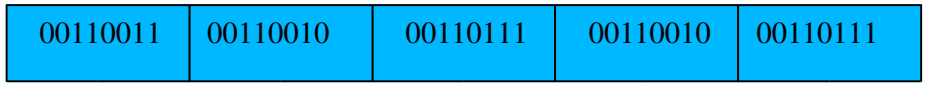
Introduction to C

CS2023 Winter 2004

Outcomes: Introduction to C

- After the conclusion of this section you should be able to
  - Recognize the sections of a C program
  - Describe the compile and link process
  - Compile and run C programs
  - Explain the difference between text and binary files
  - Explain the difference between reading from standard input and reading from a file
  - Follow the C language coding standard for CS2023
  - Begin to appreciate the importance of good programming style

Text Files

- Text files are *line-oriented* (unlike binary files)
- 32,767 as a text file: 

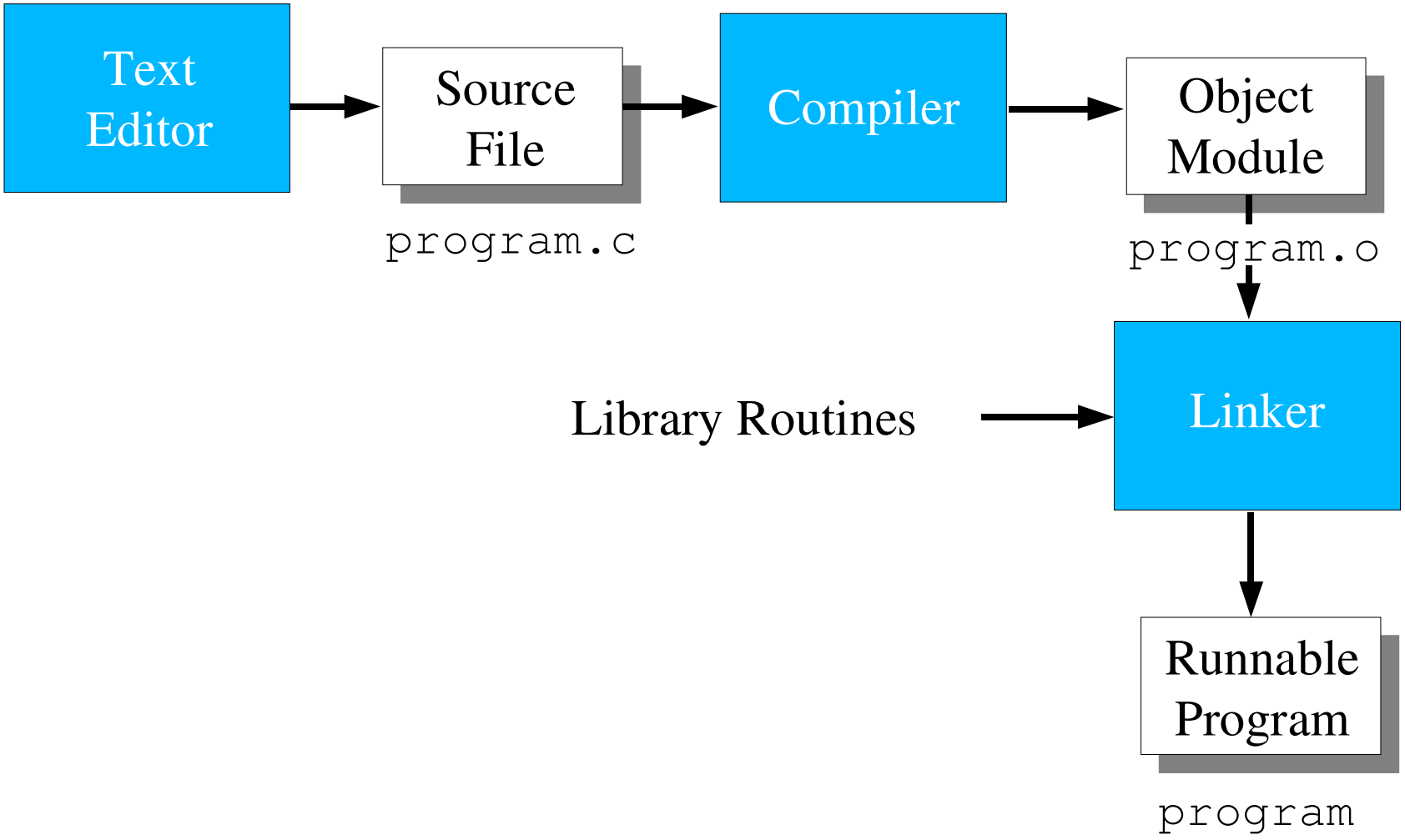
- 32,767 as a binary file: 

- End-of-line terminator:
  - Windows: carriage return + linefeed
  - Mac: carriage return
  - UNIX: linefeed

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

Source: www.asciitable.com

From source file to runnable program



Portability, Efficiency, and Correctness

- C object code runs only on target machine
- However, it is possible (and desirable!) to write portable C source code
- First make sure your program is correct, then optimize if necessary
- Design for change

Comparison of C and Java

- *Primitive data types*: character, integer, and real
  - In C, they are of different sizes
  - there is no Unicode 16-bit character set
- *Structured data types*: arrays, structures and unions.
  - In C, arrays are static
  - there are no classes
- *Control structures* are similar
- *Functions* are similar

Comparison of C and Java

- Java references are called pointers in C.
- Java constructs missing in C:
  - packages
  - threads
  - exception handling
  - garbage collection
  - standard Graphical User Interface (GUI)
  - built-in definition of a string
  - standard support for networking
  - support for program safety.

Two Simple Programs

```
/* Output all of input on one line */

#include <stdio.h>

int main()
{
  int c;

  while((c = getchar()) != EOF){
    if(c != '\n') putchar(c);
  }
  putchar('\n');
  return 0;
}
```

Two Simple Programs

```
/* Output all of file "name" on one line */

#include <stdio.h>

int main()
{
  int c;
  FILE *f;

  f = fopen("name", "r");

  while((c = getc(f)) != EOF){
    if(c != '\n') putchar(c);
  }
  putchar('\n');
  fclose(f);
  return 0;
}
```

How do these programs differ?

Source File Organization

```
/* krfind.c: print lines that match pattern from first
 * argument in file given by second argument
 * Author: Eric Aubanel
 * Date: September 13, 2002
 * Based on Kernighan & Ritchie, The C Programming
 * Language, 2nd edition P. 116
 */
#include <stdio.h>
#include <string.h>

#define MAXLINE 1000

FILE *inFile;

/* getline: get line from inFile into line[], return
length */
int getline(char line[], int max);
```

Language Coding Standard: Source Code Organization

- File Documentation

  the first item in a source file should be a comment block identifying the name of the file, its author and what functionality the code provides. Each individual function within the file should also have a comment block naming the function and the functionality it provides.

- Preprocessor Information: `#define MAXLINE 1000`

  list the header files that are needed, followed by the preprocessor macros. Preprocessor macro names should use capital letters only.

Language Coding Standard: Source Code Organization

- Type definitions

  programmer-defined data types, using the C keyword `typedef`

- Global variables (use sparingly!): `FILE *inFile;`

- Function Prototypes:

```
int getline(char line[], int max);
```

  prototypes for all programmer-defined functions should be presented. The arguments should be specified with both data type and name

Language Coding Standard: Source Code Organization

- Main function
- Programmer-defined functions

```
int main(int argc, char *argv[]){
  char line[MAXLINE];

  if (argc != 3) {
    fprintf(stderr,"Usage: %s pattern filename\n",argv[0]);
    return 1;
  }
  if((inFile = fopen(argv[2], "r")) == NULL) {
    fprintf(stderr,"Cannot open file %s\n", argv[2]);
    return 1;
  }

  while (getline(line, MAXLINE) > 0)
    if(strstr(line, argv[1]) != NULL)
      printf("%s", line);

  if(fclose(inFile) == EOF) {
    fprintf(stderr, "Cannot close file %s\n", argv[2]);
    return 1;
  }
  return 0;
}
```

Function Format

```
int main(int argc, char *argv[]){
```

- The main function must be of type int and must return appropriate return codes.
- Variable Declaration: all variables to be used in the function will be listed immediately following the function's opening brace.
  - Comments are only required for global variables (which should normally not be used, except in specific cicumstances), constant definitions, fields in structures. Instead of commenting each variable, use meaningful names. Names made up of multiple words should have the initial letter capitalized, except for the first word (e.g. FrontOfTheQueue), or be separated with underline characters (e.g. front_of_the_queue).

Function Format

- Function code
- Function return: the return statement should appear as the last statement of the function, unless the function returns nothing (void):

```
int getline(char s[], int lim)
{
  int c,i;

  i = 0;
  while (--lim > 0 && (c=fgetc(inFile))!=EOF &&
          c != '\n')
    s[i++] = c;
  if (c == '\n')
    s[i++] = c;
  s[i] = '\0';
  return i;
}
```

Statement Block Format

- Code blocks should be set off by indenting using a tab.
  - In the Emacs C environment this will be done automatically, which also helps spot syntax errors as you write.
- Statements that exceed the width of the screen (or printed page) should be broken into more than one line so that the continuation line is indented from the parent line.

Prohibitions

- The use of side effects is discouraged since it distracts from readability and comprehension, *and can lead to unpredictable results!*

```
i = 1;
j = i++ + i++;
```

  what is the value of `j`?

- Use of the standard I/O function `gets()` is absolutely prohibited since introduces an an unavoidable security hole and/or bug into every program it is a part of.

The Practice of Programming

- Kernighan and Pike (on reserve in library): practice of programming includes testing, debugging, portability, performance, design alternatives, and style
- **Simplicity**
  - Keep programs short and manageable
- **Clarity**
  - Programs should be understandable by people as well as machines
- **Generality**
  - Programs adapt well to new situations

Communication

- Only 30% of an average programmer's time is spent working alone
- Even less time is spent communicating with a computer
- An experienced programmer writes code for an audience of people rather than machines

Programming Style

- Source code is written for people!
  - A well-written program is easier to understand and modify than a poorly-written one
- Logic of program should be straightforward
- Meaningful names
- Helpful comments
- Consistency
  - Use idioms!

```
while(*p++ = *q++)
;
```

Why Idioms?

```
i =0;
while (i <= n-1)
  array[i++] = 1.0;

for (i = 0; i < n; )
  array[i++] = 1.0;

for (i = n; --i >= 0; )
  array[i] = 1.0;

for (i = 0; i < n; i++)
  array[i] = 1.0;
```

Layout of a C Program

- Program made up of *tokens*
  - *e.g.* `main`, `(`, and `)`
- Statements can be divided over any number of lines
- *Space* between tokens improves readability:

```
volume = height * length * width;
```

- *Indentation* makes nesting easier to spot
- *Blank lines* divide a program into logical units

The `main` Program

- Every C program must include a function called `main`
- Two possible prototypes:

```
int main(int argc, char *argv[])
int main()
```

- Return codes: 0 (success) or 1 (failure) or:
  - `EXIT_FAILURE`, `EXIT_SUCCESS`
    (include `stdlib.h`)

Comments

```
/* Hello */
```

Wrong:

```
 /* outer /* inner */ */

//
/**  */
```

Comments

```
if(isdigit)  /* error */
```

```
/*
 * Program to sort integer values
 */
```

```
 k++; /* k is incremented by 1*/
```

Comments

- Make every comment count.
- Don't over-comment.
- Don't comment bad code, rewrite it!
- Make sure comments and code agree.

```
if ( (country == SING) || (country == BRNI) ||
     (country == POL) || (country == ITALY) {
/* If the country is Singapore, Brunei or
 * Poland then the current time is the answer
 * time rather than the off hook time.
 * Reset answer time and set day of week */
```

Identifiers

- Use a *consistent* style throughout your code.
- Variables should have meaningful names *when appropriate* (`i`, `j`, for counters is fine)

```
LongIdentifier  long_identifier
```