Introduction to Programming

C – Fundamentals, Data Types, Operators and Expressions

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

December, 2020

The C compiler

Source code $\rightarrow$ **(C compiler)** $\rightarrow$ **Object code**

Object code $\rightarrow$ **(C linker)** $\rightarrow$ **Executable**

Standard conformance of C

Significant Features	C89	C99	C11
Implicit function declaration	Yes	No	No
IEEE 754 floating point support	No	Yes	Yes
Inline functions	No	Yes	Yes
long long int	No	Yes	Yes
complex type, complex.h	No	Yes	Yes
variadic macros	No	Yes	Yes
gets	Yes	Yes	No
Alignment specification	No	No	Yes
No return specification, stdnoreturn.h	No	No	Yes
Type generic expressions	No	No	Yes
Multi-threading, thread.h	No	No	Yes
Bounds checking	No	No	Yes

Note: C99 and C11 are formally known as ISO/IEC 9899:1999 and ISO/IEC 9899:2011, respectively.

The first C program

Source: Welcome2C.c

The first C program

Source: Welcome2C.c

```
#include<stdio.h>
int main(){
    printf("Welcome 2 C");
    return 0;
}
```

The first C program

Source: Welcome2C.c

```
#include<stdio.h>
int main(){
    printf("Welcome 2 C");
    return 0;
}
```

Compilation: gcc Welcome2C.c

The first C program

Source: Welcome2C.c

```
#include<stdio.h>
int main(){
    printf("Welcome 2 C");
    return 0;
}
```

Compilation: gcc Welcome2C.c

"Welcome2C.c" is compiled and the executable "a.out" is created.

The first C program

Source: Welcome2C.c

```
#include<stdio.h>
int main(){
    printf("Welcome 2 C");
    return 0;
}
```

Compilation: gcc Welcome2C.c

"Welcome2C.c" is compiled and the executable "a.out" is created.

Execution: ./a.out

The first C program

Source: Welcome2C.c

```
#include<stdio.h>
int main(){
    printf("Welcome 2 C");
    return 0;
}
```

Compilation: gcc Welcome2C.c

"Welcome2C.c" is compiled and the executable "a.out" is created.

Execution: ./a.out

Welcome 2 C**(cursor here!!!)**

Dissecting the code

```
#include<stdio.h>
int main(){
    statement;
    function1(argument);
    statement;
    return 0;
}
int function1(int arg){
    function2();
    statement;
}
void function2(){
    statement;
}
```

Dissecting the code

```
#include<stdio.h>
int main(){
    statement;
    function1(argument);
    statement;
    return 0;
}
int function1(int arg){
    function2();
    statement;
}
void function2(){
    statement;
}
```

Note: The program name can be anything.

Basic characteristics

- A C program should have at least the main() function.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.
- Every statement in C must end with a semicolon.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.
- Every statement in C must end with a semicolon.
- C is case sensitive.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.
- Every statement in C must end with a semicolon.
- C is case sensitive.
- C is a free-form language.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.
- Every statement in C must end with a semicolon.
- C is case sensitive.
- C is a free-form language.
- C is a loosely typed language.

Basic characteristics

- A C program should have at least the main() function.
- There can be at most a single main() function in the program.
- Every statement in C must end with a semicolon.
- C is case sensitive.
- C is a free-form language.
- C is a loosely typed language.
- C is a middle level language.

Free-form language

A C statement can be broken into multiple lines.

Free-form language

A C statement can be broken into multiple lines. E.g., the statements

```
printf("Welcome 2 C");
```

and

```
printf
("Welcome 2 C");
```

are both same producing the same output.

Loosely typed language

```
char c;
int i;
float f;
```

Loosely typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become flexible. They can be used interchangeably.

Loosely typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become flexible. They can be used interchangeably. E.g., assigning "c = 70" is same as assigning "c = 'F'".

Loosely typed language

```
char c;
int i;
float f;
```

Here, the data types of c, i, f become flexible. They can be used interchangeably. E.g., assigning "c = 70" is same as assigning "c = 'F'".

Note: Type casting is still there for converting the data types.

In-built types

Integer data types

Type	Size*	Minimum	Maximum
char	8 bits	-2^7	$2^7 - 1$
short int	16 bits	-2^{15}	$2^{15} - 1$
int	32 bits	-2^{31}	$2^{31} - 1$
long int	32 bits	-2^{31}	$2^{31} - 1$
long long int	64 bits	-2^{63}	$2^{63} - 1$
unsigned char	8 bits	0	$2^8 - 1$
unsigned short int	16 bits	0	$2^{16} - 1$
unsigned int	32 bits	0	$2^{32} - 1$
unsigned long int	32 bits	0	$2^{32} - 1$
unsigned long long int	64 bits	0	$2^{64} - 1$

* Dependent on machine configuration.

In-built types

"Real" (floating point) numbers

Type	Size
float	32
double	64
long double	128

Examples:

```
1.23456     3.45e67
1.          +3.45e67
.1          -3.45e-67
-0.12345    .00345e-32
+.4560      1e-15
```

In-built types

"Real" (floating point) numbers

Type	Size
float	32
double	64
long double	128

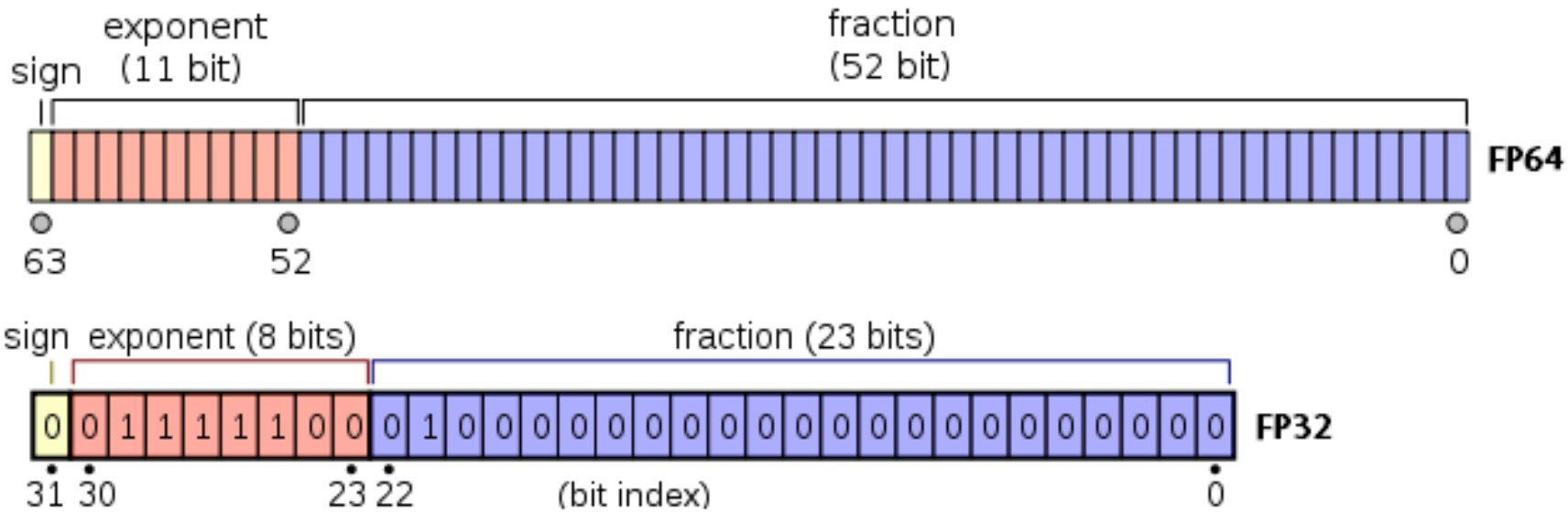
Examples:

```
1.23456     3.45e67
1.          +3.45e67
.1          -3.45e-67
-0.12345    .00345e-32
+.4560      1e-15
```

- Do not use commas as thousand-separators.
- At times behavior may be counter-intuitive (more about this later).

IEEE 754 floating point support

IEEE standard for floating point arithmetic (IEEE 754) is a technical standard for floating point representation on digital computers.



Boolean values

Any non-zero value is treated as **TRUE** and zero is treated as **FALSE**.

Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	'\0'	False
(6 - 2) * 3	True	x = 0	False
0.0075	True	x = 1	True

Type casting

```
int a = 5, b = 2;
float d;
d = a/b;
printf("Division result = %f",d);
```

Type casting

```
int a = 5, b = 2;
float d;
d = a/b;
printf("Division result = %f",d);
```

Division result = 2.000000

Type casting

```
int a = 5, b = 2;
float d;
d = (float)a/b;
printf("Division result = %f",d);
```

Type casting

```
int a = 5, b = 2;
float d;
d = (float)a/b;
printf("Division result = %f",d);
```

Division result = 2.500000

Escape sequences

How can we print the following using print statements?

', ", \

Escape sequences

How can we print the following using print statements?
', ", \

What will be the output of following?

```
char *title = "Bhattacharyya";
printf("This is Malay \r%s",title);
```

Escape sequences

How can we print the following using print statements?
', ", \

What will be the output of following?

```
char *title = "Bhattacharyya";
printf("This is Malay \r%s",title);
```

Bhattacharyya

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Division $(/)$

Modulo division $(\%)$

Arithmetic operators

Summation $(+)$

Subtraction $(-)$

Multiplication $(*)$

Division $(/)$

Modulo division $(\%)$

Note: Modulo division operator works on the integers (negative too!!!) only and returns the sign of the numerator.

Relational operators

Less than $(<)$

Less than equals to $(<=)$

Greater than $(>)$

Greater than equals to $(>=)$

Equals to $(==)$

Not equals to $(!=)$

Logical operators

Logical and (&&)

Logical or (||)

Logical not (!)

Assignment operators

Assignment $(=)$

Increment and decrement operators

Increment $(++)$

Decrement $(--)$

Increment and decrement operators

```
int i = 5;
i++;
printf("%d", i);
printf("\n%d", i++); // Post-increment operation
printf("\n%d", ++i); // Pre-increment operation
```

Increment and decrement operators

```
int i = 5;
i++;
printf("%d", i);
printf("\n%d", i++); // Post-increment operation
printf("\n%d", ++i); // Pre-increment operation
```

```
6
6
8
```

Conditional operators

Ternary (? :)

```
int i = 0;
int j = 1;
int result = (i > j)? i : j;
```

Bitwise operators

Bitwise and $(\&)$

Bitwise or $(|)$

Bitwise not $(\sim)$

Bitwise xor $(\wedge)$

Left shift $(<<)$

Right shift $(>>)$

Bitwise operators

```
int i = 11;
printf("%d", i>>1); // 1 place right shift (i unchanged)
printf("\n%d", i<<2); // 2 places left shift (i unchanged)
```

Bitwise operators

```
int i = 11;
printf("%d", i>>1); // 1 place right shift (i unchanged)
printf("\n%d", i<<2); // 2 places left shift (i unchanged)
```

```
5
44
```

Bitwise operators

```
int i = 11;
printf("%d", i>>1); // 1 place right shift (i unchanged)
printf("\n%d", i<<2); // 2 places left shift (i unchanged)
```

5
44

Decimal	Binary							
11	0	0	0	0	1	0	1	1
5	0	0	0	0	0	1	0	1
44	0	0	1	0	1	1	0	0

Operator precedence

What will be the output of the following program?

```
#include<stdio.h>
int main(){
    int n = 10;
    n = 20, 30, 40;
    printf("First n = %d\n",n);
    n = (50, 60, 70);
    printf("Second n = %d\n",n);
    return 0;
}
```

Operator precedence

What will be the output of the following program?

```
#include<stdio.h>
int main(){
    int n = 10;
    n = 20, 30, 40;
    printf("First n = %d\n",n);
    n = (50, 60, 70);
    printf("Second n = %d\n",n);
    return 0;
}
```

First n = 20
Second n = 70

Operator precedence

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right