ECE 209 ANSI C Reference

Program Structure / Functions

type fnc(*type*$_1$, ...);	function declaration

`int main() {`
declarations
statements
`return EXIT_SUCCESS;`
`}`
	main function
	defined in `<stdlib.h>`

type fnc(*type*$_1$ *param*$_1$, ...) {	function definition
declarations *statements* `return` *expr;* `}`	
fnc(*expr*$_1$, ...)	function call
`/* */`	comment
`int main(int argc, char *argv[])`	main with args

C Preprocessor

`#include <`*filename*`>`	include system file
`#include "`*filename*`"`	include user file
`#define` *name text*	replacement text
`#define` *name*`(`*var*`)` *text*	macro
`#if`	conditional compilation
`#else`	
`#elif`	
`#endif`	
`#ifdef`	
`#ifndef`	
`\`	line continuation char

Data Types

`char`	character
`int`	integer
`unsigned int`	unsigned integer
`double`	floating-point
type `*`	pointer to *type*
`const` *type* `*`	pointer to constant *type*
`typedef` *type name*	user-defined type
`size_t`	size of something (unsigned integer)

Initialization

type name = *value*	initialize variable
type name`[`*size*`] = {`*value*$_1$, ...`}`	initialize array
type name`[] = {`*value*$_1$, ...`}`	initialize array
`char` *name*`[`*size*`] = "`*string*`"`	initialize string
`char` *name*`[] = "`*string*`"`	initialize string

Literals (examples)

`123`	decimal integer
`0xA1F, 0xa1f`	hexadecimal integer
`123`	floating-point
`1.23e5`	floating-point (1.23×10^5)
`3e-5`	floating-point (3.0×10^{-5})
`'a'`	character
`'\n', '\0'`	linefeed, null character
`'\\', '\?', '\'', '\"'`	special characters
`"abc"`	constant string (ends w/null)

Pointers, Arrays

type `*`*name*	declare pointer to *type*
`void *`	generic pointer type
`*`*pointer*	value pointed to by *pointer*
`&`*var*	address of variable *var*
`NULL`	zero pointer `<stdlib.h>`
type name`[`*size*`]`	declare array of *type*
	size must be constant integer
type name`[`*size*`][`*size*`]`...	multi-dimensional array

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	
`struct` *tag name*	declare a struct variable
expr`.`*member*	member of a struct value
pointer`->` *member*	member of pointed-to struct

Note: `p->x` and `(*p).x` are the same.

Operators (grouped by precedence)

structure member operator	*name*`.`*member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
dereference, address-of	`*`*pointer*, `&`*var*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus	`*, /, %`
add, subtract	`+, -`
bitwise left shift, bitwise right shift	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise AND	`&`
bitwise XOR (exclusive-OR)	`^`
bitwise OR	`|`
logical AND	`&&`
logical OR	`||`
conditional expression	*expr1*`?`*expr2*`:`*expr3*
assignment operators	`=, +=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression, and assignment operators group right to left; all others group left to right.

String Functions `<string.h>`

s, t are strings; cs, ct are constant strings; n is size_t; c is char

`size_t strlen(cs);`	length of string
`char * strcpy(s,ct);`	copy ct to s
`char * strncpy(s,ct,n);`	copy ct to s, up to n chars
`int strcmp(cs,ct);`	compare cs to ct
`int strncmp(cs,ct,n);`	compare cs to ct, first n chars
`char * strcat(s,ct);`	append ct to s
`char * strchr(cs,c);`	pointer to first c in cs, or NULL
`char * strrchr(cs,c);`	pointer to last c in cs, or NULL

Flow of Control

`;`	statement terminator
`{ }`	compound statement
`break`	exit loop
`continue`	next iteration of loop
`return` *expr;*	return value from function
`if (`*expr*`)` *statement*	conditional
`if (`*expr*`)` *statement* `else` *statement*	conditional with else
`while (`*expr*`)` *statement*	while loop
`for (`*expr1; expr2; expr3*`)` *statement*	for loop
`do` *statement* `while (`*expr*`);`	

Input/Output `<stdio.h>`

`stdin`	standard input stream
`stdout`	standard output stream
`EOF`	end of file
`printf(`*fmt, value1,* ...`)`	print to stdout
`scanf(`*fmt, ptr1,* ...`)`	read from stdin
`FILE *`*fp*	declare file pointer
`fopen(`*name, mode*`)`	open named file mode is "r", "w", "a"
`fprintf(`*fp, fmt, value1,* ...`)`	print to stream *fp*
`fscanf(`*fp, fmt, ptr1,* ...`)`	read from stream *fp*
`fclose(`*fp*`)`	close stream

Select conversion codes for printf: "%*#c*"

(optional) is minimum field width; *c* is one of the following:

`d,i`	decimal integer
`u`	unsigned decimal integer
`f`	double
`e,E`	double (exponent notation)
`g,G`	double (f or e,E notation)
`c`	character
`s`	string
`p`	pointer
`x,X`	hexadecimal integer

Format string for scanf may contain:

- Spaces or tabs (ignored). (Don't use them!)
- Ordinary characters (not %), expected to match the next non-white character.
- Conversion code: % followed by optional assignment suppression character (*), an optional integer (maximum field width), and a conversion character (see printf).

%i is for general integer (decimal, or hex if 0x...).
%f, %e read float values, not double. No E, g, G.

Memory Allocation

`malloc(`*n*`)`	allocate *n* bytes, return pointer
`free(`*ptr*`)`	deallocate allocated memory at *ptr*

ASCII Code

Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp (space)	32	20	@	64	40	` (accent)	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	' (single quote)	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht (tab)	9	09	)	41	29	I	73	49	i	105	69
lf (linefeed)	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	, (comma)	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_ (underscore)	95	5F	del	127	7F

ADD*	0001	DR	SR1	0	00	SR2	
ADD*	0001	DR	SR1	1	imm5		
AND*	0101	DR	SR1	0	00	SR2	
AND*	0101	DR	SR1	1	imm5		
BR	0000	n	z	p	PCoffset9		
JMP	1100	000	BaseR	000000			
JSR	0100	1	PCoffset11				
JSRR	0100	0	00	BaseR	000000		
LD*	0010	DR	PCoffset9				
LDI*	1010	DR	PCoffset9				
LDR*	0110	DR	BaseR	offset6			
LEA*	1110	DR	PCoffset9				
NOT*	1001	DR	SR	111111			
ST	0011	SR	PCoffset9				
STI	1011	SR	PCoffset9				
STR	0111	SR	BaseR	offset6			
TRAP	1111	0000	trapvect8				

*Instructions marked with * also set condition codes.*