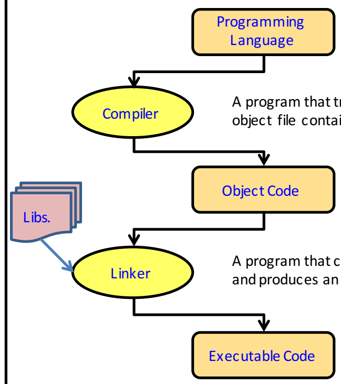
Outline

- Introduction to the C programming language
- Writing and compiling a C program
- Literal Values and their Type
- Operators
- Variables

Programming Languages



E.g. Fortran (FORMular TRANslation)
Pascal; COBOL; Java; and of course C.

A program that translates the higher level language into an object file containing *machine code*.

A program that combines the object code with predefined routines and produces an executable program.

Computers work on *binary numbers*
1001 1100 (Called *machine code*)
0011 1011 … Etc.

What is C?

- C is a medium-level, structured, procedural, imperative programming language.
- Compared to high-level languages, like BASH, Python, Fortran, Java, Haskell, *et al*, statements in C are "closer" to the native instructions of the computer.
- Compared to low-level (assembly) languages, there is not a precise one-to-one mapping to those instructions, however.

What is C?

- C is a medium-level, structured, procedural, imperative programming language.
- You can group together sets of statements into *blocks* that can be treated as a whole.
- *Conditional* statements let you execute different blocks dependant on a test result.
- *Flow control* (loops) lets you repeat a block more than once.

What is C?

* C is a medium-level, structured, procedural, imperative programming language.
* *Functions* let you encapsulate common solutions into self-contained units.
* The function can then be used at multiple locations in the program, or by other people with access to your code, without having to write it out again.

From "Procedure Call"
i.e. program uses calls to things typical called 'routines', 'subroutines', 'functions' or 'methods'

What is C?

* C is a medium-level, structured, procedural, imperative programming language.
* Statements in C are *commands* that the computer is expected to obey.
* They are obeyed, one after the other, in the order they are written, by default.

What is C?

- C is a relatively small language – does basic things.
- It's 'medium level' - produces efficient code.
- C is like a sharp knife – very useful but a bit dangerous.
- C does not protect you from yourself – experts love it because they can do the risky and obscure; Students can hate it because it seems designed to be full of booby-traps.

History of C

* The original C language was developed in the early 1970s at Bell Labs (a descendant of the BCPL language).
* Intended as a "systems programming language", for convenient low-level access to a computer, but without having to write in assembly/machine code directly.

History of C

* Until 1989, there was no official standard for C, all versions of it being based on interpretation of Kernighan and Ritchie's book "The C Programming Language".
* This version of C is often called "K&R C".
* In 1989 the first C Standard was produced by the American National Standards Institute (ANSI).
* Often called "ANSI C".

History of C

* Since then, there have been several other revisions of the Standard to add new features, or fix issues with older standards.
* We will mostly be sticking to the second Standard, C99 (released in 1999 by the ISO).
* (There are later standards, but C99 has good support in all modern compilers.)

C Program Structure

Programs Contains Data and Instructions

```
#include <stdio.h>

     int main()
     {
          int i, j;
          i = 3;
          j = i * 2;
          printf("%d doubled is %d\n",i, j);
          return 0;
     }
```

Starts with one or more standard lines which tells the linker where to find information.

Program consists of one (or more) functions.

Function composed in terms of:
DECLARATIONS and STATEMENTS
(which end with a semi-colon!)

" j = i * 2;" is an expression statement, composed of:
Variables (i and j);
*Operators (= and *);*
Literal (2).

"printf" is a function-call statement.

Return code is either '0' (success) or non-zero (error-code – failure); can be used by calling script.

Compiling a Program

Under Linux command-line compilers are most common (as oppose to the interfaces you get, say, on Windows machines).

The following command compiles a program contained in a file called myProg.c

```
> gcc  -o myProg  myProg.c
```

which tells the compiler to read, compile, and link the source file myProg.c and write the executable code into the output file myProg

The program is then run on a linux machine using the command:

```
> ./myProg
```

Where the ./ tells it to look in your 'present working directory' (in case this is not in your 'path' – refer to Linux lectures).

Literal values

* A literal is an explicit value written into the C source code itself.
  - `99` Integer type
  - `'a'` Character type
  - `5.67` Floating point number - float type
* All values in C, including literals, have a *type* that defines what kind of value they are.

Types

* The type of a value determines how the computer interprets the pattern of bits that represent it in memory.
* Types also determine how many bits are needed to store the value in memory.
* C has several built-in types

  …e.g. Integer, character, floating-point number…
* C also allows you to define "composite" types derived from those built-in types.

Computer Memory

➢The smallest bit of information stored by a computer is a 1 or 0 - this is called *a **bit*** and, therefore, the natural number systems for computers, at the lowest level, is binary.

1

Integer Types

* Representations of integer values.
* Several different types, for different memory use (and consequent max and min representable values).
* Each type also has a signed and an unsigned (positive only) subtype (default to signed).

Types on Intel 64bit	`short (unsigned short)`	`int (unsigned int)`	`long (unsigned long)`
Max value	32767 (65535)	2,147,483,647 (4294967296)	2^{63}-1 (2^{64}-1)
Min value	-32768 (0)	-2,147,483,648 (0)	-2^{63} (0)
Size (bytes)	2	4	8

E.g. unsigned int i = 321;

Floating-Point Types

- Limited precision representations of real numbers.
- http://moodle2.gla.ac.uk/mod/page/view.php?id=149958

Types on Intel 64bit	`float`	`double`	`long double`
Max value	$\pm 3.4 \times 10^{38}$	$\pm 2.2 \times 10^{308}$	$\pm 1.18 \times 10^{4932}$
Smallest non-zero value	$\pm 1.18 \times 10^{-38}$	$\pm 2.2 \times 10^{-308}$	$\pm 3.65 \times 10^{-4951}$
Decimal precision	7 sig. fig.	16 sig. fig.	19 sig. fig.
Size (bytes)	4	8	16 (often)

Other Types

- For non-numeric data, there are a small set of other types.
- `char`s are single characters of text (the symbol '9' is distinct from the number 9). Internally they are represented as *numbers*, indices into a table of symbols.
- This means that we can 'add' values to `char`s, to get the value that many spaces along in the table (`'a' + 1` is `'b'`).

Types on Intel 64bit	`void`	`char`
Kind of data	"No type": used to indicate explicitly that there is no type or value expected.	Single characters (`'a'`, `'G'`, `'9'`, `':'`)
Size (bytes)	N/A (effectively 1)	1 (by standard)

e.g. char ch1 = 'a';

The quotes ' ' are a flag to the compiler to interpret contents as a character.

ASCII: American Standard Code for Information Interchange.

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

Other ASCII Characters

- We can include characters which are not printable (or, like the character ', can't be written in as a literal without terminating the quotes) using the \ symbol to 'escape' them.
- So, `'\''` is the ' as a `char`, `'\n'` is the result of pressing enter, `'\t'` is a tab space and `'\\'` is an actual \ as a character.
- Some more special types will be covered later in the course.

This will make more sense when we look in more detail at the printf() function, used to print out characters to the screen. The escape \ is basically another flag to the compiler to tell it how to interpret the subsequent character.

Operators

* Operators are built-in methods for manipulating values.
* C provides a small set of operators for arithmetic, logical and utility purposes, most of which translate directly to operations that the CPU itself has built-in.
* We will spend some slides listing the most relevant ones to this course (there are a few others that you can look up, and some we will introduce later on).
* Just as with mathematical operators, operators in C have precedence rules to resolve compound expressions like `2*3+4` . We can always put parentheses around a subexpression that we want to force to be calculated first, to override precedence: `2*(3+4)`.

Arithmetic Operators

* The Arithmetic Operators work on any data type that is numeric.
* They are defined to work within the type of the values they are given - division of two integer type value produces the nearest *integer* value as the result!
* Mixing two types in an operation (e.g. `1.2 * 15`) performs the calculation as if both values had the “widest” type in the operation (that is, the type that can store the largest values). Usually, this means that you'll get a floating-point calculation if you mix floats with ints.

Operation	Symbol	Precedence	Example
Addition	+	Low	2+4 (=6)
Subtraction	-	Low	2-4 (= -2 if signed type)
Multiplication	*	High	2*4 (=8)
Division	/	High	2/4 (=0 if integer type!)
Remainder	%	High	2%4 (=2)

Relational Operators

* Relational operators compare two values, and return a true (1) or false (0) result dependant on if the given relation is the case.
* In general, “non-zero” is taken to mean “true” for the purpose of assigning truth or falsity to integers.

Operation	Symbol	Precedence	Example
Equality	==	Low	2==4 (false)
Greater Than	>	Low	2>4 (false)
Less Than	<	Low	2<4 (true)
Greater than or equal	>=	Low	2>=4 (false)
Less than or equal	<=	Low	2<=4 (true)
Not equal	!=	Low	2!=4 (true)

E.g. if (x == 4) “do something”

Logical & Misc. Operators

* Logical operators combine two true or false values using Boolean logic.
* Note that, as all non-zero integer values are considered `true`, these operators can be used on integer types as well as the logical values themselves.
* The `sizeof` operator returns the amount of bytes used to represent the value in memory. Often used in code which needs to run on many different types of machine (where the size of, say, `int` might be different).

<table>
<tr><th>Operation</th><th>Symbol</th><th>Precedence</th><th>Example</th></tr>
<tr><td>Logical AND</td><td>&&</td><td>Low</td><td>1 && 0 (0)</td></tr>
<tr><td>Logical OR</td><td>||</td><td>Low</td><td>1||0 (1)</td></tr>
<tr><td>Logical NOT</td><td>!</td><td>Low</td><td>!1 (0)</td></tr>
<tr><td>Number of bytes to represent value</td><td>sizeof()</td><td>Highest</td><td>sizeof(321) (4)</td></tr>
</table>

Casts: Converting types

* As mentioned above, some operators automatically change the type of their operands so that they all match in type.
* We can also explicitly change the type of a value using a `cast`.
* Specifying the name of a type surrounded by `( )` tells the compiler to convert the following value to the indicated type.
* For example, we can cast a `float` to a `double` like so:

```
myDouble = (double) myfloat;
```

* Obviously, casting a value to a type which does not support it (casting to int a value bigger than the largest value an int can store, for example) will cause a loss of information or unexpected effects.