COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

School of Computing
(University of Plymouth)

Introduction to Low-Level Programming using C Programming Language

- For the rest of this module, we will be using C language, as it is the dominant language for low-level programming and Operating Systems
  - C and assembly can be mixed

- **Outline of this presentation**
  - Introduction to C programming
  - C data types
  - I/O
  - Strings in C
  - Functions
  - For loops

Why C ?

- C powers the World
  - Many nowadays Operating Systems have been build using C language
    - Windows / Windows phone
    - OS X / iOS
    - Android
    - Linux
  - Several DataBases have been build using C
    - Oracle DataBase
    - MySQL
    - MS SQL server
    - PostgreSQL
  - C/C++ is the dominant language in Embedded Systems
  - C/C++ is the dominant language in High Performance Computing

C Language Pros and Cons

- **Advantages**
  - Highly portable
  - It is very close to assembly and high compatible with assembly
  - It is the building block of many other programming languages such as python and java
  - It is very fast (execution time)
  - Dynamic memory allocation
  - Can be used for low level programming such developing OS kernel and drivers
- **Disadvantages**
  - Not support of Object Oriented programming. However, C++ provides that
  - More difficult to debug

First C program


```
#include <stdio.h> //this is needed for printf()

int main()   // Every program starts with main()
{ //start of main() block

printf("\n Hello, World! \n"); // '\n' is for new line


return 0; //the main function must return an int

} /* end of main() block */
```

Comment – works just for a single line

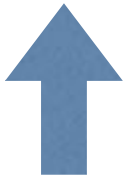
Comment – works for many lines

Basic Data Types

- Variables must be declared before use
- All data in C have a specified type, e.g.,
  - ***Int*** – normally 4 bytes
  - ***Short int*** – normally 2 bytes
  - ***Long int*** – normally 8 bytes
  - ***Unsigned Int*** – normally 4 bytes
  - ***Unsinged Short int*** – normally 2 bytes
  - ***Unsigned Long int*** – normally 8 bytes
  - ***Float*** - 4 bytes Floating point
  - ***Double*** - 8 bytes Floating point
  - ***Char*** – 1 byte (characters)
  - ***Void*** – it represents the lack of a data type

formatted print - printf


```
int printf(const char* format, ...);
```



number of characters printed
(or -ve error code)

format string (see printf)

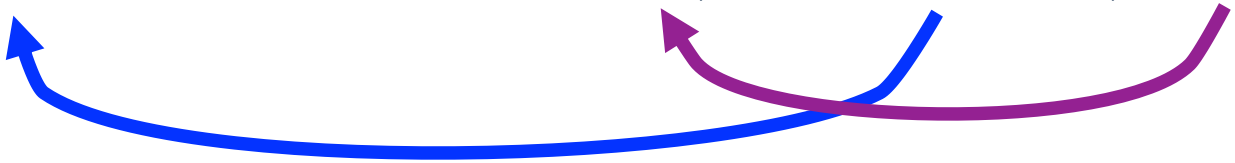
List of constants/ variables

Example 1:

```
printf("The year %d was a long time ago\n", 1966);
```

Example 2:

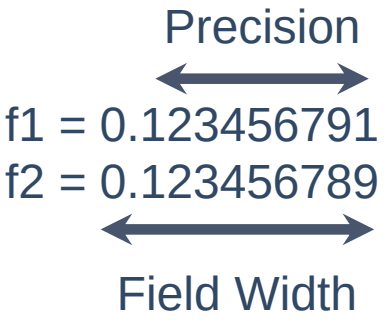
```
printf("The %s was invented in %d","transistor",1948);
```



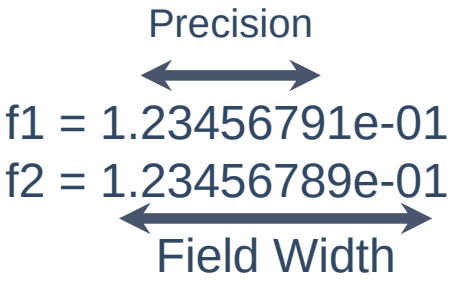
Output field width and precision


```
float  f1 = 0.123456789;
double f2 = 0.123456789;

printf("f1 = %11.9f\n", f1);
printf("f2 = %11.9f\n", f2);
```




```
printf("f1 = %14.8e\n", f1);
printf("f2 = %14.8e\n", f2);
```



printf Formatted Output Placeholders

d,i	int as a signed decimal number.
u	unsigned decimal
f, F	double in normal (fixed-point) notation.
e, E	double value in standard form ([-]d.ddd e[+/-]ddd).
x, X	unsigned int as a hexadecimal number. 'x' uses lower-case letters and 'X' uses upper-case.
s	null-terminated string.
c	char (character).

1st Activity (Task 1)


```
printf("Size of char: %lu\n", sizeof(char) );
printf("Size of short: %lu\n", sizeof(short) );
printf("Size of int: %lu\n", sizeof(int) );
printf("Size of long: %lu\n", sizeof(long) );
printf("Size of long long: %lu\n", sizeof(long long) );
printf("Size of float: %lu\n", sizeof(float) );
printf("Size of double: %lu\n", sizeof(double) );
printf("Size of long double: %lu\n", sizeof(long double) );
```

What is a function?

- Functions are self contained modules of code that accomplish a specific task.
- Functions have inputs and output
- main() is the first function always being executed

```
int function(int temp); //declare function

void main() {

int temp=5;
int out;

out=function(temp);
printf("\nout=%d",out); //should print 6
}
```

```
int function (int temp) { //define function
  return temp+1;
}
```

Task2 – Array Addition

Strings in C

- Strings are 1d arrays of characters (type **char**) terminated by a null character '\0'
- The **char** data type is a single byte, giving a value from 0 to 255.
- A **char** variable can therefore represent a single ASCII character.

```
char myChar = 65;
```

Identical

```
char myChar = 'A';
```

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

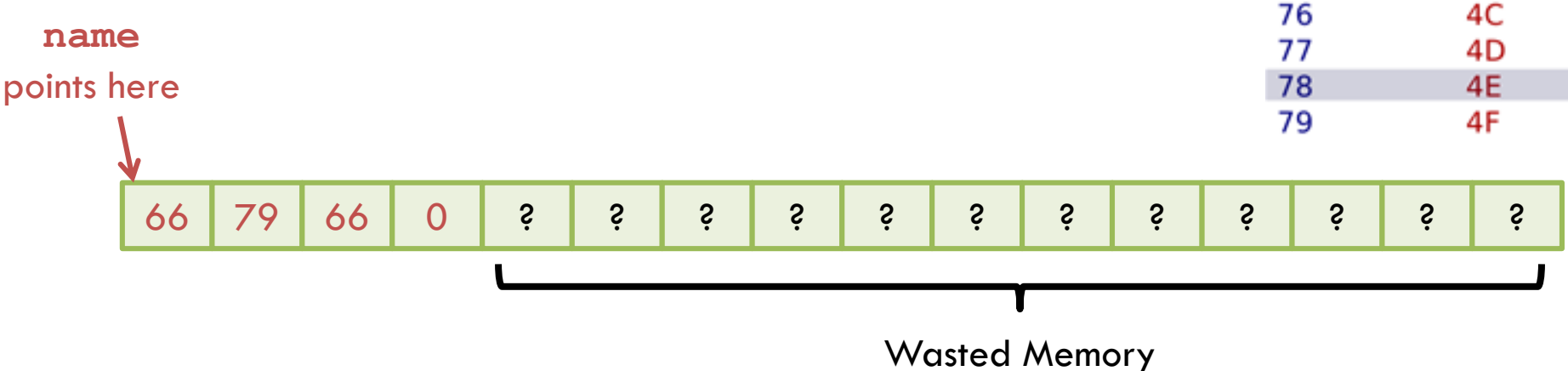
Source: www.LookupTables.com

Strings in C (2)


```
// Make C-string and copy text "BOB" into it.
char name[16];
strcpy(name, "BOB");
```

Decimal	Hex	Cha
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

- Allocates space for 16 chars.
  - So what's the maximum name length?
  - 15: need to include a 0 in the end.
- Mark the end of the actual text with a zero – elements after this can be anything.



Strings in C: Some Operations


```
char name[16];
strcpy(name, "BOB");
printf("%d", strlen(name));
```

Prints: 3

```
strcat(name, " MERRISON");
printf("%s", name);
```

Prints: BOB MERRISON

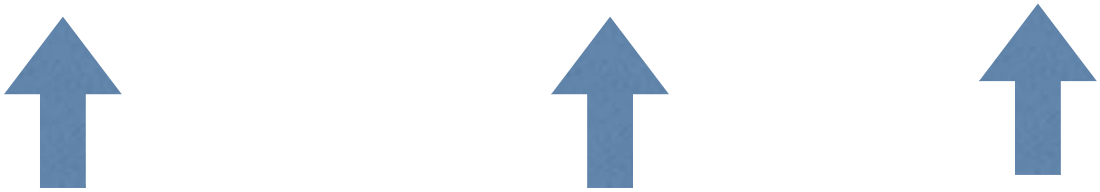
```
name[0] = NULL;
printf("%s",name);
```

Prints nothing.

To clear a string: set first element to NULL (0).

formatted input - scanf


```
int scanf(const char* format, ...);
```



number of items matched or EOF

pattern matching string

List of variable **addresses**

Example 1:

```
int x;
scanf("%d", &x);
```

Example 2:

```
char myString[32];  //32 character string
scanf("%s",myString);
```

Scanf – Task3


```
#include <stdio.h>
#include <windows.h> //this library is needed for pause() function

int main()
{
    char name[20]; //array of 20 characters. The last character is always zero

    printf("\nEnter your name: "); //get input from keyboard

    scanf_s("%19s", name, sizeof(name)); //read 19 characters from the keyboard
and store them into the name[] array.

    printf("\nYour name is %s\n", name);

system("pause");
    return 0;

}
```

Homework Activities

- Task4: Find the frequency of a character into a string.

- Task 5: Write a C program which calculates the length of a string.

Variable Address

- Every variable is stored into a memory location
- Every memory location has an address
- An **address** can be accessed by using ***the ampersand (&) operator***

```
int main() {

int var1=4;

printf("\nThe memory address of %p contains %d \n", &var1, var1);
return 0;
}
```

Activities

- Task 6. Can you write a program that prints the memory addresses of an array?
- Task 7. Can you write a program that adds two arrays and stores the result into another?
- Task 8. Can you write a program finding the largest element of an array?

Further Reading

- If you want to learn more about C programming you can study the following links:
  - Tim Bailey, 2005, An Introduction to the C Programming Language and Software Design, available at: http://www-personal.acfr.usyd.edu.au/tbailey/ctext/ctext.pdf
  - C examples, Programiz, available at https://www.programiz.com/c-programming/examples
  - C Programming examples with Output, Beginners book, available at https://beginnersbook.com/2015/02/simple-c-programs/
  - C Programming Tutorial, from tutorialspoint.com, available at https://www.unf.edu/~wkloster/2220/ppts/cprogramming_tutorial.pdf

Thank you