Data Types In ‘C’

Data Types In ‘C’
Data Types In ‘C’
Data Types In ‘C’

Course Title :- Structured Programming Language Sessional

Course Code :- CSE-122

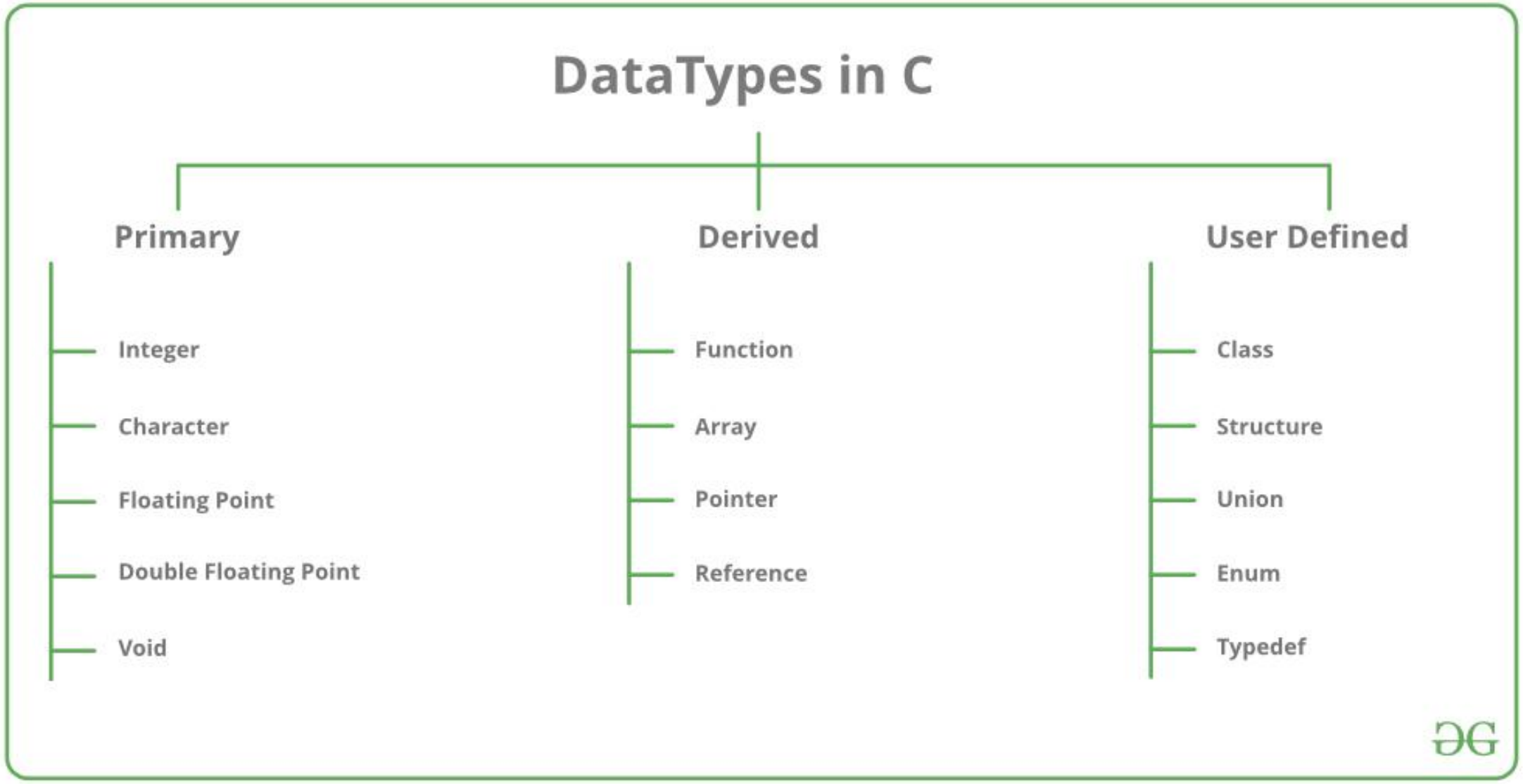
Level Term: 1-II-A(G1) & 1-II-B(G3,G4)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

Topics in Data types

- Data types
  1. primitive data type
  2. derived data type
  3. user defined data type
- Basic format specifiers
- The sizeof() operator
- Character Data types
- Integer Data types
- Float Data types
- Double Data types
- Boolean in C
- Type conversion in C
  1. implicit type conversion
  2. explicit type conversion

Data Types


DataTypes in C
Primary
Integer
Character
Floating Point
Double Floating Point
Void
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

Data Type	Size (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

Size of Data Types in C

```
#include<stdio.h>
int main()
{
  int i;
  float f;
  double d;
  char c;
  char ch[100];

  printf("size of int  = %d bytes\n", sizeof(i));
  printf("size of int  = %d bytes\n", sizeof(f));
  printf("size of int  = %d bytes\n", sizeof(d));
  printf("size of int  = %d bytes\n", sizeof(c));
  printf("size of int  = %d bytes\n", sizeof(ch));

}
```

Output:

```
size of int  = 4 bytes
size of int  = 4 bytes
size of int  = 8 bytes
size of int  = 1 bytes
size of int  = 100 bytes
```

Integer Data Type

The integer datatype in C is used to store the integer numbers(any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

Range: -2,147,483,648 to 2,147,483,647

Size: 4 bytes

Format Specifier: %d

Syntax of Integer

```
int var_name;
```

```
#include <stdio.h>
int main() {
  int myNum = 1000;
  printf("%d", myNum);
}
```

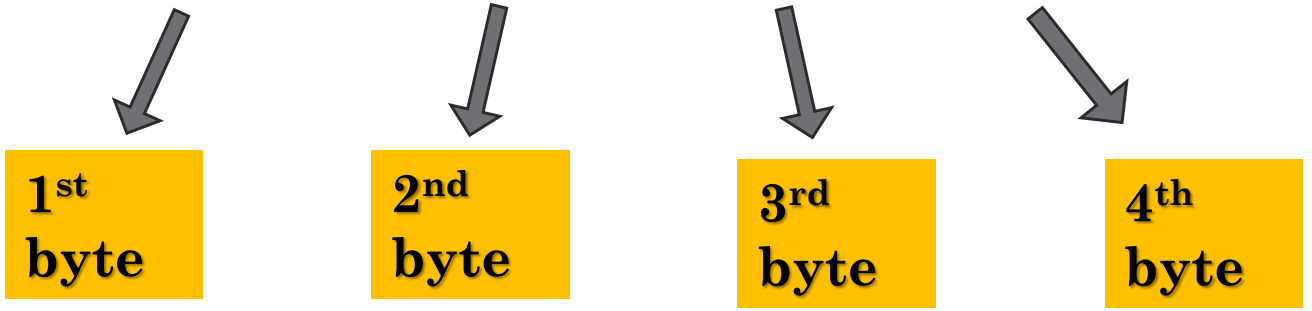
Integer size 4bytes

1 byte = 8bit

4 byte = 8 * 4 bit = 32bit

If n=3

Memory will be store it as:- 00000000 00000000 00000000 00000011



Character Data Type

- Character data type allows its variable to store only a single character. The size of the character is 1 byte.
- It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **Range:** (-128 to 127) or (0 to 255)
- **Size:** 1 byte
- **Format Specifier:** %c

Syntax of char
The **char keyword** is used to declare the variable of character type:
char *var_name;*

1	▯	17	▯	33	!	49	1	65	A	81	Q	97	a	113	q
2		18	▯	34	"	50	2	66	B	82	R	98	b	114	r
3	▯	19	▯	35	#	51	3	67	C	83	S	99	c	115	s
4	▯	20	▯	36	$	52	4	68	D	84	T	100	d	116	t
5	▯	21	▯	37	%	53	5	69	E	85	U	101	e	117	u
6	▯	22	▯	38	&	54	6	70	F	86	V	102	f	118	v
7	▯	23	▯	39	'	55	7	71	G	87	W	103	g	119	w
8	▯	24	▯	40	(	56	8	72	H	88	X	104	h	120	x
9		25	▯	41	)	57	9	73	I	89	Y	105	i	121	y
10		26	▯	42	*	58	:	74	J	90	Z	106	j	122	z
11	▯	27	▯	43	+	59	;	75	K	91	[	107	k	123	{
12	▯	28		44	,	60	<	76	L	92	\	108	l	124	|
13		29		45	-	61	=	77	M	93	]	109	m	125	}
14	▯	30		46	.	62	>	78	N	94	^	110	n	126	~
15	▯	31		47	/	63	?	79	O	95	_	111	o	127	▯
16	▯	32		48	0	64	@	80	P	96	`	112	p	128	€

Character → ASCII → Character

```
#include<stdio.h>
int main(){
   /// character --> ascii value
   char ch;
   printf(" Enter a ascii characters: ");
   scanf("%c", &ch);
   printf(" corresponding value : %d\n\n", ch);
   ///ascii value --> character
   int a;
   printf(" Enter a value: ");
   scanf("%d", &a);
   printf(" corresponding ascii character : %c\n", a);

}
```

```
Enter a ascii characters: T
corresponding value : 84

Enter a value: 83
corresponding ascii character : S
```

Lowercase → Uppercase → Lowercase

```
#include<stdio.h>
int main(){
    ///lower to upper
    char lower;
    printf("Enter a letter between (a-z): ");
    scanf("%c", &lower);
    printf("uppercase method-1 : %c\n", (lower - 32));
    printf("uppercase method-2 : %c\n\n", toupper(lower));
    getchar();
    ///lower to upper
    char upper;
    printf("Enter a letter between (A-Z): ");
    scanf("%c", &upper);
    printf("uppercase method-1 : %c\n", (upper + 32));
    printf("uppercase method-2 : %c\n\n", tolower(upper));
}
```

```
Enter a letter between (a-z): a
uppercase method-1 : A
uppercase method-2 : A

Enter a letter between (A-Z): R
uppercase method-1 : r
uppercase method-2 : r
```

Float Data Type

- In C programming float data type is used to store floating-point values. Float in C is used to store decimal and exponential values. It is used to store decimal numbers (numbers with floating point values) with single precision.
- **Range:** 1.2E-38 to 3.4E+38
- **Size:** 4 bytes
- **Format Specifier:** %f

Syntax of float

The **float keyword** is used to declare the variable as a floating point:

```
float var_name;
```

```
#include <stdio.h>
int main()
{
    float a = 9.0;
    printf("%f\n", a);
}
```

Output

```
9.000000
```

Double Data Type

- A Double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It is used to define numeric values which hold numbers with decimal values in C.
- The double data type is basically a precision sort of data type that is capable of holding 64 bits of decimal numbers or floating points. Since double has more precision as compared to that float then it is much more obvious that it occupies twice the memory occupied by the floating-point type. It can easily accommodate about 16 to 17 digits after or before a decimal point.
- **Range:** 1.7E-308 to 1.7E+308
- **Size:** 8 bytes
- **Format Specifier:** %lf

```
#include <stdio.h>
int main()
{
    double b = 12.293123;
    printf("%lf\n", b);
}
```

Syntax of Double

The variable can be declared as double precision floating point using the **double keyword:**

```
double var_name;
```

Output

```
12.293123
```

Double Data Type

float vs. double

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore, it is often safer to use double for most calculations - but note that it takes up twice as much memory as float (8 bytes vs. 4 bytes).

Scientific Numbers

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

```
#include <stdio.h>
int main()
{
  float f1 = 35e3;
  printf("%f\n", f1);
}
Output:
35000
```

C Decimal Precision

If you want to remove the extra zeros (set decimal precision), you can use a dot (.) followed by a number that specifies how many digits that should be shown after the decimal point:

```
#include <stdio.h>
int main()
{
  float myFloatNum = 3.5;
  printf("%f\n", myFloatNum); // Default will show 6 digits after the decimal point
  printf("%.1f\n", myFloatNum); // Only show 1 digit
  printf("%.2f\n", myFloatNum); // Only show 2 digits
  printf("%.4f", myFloatNum);   // Only show 4 digits
}
```

Boolean in C

C Boolean

In C, Boolean is a data type that contains two types of values, i.e., 0 and 1. Basically, the bool type value represents two types of behavior, either true or false. Here, '0' represents false value, while '1' represents true value.

In C Boolean, '0' is stored as 0, and another integer is stored as 1. We do not require to use any header file to use the Boolean data type in C++, but in C, we have to use the header file, i.e., stdbool.h. If we do not use the header file, then the program will not compile.

Syntax

```
bool variable_name;
```

Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

YES / NO

ON / OFF

TRUE / FALSE

For this, C has a bool data type, which is known as booleans.

Booleans represent values that are either true or false.

Boolean in C

Therefore, you must use the **%d** format specifier to print a boolean value:

```
#include <stdio.h>
#include <stdbool.h>  // Import the boolean header file
int main()
{
  bool isProgrammingFun = true;
  bool isFishTasty = false;
  printf("%d\n", isProgrammingFun);  // Returns 1 (true)
  printf("%d", isFishTasty);       // Returns 0 (false)
}
```

Output:

```
1
0
```

Comparing Values and Variables

Comparing values are useful in programming, because it helps us to find answers and make decisions.
For example, you can use a comparison operator, such as the **greater than** (>) operator, to compare two values:

```
#include <stdio.h>
int main()
{
  printf("%d", 10 > 9);
 // Returns 1 (true) because 10 is greater than 9
}
Output:
1
```

```
#include <stdio.h>
int main() {
  int x = 10;
  int y = 9;
  printf("%d", x > y);
// Returns 1 (true) because 10 is greater than 9
}
Output:
1
```

Comparing Values and Variables

In the example below, we use the equal to (==) operator to compare different values:

```
#include <stdio.h>
int main() {
  printf("%d\n", 10 == 10); // Returns 1 (true), because 10 is equal to 10
  printf("%d\n", 10 == 15); // Returns 0 (false), because 10 is not equal to 15
  printf("%d", 5 == 55); // Returns 0 (false) because 5 is not equal to 55
}
```

Output:

```
1
0
0
```

```
#include <stdio.h>
#include <stdbool.h>
int main() {
  bool isHamburgerTasty = true;
  bool isPizzaTasty = true;
  printf("%d", isHamburgerTasty == isPizzaTasty);
}
```

Output: 1

Comparing Values and Variables :- Real Life Example

Let's think of a "real life example" where we need to find out if a person is old enough to vote.

In the example below, we use the >= comparison operator to find out if the age (25) is greater than OR equal to the voting age limit, which is set to 18:

```
#include <stdio.h>
int main()
{
  int myAge = 25;
  int votingAge = 18;
  printf("%d", myAge >= votingAge); // Returns 1 (true), meaning 25 year olds are allowed to vote!
}
```

Output:

```
1
```

Type Conversion in C

Type conversion in C is the process of converting one data type to another. The type conversion is only performed to those data types where conversion is possible. Type conversion is performed by a compiler. In type conversion, the destination data type can't be smaller than the source data type. Type conversion is done at compile time and it is also called widening conversion because the destination data type can't be smaller than the source data type. ***There are two types of Conversion:***

1. Implicit Type Conversion

Also known as 'automatic type conversion'.

A. Done by the compiler on its own, without any external trigger from the user.

B. Generally takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.

C. All the data types of the variables are upgraded to the data type of the variable with the largest data type.

D. It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long is implicitly converted to float).

bool → char → int → long long → float → double → long double

Sometimes, you have to convert the value of one data type to another type. This is known as type conversion.

For example, if you try to divide two integers, 5 by 2, you would expect the result to be 2.5. But since we are working with integers (and not floating-point values), the following example will just output 2:

```
#include <stdio.h>

int main() {
  int x = 5;
  int y = 2;
  int sum = 5 / 2;

  printf("%d", sum);
  return 0;
}
```

output: 2

Implicit conversion is done automatically by the compiler when you assign a value of one type to another.

For example, if you assign an int value to a float type:

Example

```
// Automatic conversion: int to float
float myFloat = 9;

printf("%f", myFloat); // 9.000000
```

As you can see, the compiler automatically converts the int value 9 to a float value of 9.000000.

This can be risky, as you might lose control over specific values in certain situations.

Example of Type Implicit Conversion

```
// An example of implicit conversion
#include <stdio.h>
int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
}
```

Output

```
x = 107, z = 108.000000
```

Especially if it was the other way around - the following example automatically converts the float value 9.99 to an int value of 9:

Example

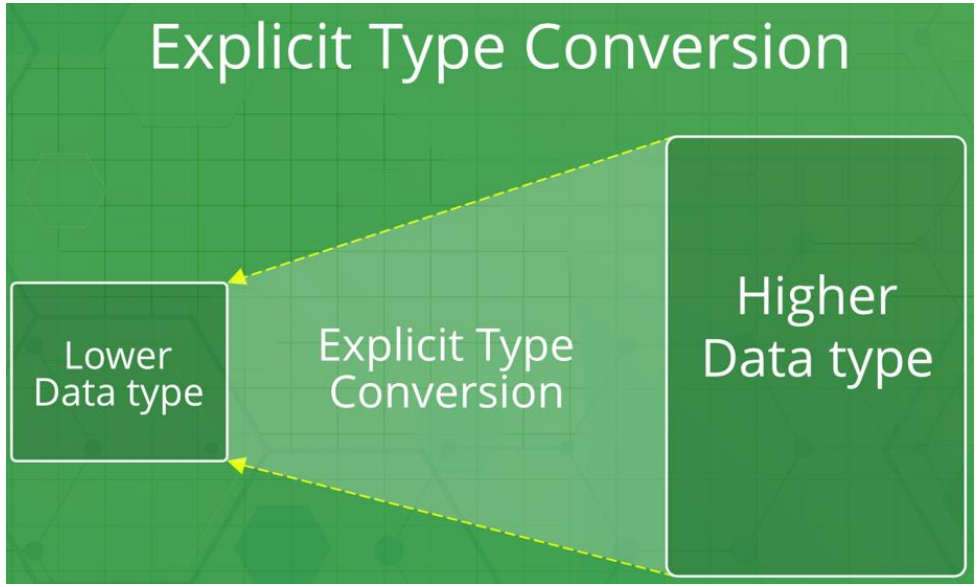
```
// Automatic conversion: float to int
int myInt = 9.99;

printf("%d", myInt); // 9
```

What happened to .99? We might want that data in our program! So be careful. It is important that you know how the compiler work in these situations, to avoid unexpected results.

2. Explicit Type Conversion

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.
The syntax in C Programming:
(type) expression
Type indicated the data type to which the final result is converted.



Explicit conversion is done manually by placing the type in parentheses () in front of the value.

Considering our problem from the example above, we can now get the right result:

Example

```
// Manual conversion: int to float
float sum = (float) 5 / 2;

printf("%f", sum); // 2.500000
```

```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
	double x = 1.2;
	// Explicit conversion from double to int
	int sum = (int)x + 1;
	printf("sum = %d", sum);
	return 0;
}
```

Output

```
sum = 2
```

You can also place the type in front of a variable:

```
int num1 = 5;
int num2 = 2;
float sum = (float) num1 / num2;

printf("%f", sum); // 2.500000
```

And since you learned about "decimal precision" in the previous chapter, you could make the output even cleaner by removing the extra zeros (if you like):

```
int num1 = 5;
int num2 = 2;
float sum = (float) num1 / num2;

printf("%.1f", sum); // 2.5
```

Advantages of Type Conversion

- **Type safety:**
- **Improved code readability**:
- **Improved performance:**
- **Improved compatibility**
- **Improved data manipulation**
- **Improved data storage**

Disadvantages of type conversions in C programming:

- **Loss of precision**: Converting data from a larger data type to a smaller data type can result in loss of precision, as some of the data may be truncated.
- **Overflow or underflow**: Converting data from a smaller data type to a larger data type can result in overflow or underflow if the value being converted is too large or too small for the new data type.
- **Unexpected behavio**r: Type conversions can lead to unexpected behavior, such as when converting between signed and unsigned integer types, or when converting between floating-point and integer types.
- **Confusing syntax**: Type conversions can have confusing syntax, particularly when using typecast operators or type conversion functions, making the code more difficult to read and understand.
- **Increased complexity**: Type conversions can increase the complexity of your code, making it harder to debug and maintain.
- **Slower performance:** Type conversions can sometimes result in slower performance, particularly when converting data between complex data types, such as between structures and arrays.

Real-life Example Of Using Different Data Types

Here's a real-life example of using different data types, to calculate and output the total cost of a number of items:

```
#include <stdio.h>
int main() {
  // Create variables of different data types
  int items = 50;
  float cost_per_item = 9.99;
  float total_cost = items * cost_per_item;
  char currency = '$';

  // Print variables
  printf("Number of items: %d\n", items);
  printf("Cost per item: %.2f %c\n", cost_per_item, currency);
  printf("Total cost = %.2f %c\n", total_cost, currency);
}
```

Real-Life Example of type conversion

Here's a real-life example of data types and type conversion where we create a program to calculate the percentage of a user's score in relation to the maximum score in a game:

```
#include <stdio.h>
int main() {
  // Set the maximum possible score to 500
  int maxScore = 500;
  // The actual score of the user
  int userScore = 420;
  // Calculate the percantage of the user's score in relation to the maximum available score
  float percentage = (float) userScore / maxScore * 100.0;
  // Print the percentage
  printf("User's percentage is %.2f", percentage);
}
```

```
Output:
User's percentage is 84.00
```