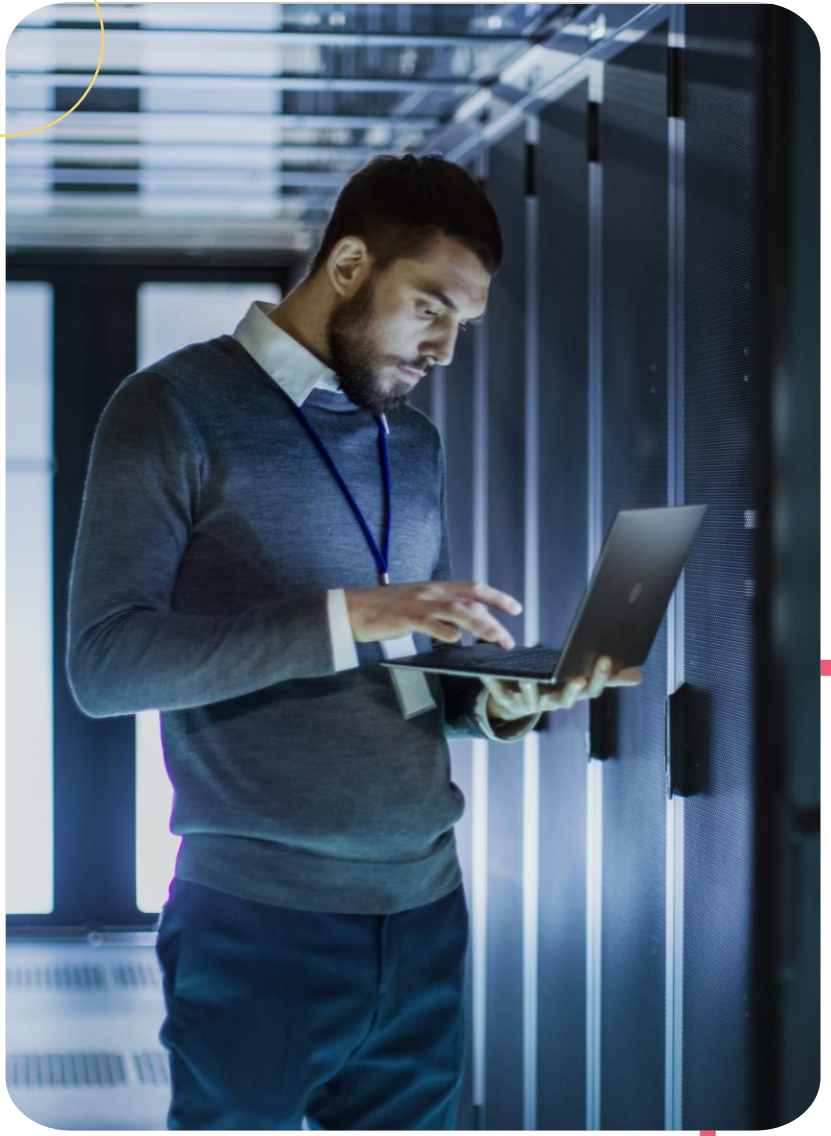
Diploma in

Computer Science

Typecasting & Advanced Data Structures



Objectives

- Explore structures and unions
- Define typecasting and type conversion
- Recall data types and their uses

Lesson 5 challenge feedback

Create an array using a pointer to traverse the array.

```
int ages[];
```

Structures

What's inside a structure?

The variables are physically grouped in the same block of memory.

You can store a number of different data types to form a record.

The computer allocates enough space for the data type you specify.

```c
struct string {
    int length;
    char *data;
};
```

Example

Computer allocates enough space to hold an int and a char.

```c
struct string {
    int length;
    char *data;
};

int
main(int argc, char **argv)
{
    struct string s;

    s.length = 4;
    s.data = "This is a long string!";

    puts(s.data);

    return 0;
```

Example

Unions

- Composite data type closely related to structs
- Members of different data types are declared but they all occupy same region of storage
- Only one member valid at a time – all members cannot co-exist
- Size of the union is size of largest member

```
union [union tag] {
    member definition;
    member definition;
    ...
    member definition;
} [one or more union variables];
```

Example

Unions save space

Data component can be one of number of data types

Create a union that only occupies the size of the largest member…

… to save memory when working with large sets of data

Typecasting

A few steps back...

Variables and constants

Each has a data type associated with it.

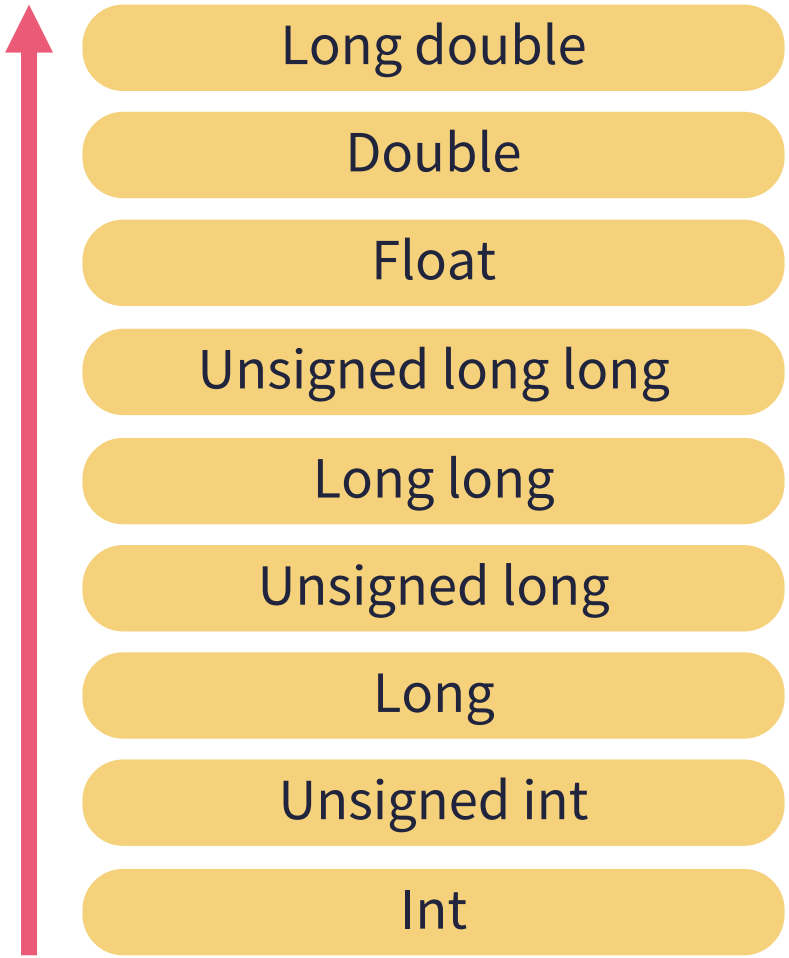
The computer can tell what kind of data is stored in the variable or constant using this data type.

When different types are put together they become the same data type.

Type conversion

- Converting one predefined type into another
- Prevents unwanted consumption of memory
- Typecast values to a data type that makes sense

Data type hierarchy



Promotion: when a data type that is lower in the hierarchy is converted to a data type higher up

Demotion: when a data type that is in a higher position in the hierarchy is converted to a data type that is lower

Two methods of type conversion in C

- Explicit type conversion
- Implicit type conversion

Explicit type conversion

The programmer poses the data type of the expression in a specific type so that data of a higher value type is converted into a lower type value.

```c
#include<stdio.h>
void main()
{
float num=3.141592654;
int integer;
integer = (int) num;
printf("Explicit value is %d",integer);
return 0;
}
```

Example

Cast the data type *int* onto num.

Fractional part has been discarded.

```c
#include<stdio.h>
void main()
{
float num=3.141592654;
int integer;
integer = (int) (num+0.5);
printf("Explicit value is %d",integer);
return 0;
}
```

Example

- Adding 0.5 will cause our floating-point number to jump into the next interval.
- Anything in parentheses is always operated on first.

Implicit type conversion

Data is automatically converted into another type by the compiler without the programmer’s involvement.

Type conversion only occurs if both the data types are compatible with each other.

Key differences...

Typecasting – conversion the user

Type conversion – conversion by the compiler

Typecasting – both data types are not compatible

Type conversion – both data types must be compatible

Typecasting – requires ()

Type conversion – does not require ()

Typecasting – done while writing code

Type conversion – happens during compilation

Implicit typecasting

- Conversion of data type does not alter original value of the data
- Happens automatically when a value is copied into a compatible data type

if statement for conversion

All short and char are automatically converted to int

If either of the operands is of type long double, then others will be converted to long double and result will be long double

Else, if either of the operand is double, then others are converted to double

Else, if either of the operand is float, then others are converted to float

Else, if either of the operand is unsigned long int, then others will be converted to unsigned long Int

if statement for conversion

Else, if one of the operands is long int, and the other is unsigned int

If a long int can represent all values of an unsigned int, the unsigned int is converted to long int

otherwise, both operands are converted to unsigned long int

Else, if either operand is long int then other will be converted to long int

Else, if either operand is unsigned int then others will be converted to unsigned int.

Implicit conversion

- Does not require any special input from the programmer
- Conversion happens whenever the compiler encounters compatible data types
- Causes compilation warnings

Built-in type conversion

Built-in functions

atof() converts a string to float

atoi() converts a string to int

atol() converts a string to long

itoa() converts int to string

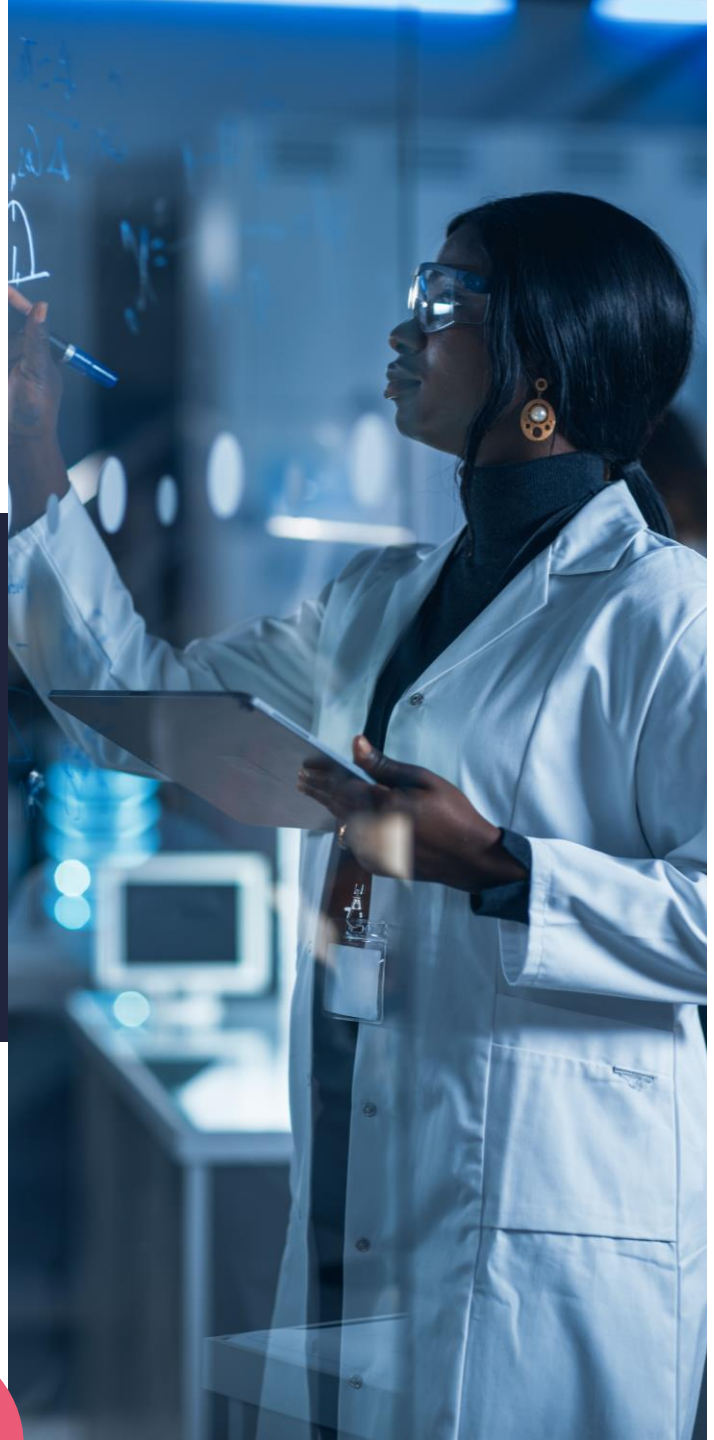
ltoa() converts long to string

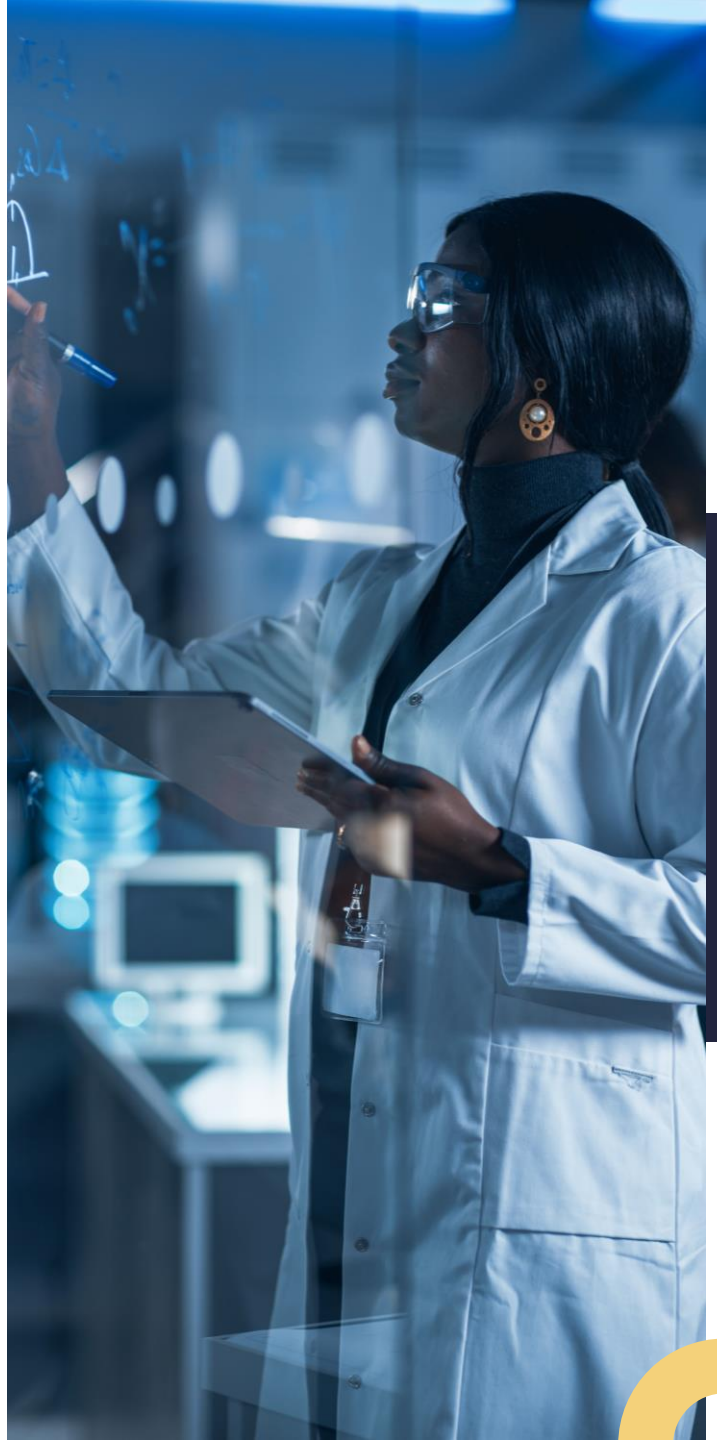
atof() converts a string to float

```
double atof (const char* string);
```

Syntax

The function returns the converted floating-point number is it value of data type double.





atoi() converts a string to int

```
int atoi(const char* string)
```

Syntax

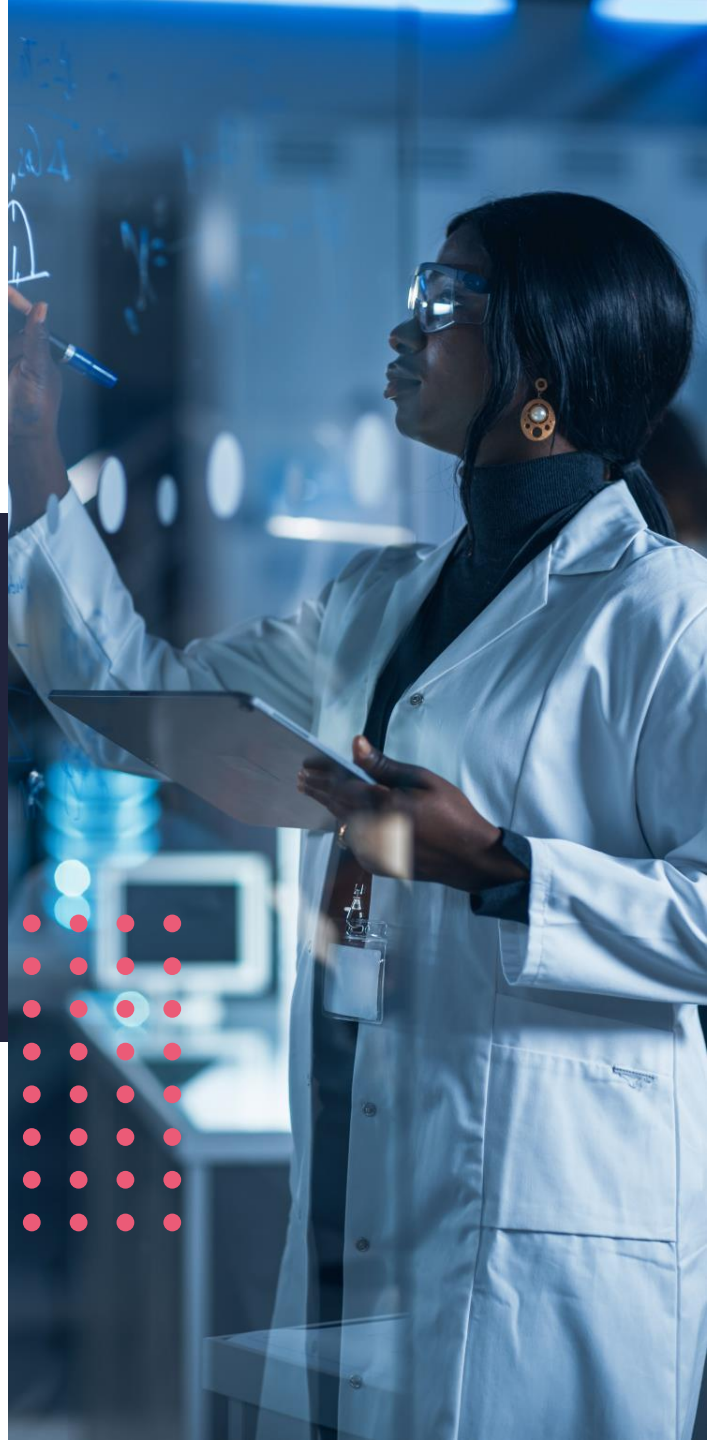
The function will return the converted value as an integer value.

atol() converts a string to long

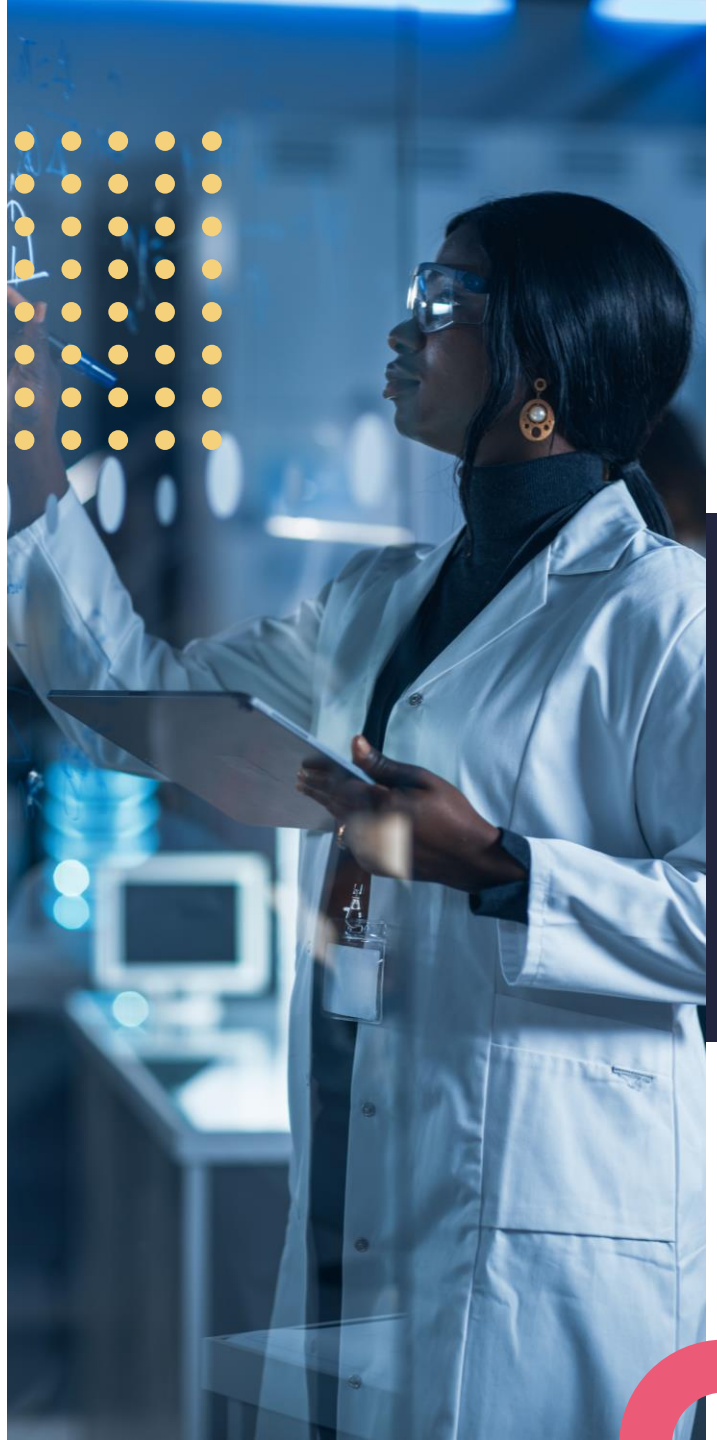
```
double atol(const char* string)
```

Syntax

The function will return the converted value as a long value.



What if you want to convert other data types to strings?



itoa() converts int to string

```
char *  itoa ( int value, char * str,
int base );
```

Syntax

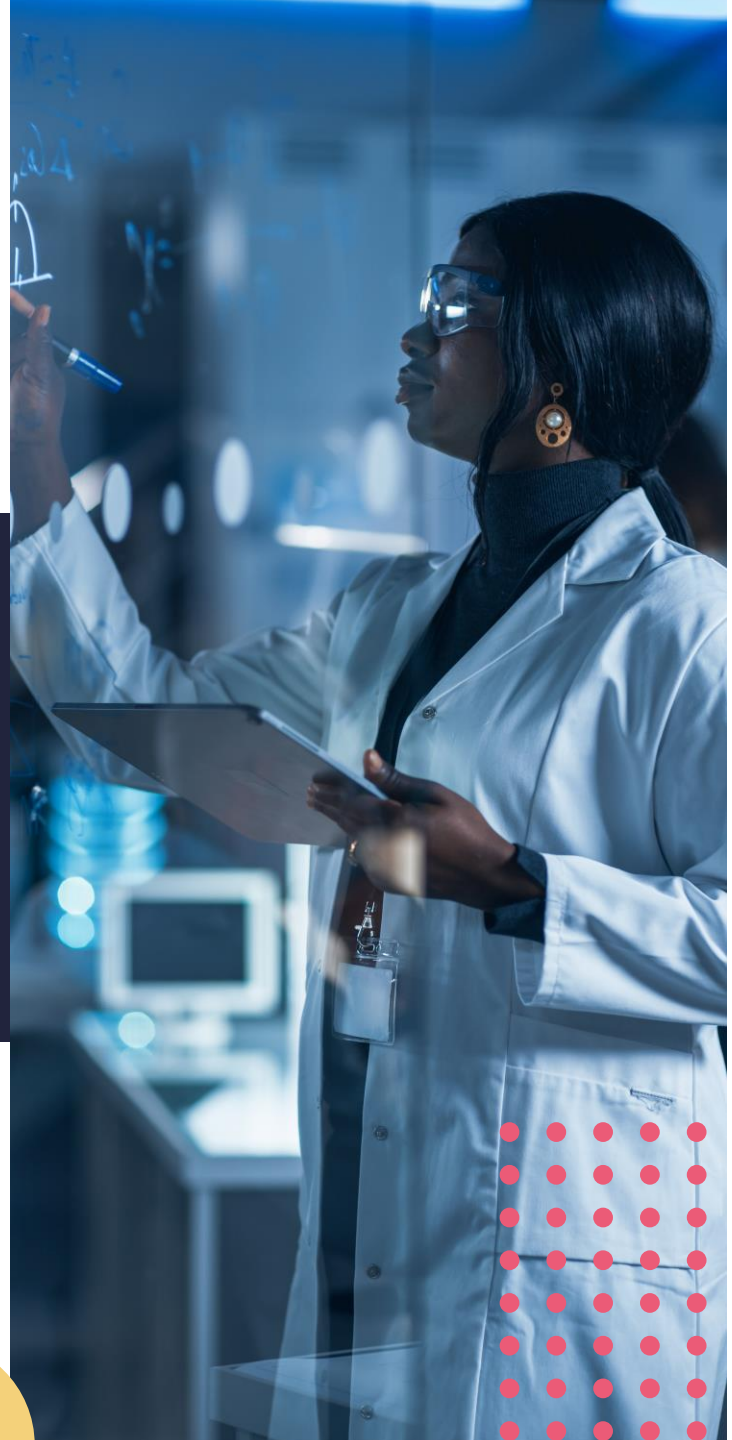
The function will return a pointer to the resulting null-terminated string.

ltoa() converts long to string

```
char* ltoa( long value, char* str, int
base);
```

Syntax

The function returns a null terminated string.



Keyword ‘typedef’

Used on user-defined data types when data type names are slightly complicated

Does not create a new data type (except in the case of a qualified typedef of an array type)

typedef simplifies the syntax of complex data structures that consist of unions and structs

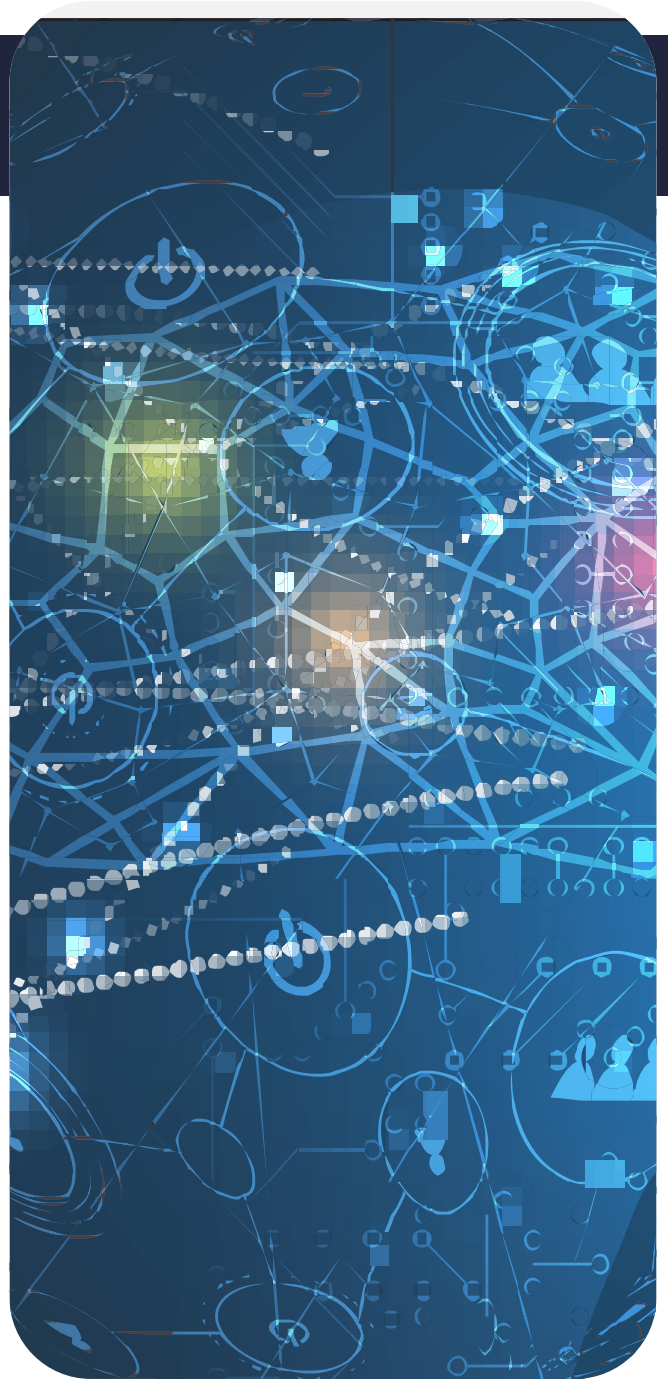
```
typedef type-definition identifier
```

Syntax

typedef

Limited to assigning symbolic names to data types

Handled by the compiler



#define

C-directive to define aliases for data types as well as values

Processed by the pre-processor