Types

- In C variables are **statically** typed
  - A type must be specified when a variable is created, and cannot change thereafter
- Languages with **dynamic** typing (e.g., PHP, Python, Perl, Ruby, Javascript, …) are more flexible

Specializations of Fundamental Types

- Integers can be…
  - **`signed`** or **`unsigned`** (**`signed`** by default)
  - really short (**`char`**), **`short`**, regular (**`int`** by default), **`long`**, really long (**`long long`**)
- Floating point (always signed) can be…
  - regular precision (**`float`**)
  - double precision (**`double`**)
  - extended precision (**`long double`**)

COLLEGE OF COMPUTING AND INFORMATICS

Min and Max Integer Values

The **lengths** (in bits) (and the max and min values) of these types are **platform dependent**

Type	# bits	Value
Min ‘unsigned anything’	n.a.	0
Min ‘signed char’	8	-128
Max ‘signed char’	8	127
Max ‘unsigned char’	8	255
Min ‘signed short int’	16	-32,768
Max ‘signed short int’	16	32,767
Max ‘unsigned short int’	16	65,535

Integer Values… (cont’d)

Type	# bits	Value
Min ‘**signed int**’	32	-2,147,483,648
Max ‘**signed int**’	32	2,147,483,647
Max ‘**unsigned int**’	32	4,294,967,295
Min/Max ‘**signed long int**’	64	same as ‘**signed long long int**’
Max ‘**unsigned long int**’	64	same as ‘**unsigned long long int**’
Min ‘**signed long long int**’	64	-9,223,372,036,854,775,808
Max ‘**signed long long int**’	64	9,223,372,036,854,775,807
Max ‘**unsigned long long int**’	64	18,446,744,073,709,551,615

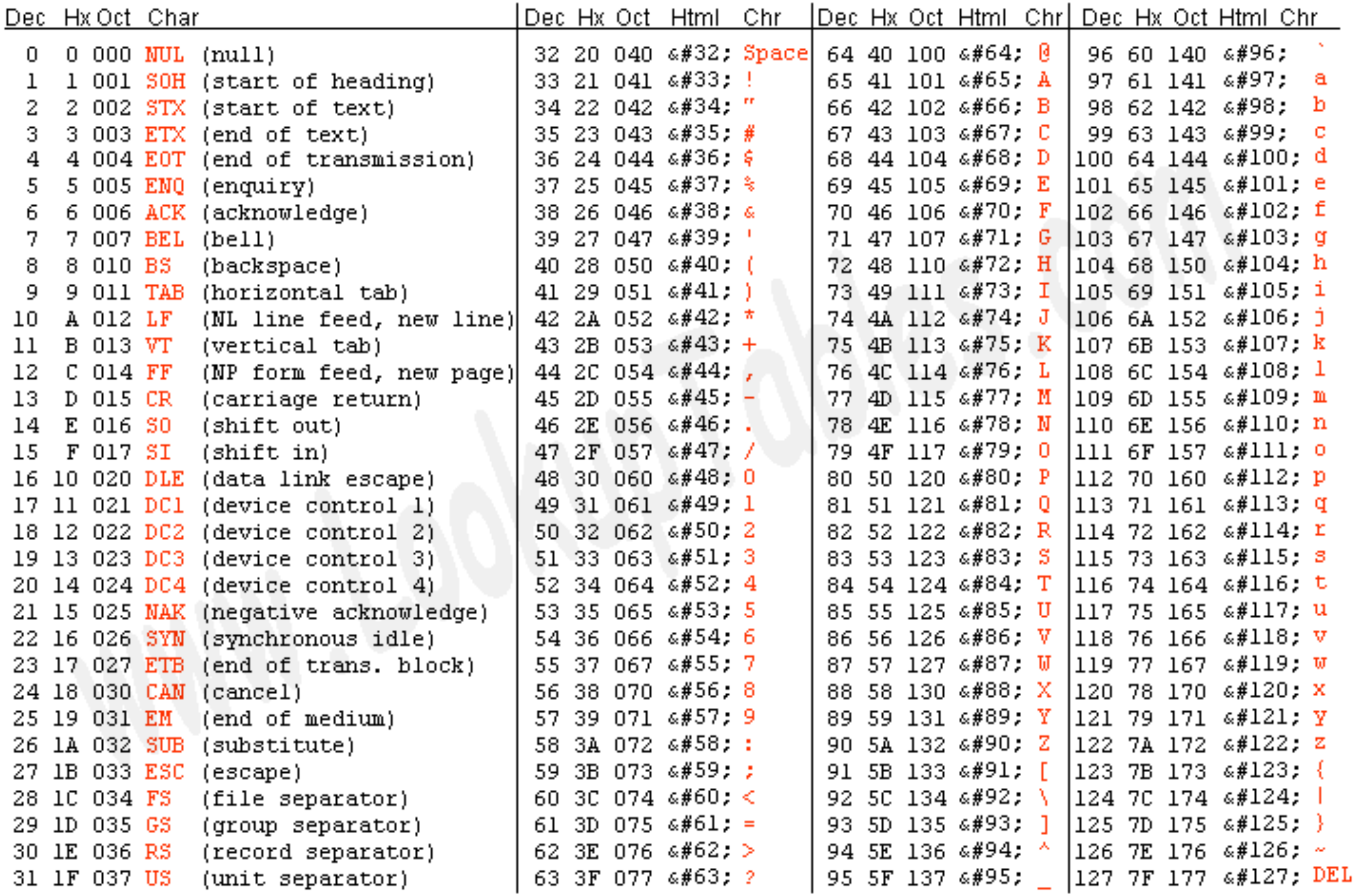
Constants

- Types of constants (set once and never changed)
  - **integer**
  - **floating point**
  - **character** (a type of integer)
  - **enumeration** *(we'll talk about these later)*
- Character constants in single quotes: **`'a'`**, **`'b'`**
  - value stored is the numeric value of the character in ASCII
- **`#define <CONSTANT_NAME> <value>`**

ASCII

- The ASCII code is used by computers to represent characters, such as letters, special symbols and digits
- ASCII is a specific 8-bit encoding of Western characters (punctuation, digits, upper and lower case characters)
- Only the **first 128 values** are standardized
- The interpretation of the **remaining 128** values are **application/platform-specific**

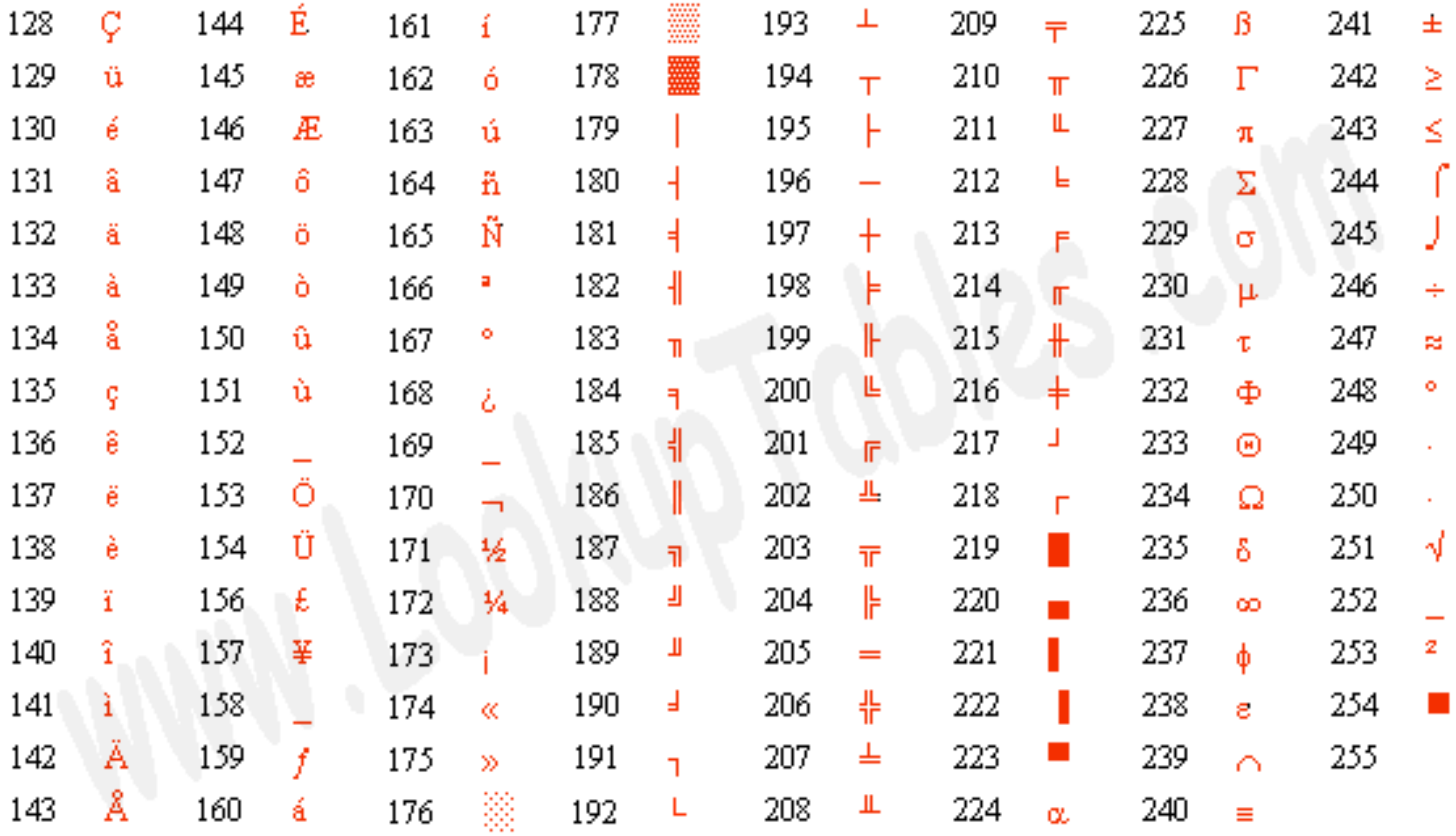
Standardized ASCII (0-127)



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

One Interpretation of 128-255



128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179	│	195	├	211	╙	227	π	243	≤
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	169	_	185	╣	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	171	½	187	╗	203	╦	219	█	235	δ	251	√
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	_
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
141	ì	158	_	174	«	190	╛	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	
143	Å	160	á	176	░	192	└	208	╨	224	α	240	≡		

Source: www.LookupTables.com

Useful Character Constant Escape Sequences

- `\0`   Null character
- `\‘`   Single quote
- `\“`   Double quote
- `\\`   Backslash
- `\n`   Newline
- `\t`   Horizontal tab
- `\nnn` Octal value of character
  (ex: `‘a’ == ‘\141’`)
- `\xnn` Hexadecimal value of character
  (ex: 'a' `== ‘\x61’`)

(see `letter.c` in *Code samples and Demonstrations* in *Canvas*)

References

- S. J. Matthews, T. Newhall and K. C. Webb, *Dive into Systems*, Version 1.2. Free online textbook, available at: https://diveintosystems.org/book/

- K. N. King, *C Programming: A Modern Approach*, 2nd Edition. W. W. Norton & Company. 2008.

- D.S. Malik, *C++ Programming: From Problem Analysis to Program Design*, Seventh Edition. Cengage Learning. 2014.