

CS-2060 Technical Documentation

GE03 - February 5, 2024

Table of Contents

Secure C Programming Guidelines:	1
Avoid Single-Argument printf's – Section 2.7	1
Scanf, printf, scanf_s and printf_s – Section 2.7 & 4.11	1
Arithmetic Overflow – Section 3.13	2
Random Number Generation – Section 5.17	2
ASCII Reference Chart:	3
Functions:	4
Programmer Defined Functions:	4
Function Examples from C Library:	4
Function Prototype and Definition:	5
Important Resources:	5
• How to clear the input buffer	5
• Agile Software Development Cycle	5
• Software Design Principles DR and KISS	5
• How to Solve Programming Problems in 4 Steps	5

Secure C Programming Guidelines:

(Textbook Chapters 2, 3, 4, 5)

Avoid Single-Argument printf's - Section 2.7

- Can be exploited by user by inputting a string with more conversion specifications than there are additional printf arguments. This allows them to [read from memory](#).

- `Printf("Welcome to C!"); // Incorrect`

- `Put("Welcome to C!"); // Correct`

Scanf, printf, scanf_s and printf_s - Section 2.7 & 4.11

- Scanf_s and printf_s **will not** be used as they are a part of annex K which is implemented by windows.
- **Scanf**
 - Returns an int indicating whether the input operation was successful
 - EOF is returned if an input failure occurs (<stdio.h>)

- `scanf("%d", &grade);` // read grade from user
- If integer is entered, `scanf` returns 1
- If a string is entered (or any other data type) 0 is returned and grade does not receive a value
- Range Checking
 - You can use a range (0-100 for example) to check whether the input is valid, if not re-prompt the user to enter a correct value
 - Example: `if (scanfResult == 1) { if (grade <= 100 && grade >= 0) { // allow continuation of code }}`

Arithmetic Overflow - Section 3.13

- `Sum = integer1 + integer2;` // Could be a potential problem
 - Two integers could be added which exceed the maximum value that an integer can hold; **this would result in arithmetic overflow**
- Ensure numbers are kept within the `INT_MAX` and `INT_MIN` defined in header `<limits.h>`
- A solution to the following issue can be found in [INT32-C](#):

```
#include <limits.h>

void f(signed int si_a, signed int si_b) {
    signed int sum;
    if (((si_b > 0) && (si_a > (INT_MAX - si_b))) ||
        ((si_b < 0) && (si_a < (INT_MIN - si_b)))) {
        /* Handle error */
    } else {
        sum = si_a + si_b;
    }
    /* ... */
}
```

Random Number Generation - Section 5.17

- The `rand()` function should not be used for industrial-strength cryptography.
 - Some implementations of the function produce sequences which are distressingly non-random low-order bits
 - See [MSC30-C](#)
 - A predictable random generator could look like:

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

enum { len = 12 };

void func(void) {
    /*
     * id will hold the ID, starting with the characters
```

```

    * "ID" followed by a random integer.
    */
    char id[len];
    int r;
    int num;
    /* ... */
    r = rand(); /* Generate a random integer */
    num = sprintf(id, len, "ID%-d", r); /* Generate the ID */
    /* ... */
}

```

ASCII Reference Chart:

dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
0	0	00	NUL (null)	32	40	20	(space)	64	100	40	@	96	140	60	`
1	1	01	SOH (start of header)	33	41	21	!	65	101	41	A	97	141	61	a
2	2	02	STX (start of text)	34	42	22	"	66	102	42	B	98	142	62	b
3	3	03	ETX (end of text)	35	43	23	#	67	103	43	C	99	143	63	c
4	4	04	EOT (end of transmission)	36	44	24	\$	68	104	44	D	100	144	64	d
5	5	05	ENQ (enquiry)	37	45	25	%	69	105	45	E	101	145	65	e
6	6	06	ACK (acknowledge)	38	46	26	&	70	106	46	F	102	146	66	f
7	7	07	BEL (bell)	39	47	27	'	71	107	47	G	103	147	67	g
8	10	08	BS (backspace)	40	50	28	(72	110	48	H	104	150	68	h
9	11	09	HT (horizontal tab)	41	51	29)	73	111	49	I	105	151	69	i
10	12	0a	LF (line feed - new line)	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	0b	VT (vertical tab)	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	0c	FF (form feed - new page)	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	0d	CR (carriage return)	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	0e	SO (shift out)	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	0f	SI (shift in)	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	DLE (data link escape)	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	DC1 (device control 1)	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	DC2 (device control 2)	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	DC3 (device control 3)	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	DC4 (device control 4)	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	NAK (negative acknowledge)	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	SYN (synchronous idle)	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	ETB (end of transmission block)	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CAN (cancel)	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	EM (end of medium)	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	SUB (substitute)	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	ESC (escape)	59	73	3b	;	91	133	5b	[123	173	7b	{
28	34	1c	FS (file separator)	60	74	3c	<	92	134	5c	\	124	174	7c	
29	35	1d	GS (group separator)	61	75	3d	=	93	135	5d]	125	175	7d	}
30	36	1e	RS (record separator)	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	US (unit separator)	63	77	3f	?	95	137	5f	_	127	177	7f	DEL (delete)

Functions:

(Lecture 5, Ch05, Slides 6-8)

Programmer Defined Functions:

Programmer Defined Functions

Why Functions?

- Define new function to perform a specific task.
- They are used to reuse, organize and help debug by grouping lines of code.
- You must define function and then you can invoke the function

Function Definition

- what value it returns (if any)
- what values it takes (if any)
- what the function does



```
returnType FunctionName (parameter list)
{
    //initialize local variables
    statement; // function body
    ....
    statement;
}
```

Local Variable

- Variable defined within a function
- Can only be accessed within the function

returnType - data type for value that is returned (int, double, bool, etc.) or void (nothing to return)

functionName - a user defined name

parameter list - values being passed to the function (explicitly specify type for each value)

Function Examples from C Library:

Examples from C Library

Function Prototype

- General form of prototype also called declaration
 - Looks like definition but no function body - { }
- Value Returning Function - performs some task and returns a value
- Void Function - performs some task without returning a value

Parameters

- The values that are specified in the function definition
- How information is passed to a function
- Comma separated list of variables

Arguments

- The values that are specified in the function invocation (call)
- The information we want to send to a function

Function Invocation

- General form of "invoking" or "calling" a function

[Power Function](#) from <math.h>

C Library declaration:

double pow(**double** x, **double** y)

Return value: **double**

Parameters: **double** x, **double** y

Invocation:

```
double radiusSquared = pow(3.2, 2);
```

Arguments: 3.05, 1.98

[Seed Random Number Generator](#)

from <stdlib.h>

C Library declaration:

void srand(unsigned int seed)

Return value: **void** (none)

Parameter: **seed**

Invocation:


```
srand(time(0));
```

Arguments: time(0)


Function Prototype and Definition:

Function Prototype and Definition

Include Function prototype. Notice it is the same as the first line of function definition. Prototype has semicolon at the end.

 **Error-Prevention**

Check that your functions that are supposed to return values do so. Check that your functions that are not supposed to return values do not.

 **Good Programming**

Although it's not incorrect to do so, it is good practice to not use the same names for a function's arguments and the corresponding parameters in the function definition. This helps avoid ambiguity. **Do this!**

```
// Fig. 5.3: fig05_03.c
// Creating and using a prototype function.
#include <stdio.h>

int square(int y); // function prototype

int main(void)
{
    // loop 10 times and calculate and output square of x each time
    for (int x = 1; x <= 10; ++x) {
        printf("%d ", square(x)); // function call
    }
    puts("");
}

// square function definition returns the square of its parameter
int square(int y) // y is a copy of the argument to the function
{
    return y * y; // returns the square of y as an int
}
```

function prototype (declaration)

function definition

Important Resources:

- [How to clear the input buffer](#)
- [Agile Software Development Cycle](#)
- [Software Design Principles DR and KISS](#)
- [How to Solve Programming Problems in 4 Steps](#)