Basic Knowledge

Of

C Programming

Created By:
Prof. Arbha

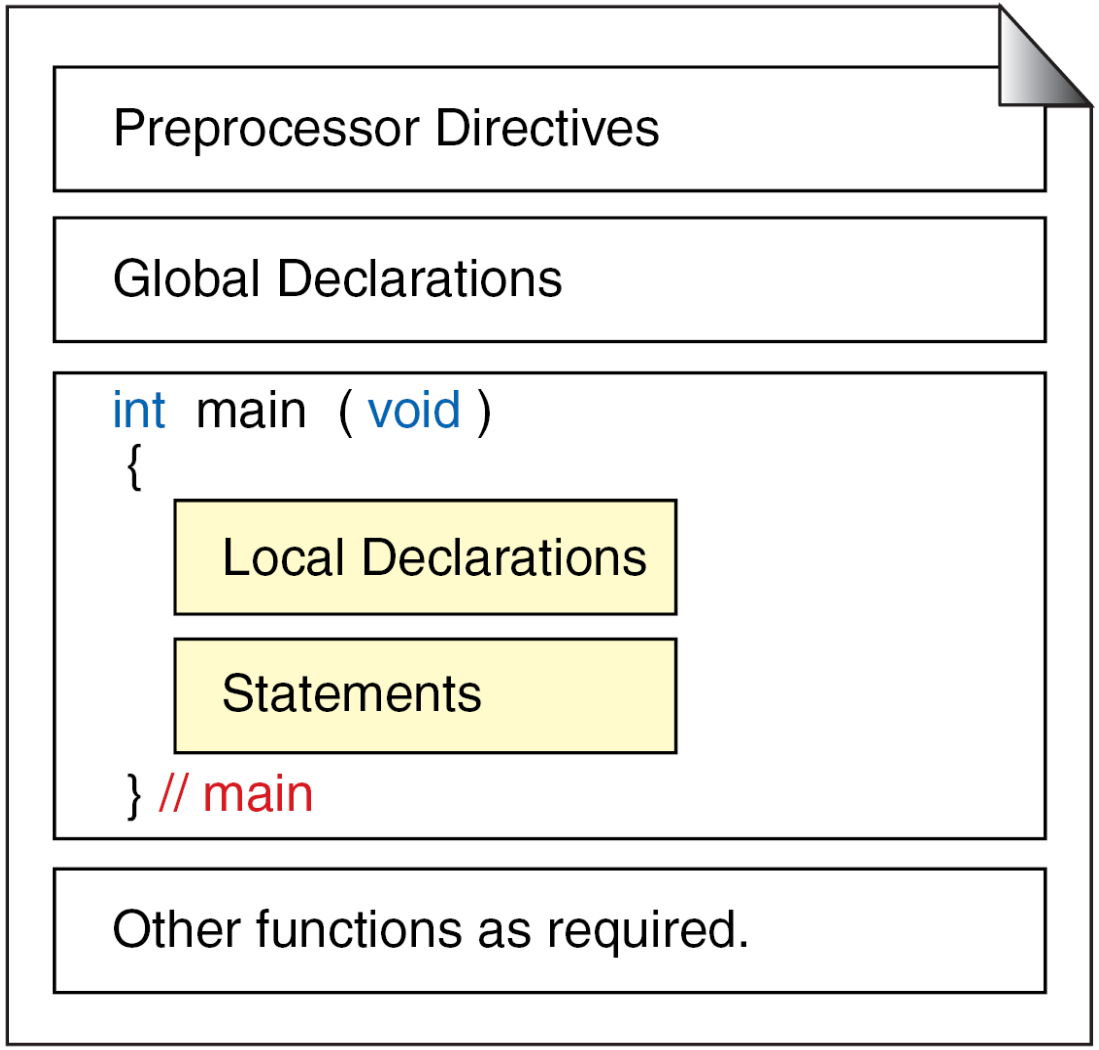
Introduction to the C Language

Objectives

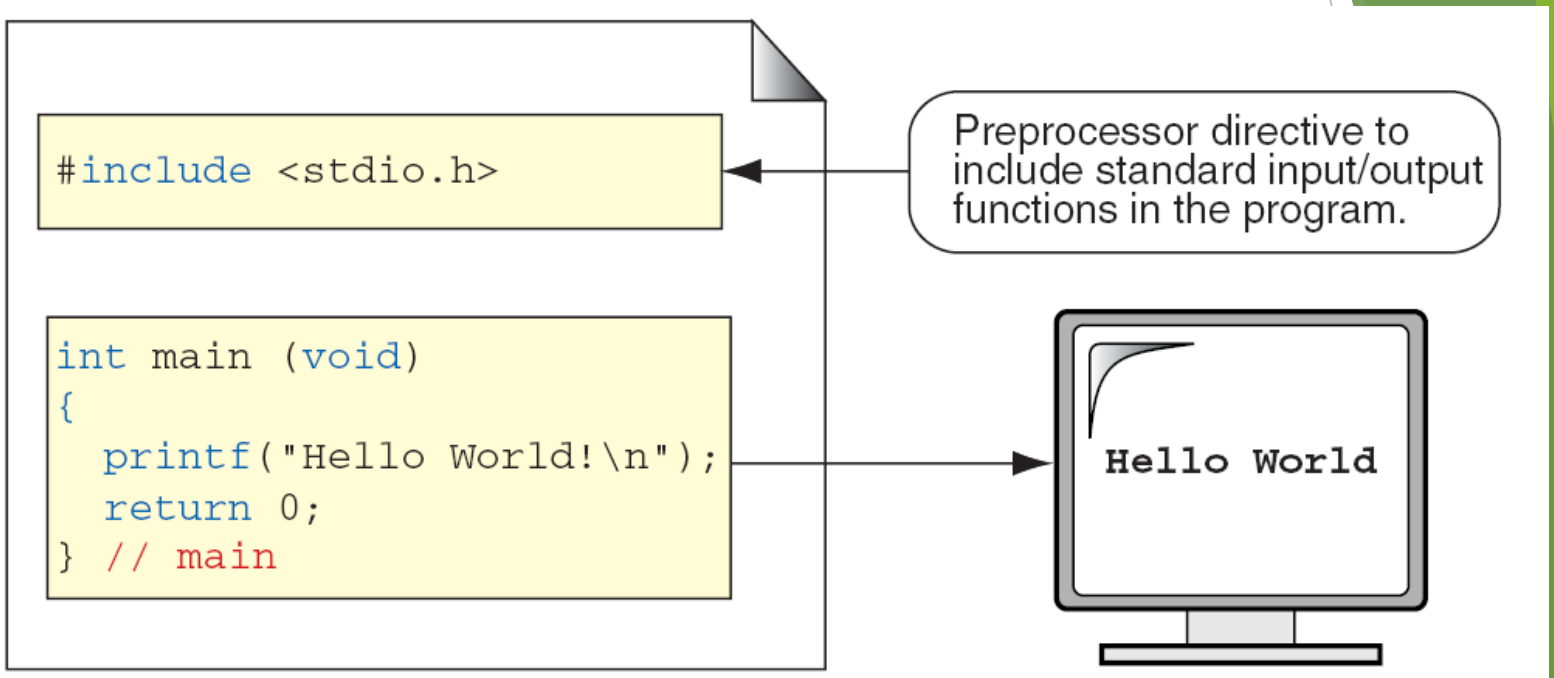
- To understand the structure of a C-language program.
- How to write simple C program.
- To introduce the include preprocessor command.
- To understand the concept of identifiers.
- To understand and use the C basic data types.
- To understand , create and use variables and constants.
- To understand input and output concepts.

Background

C is a structured programming language. It is considered a high-level language because it allows the programmer to concentrate on the problem at hand and not worry about the machine that the program will be using. That is another reason why it is used by software developers whose applications have to run on many different hardware platforms.



Structure of a C Program



Simple Program

Simple Program

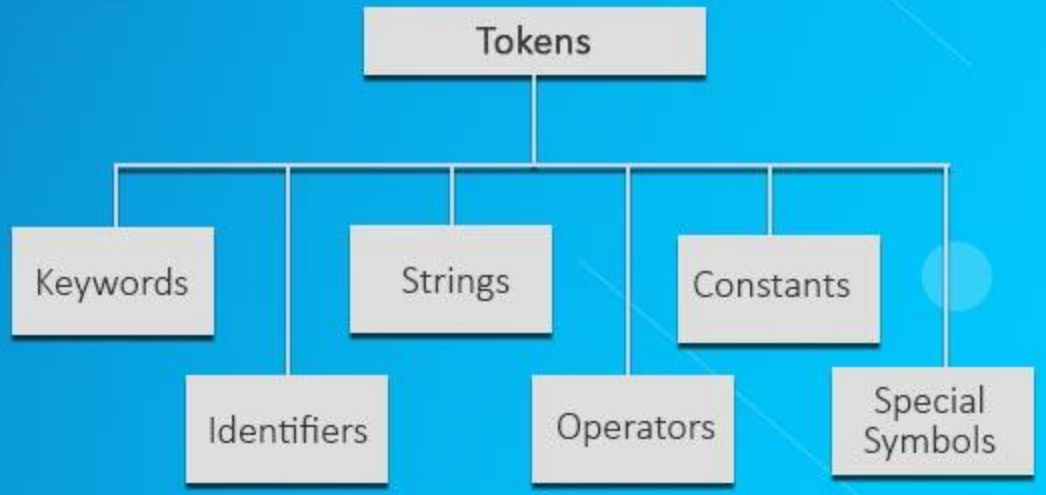
```
1  /* The greeting program. This program demonstrates
2     some of the components of a simple C program.
3        Written by:  your name here
4        Date:        date program written
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11
12 // Statements
13
14    printf("Hello World!\n");
15
16    return 0;
17 } // main
```

Token

Token is the smallest unit used to create the c program

*****There are various types of tokens available in C language

Tokens in C



www.educba.com

Identifiers

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc.

Rules

1. First character must be alphabetic character or underscore.
2. Must consist only of alphabetic characters, digits, or underscores.
3. First 63 characters of an identifier are significant.
4. Cannot duplicate a keyword.

Note

C is a case-sensitive language.

Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

Examples of Valid and Invalid Names

C Keywords

•In C programming language there are some reserved words that are used internally and have some special purpose and meaning, such words are called **Keywords** or just **Reserved Keywords**.
•The meaning and purpose of the keywords cannot be changed.
•Each Keyword is unique.
•There are only 32 keywords in C programming.
•You cannot use keywords as name for variables, functions, pointers, etc. because if you do so, the compiler will get confused and give you an error.

C Operators

•The C language supports a rich set of built-in operators.
•An **operator** is a **symbol** that tells the compiler to perform a certain operation (arithmetic, comparison, etc.) using the values provided along with the operator.

Type of C Operators

C operators can be classified into the following types:

1.Arithmetic operators
2.Relational operators
3.Logical operators
4.Bitwise operators
5.Assignment operators
6.Conditional operators
7.Special operators

What is an Operand?

- Every operator works with some values.
- The value with which an operator works is called as **Operand**.
- For example, when we say **4+5**, numbers **4** and **5** are operands whereas **+** is an operator.
- Different operators work with **different numbers of operands** like the + operator requires two operands or values, the increment operator ++ requires a single operand.

1. Arithmetic Operators in C

The C language supports all the basic arithmetic operators such as **addition**, **subtraction**, **multiplication**, **division**, etc.

The following table shows all the basic arithmetic operators along with their descriptions.

Operator	Description	Example (where a and b are variables with some integer value)
+	adds two operands (values)	a+b
-	subtract second operands from first	a-b
*	multiply two operands	a*b
/	divide the numerator by the denominator, i.e. divide the operand on the left side with the operand on the right side	a/b
%	This is the **modulus operato**r, it returns the remainder of the division of two operands as the result	a%b

2. Relational operators in C

•The relational operators (or **comparison** operators) are used to check the relationship between two operands.
•It checks whether two operands are **equal** or **not equal** or **less than** or **greater than**, etc.
•It returns **1** if the relationship checks **pass**, otherwise, it returns **0**.
•For example, if we have **two numbers 14 and 7**, if we say **14 is greater than 7**, this is **true**, hence this check will **return 1** as the result with relationship operators. On the other hand, if we say **14 is less than 7**, this is **false**, hence it will **return 0**.

The following table shows all relational operators supported in the C language.

Operator	Description	Example (a and b, where a = 10 and b = 11)
==	Check if the two operands are equal	a == b, returns 0
!=	Check if the two operands are not equal.	a != b, returns 1 because a is not equal to b
>	Check if the operand on the left is greater than the operand on the right	a > b, returns 0
<	Check operand on the left is smaller than the right operand	a < b, returns 1
>=	check left operand is greater than or equal to the right operand	a >= b, returns 0
<=	Check if the operand on the left is smaller than or equal to the right operand	a <= b, returns 1

3. Logical Operators in C

C language supports the following 3 logical operators.

rator	Description	Example (a and b, where a = 1 and b = 0)
&&	Logical AND	a && b, returns 0
||	Logical OR	a || b, returns 1
!	Logical NOT	!a, returns 0

These operators are used to perform logical operations and are used with conditional statements like C if-else statements.

1.With the **AND** operator, only **if both operands are true**, the **result is true**

2.With the **OR** operator, if a **single operand is true**, then the **result will be true**.

3.The **NOT** operator **changes true to false**, and **false to true**.

4. Bitwise Operators in C

- Bitwise operators perform manipulations of data at the bit level.
- These operators also perform the **shifting of bits from right to left**.
- Bitwise operators are not applied to float or double, long double, void
- There are **6 bitwise operators** in C programming.

The following table contains the bitwise operators.

Operator	Description	Example
&	Bitwise AND	a&b
|	Bitwise OR	a|b
^	Bitwise Exclusive OR (XOR)	a^b
~	One's complement (NOT)	~a
>>	Shift right	a>>2
<<	Shift left	a<<2

5. Assignment Operators in C

•The assignment operators are used to assign value to a variable.

•For example, if we want to assign a value **10** to a variable x then we can do this by using the assignment operator like: x = 10; Here, = (equal to) operator is used to assign the value.

•In the C language, the = (equal to) operator is **used for assignment** however it has several other variants such as +=, -= to combine two operations in a single statement. You can see all the assignment operators in the table given below.

ator	Description	Example (a and b are two variables, with where a=10 and b=5)
=	assigns values from right side operand to left side operand	a=b, a gets value 5
+=	adds right operand to the left operand and assign the result to left operand	a+=b, is same as a=a+b, value of a becomes 15
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b, is same as a=a-b, value of a becomes 5
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b, is same as a=a*b, value of a becomes 50
/=	divides left operand with the right operand and assign the result to left operand	a/=b, is same as a=a/b, value of a becomes 2
%=	calculate modulus using two operands and assign the result to left operand	a%=b, is same as a=a%b, value of a becomes 0

6. Conditional Operator (?:)

The ternary operator, also known as the conditional operator in the C language can be used for statements of the form if-then-else.

The **basic syntax** for using ternary operator is:

```
EXP1 ? EXP2 : EXP3;
```

Here is how it works:

•The question mark ? in the syntax represents the if part.
•The first expression (expression 1) returns either **true** or **false**, based on which it is decided whether (expression 2) will be executed or (expression 3)
•If (expression 1) returns **true** then the (expression 2) is executed.
•If (expression 1) returns **false** then the expression on the right side of : i.e (expression 3) is executed.

For exp:-

```
result = (a==b)?(a+b):(a-b);
```

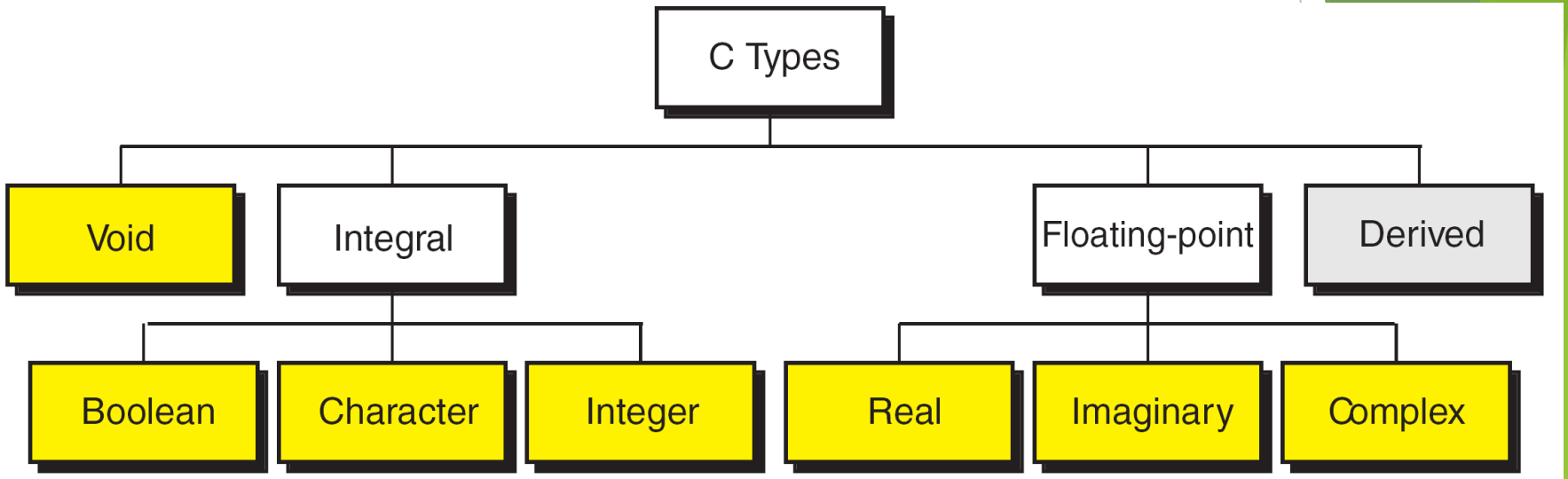
Special Operator

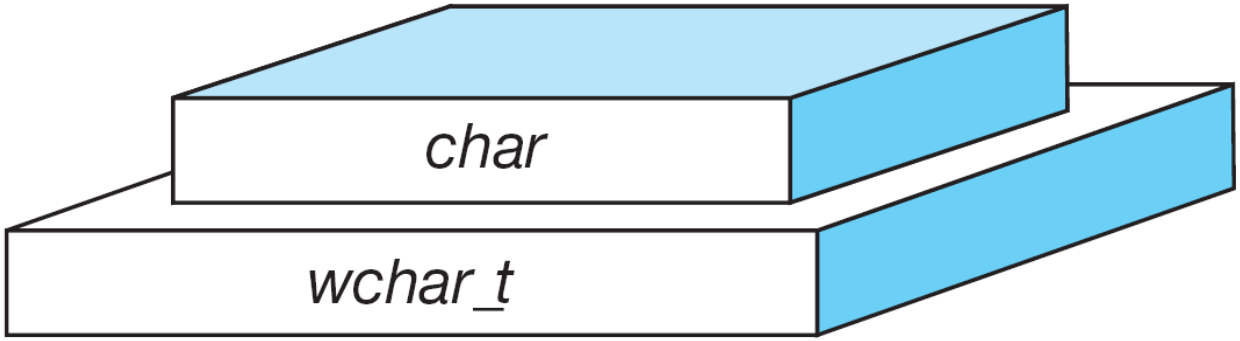
C language uses some other operators such as:

1.sizeof operator
2.& operator
3.* operator
4.The . (dot) and -> (arrow) operators
5.[] operator, etc.

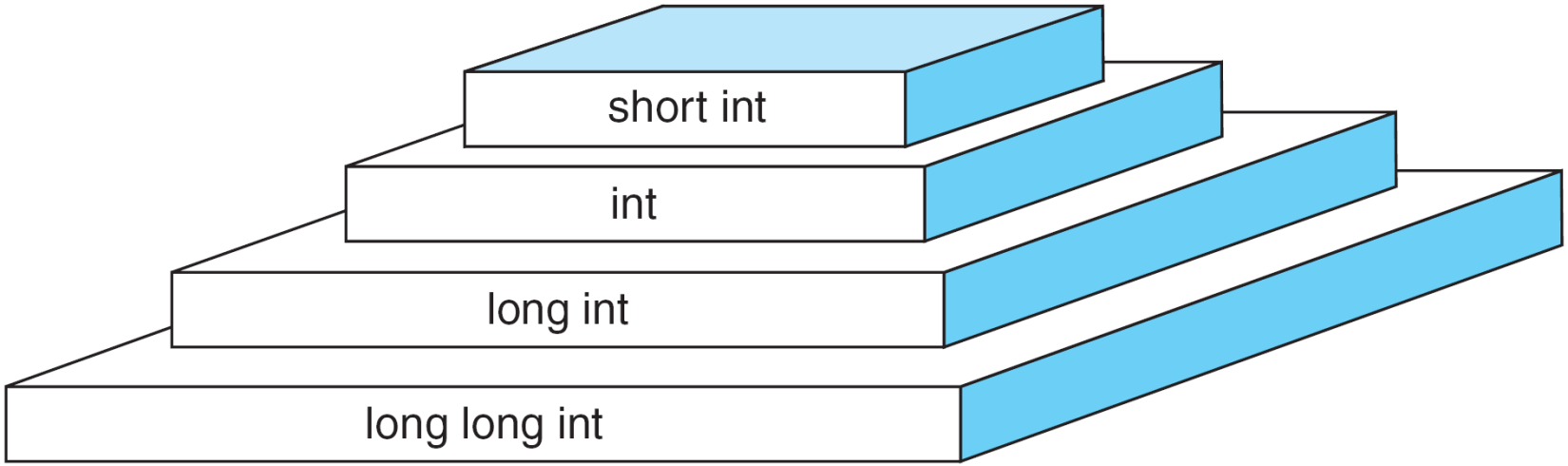
Operator	Description	Example
sizeof	returns the **size(length in bytes) of the entity**, for eg. a variable or an array, etc.	sizeof(x) will return the size of the variable x
&	returns the memory address of the variable	&x will return the address of the variable x
*	represents a pointer to an object. The * operator returns the value stored at a memory address.	m = &x (memory address of the variable x) *m will return the value stored at the memory address m
. (dot) operator	used to access individual elements of a C structure or C union.	If emp is a structure with an element int age in it, then emp.age will return the value of age.
-> (arrow) operator	used to access structure or union elements using a pointer to the structure or union.	If p is a pointer to the emp structure, then we can access age element using p->age
[] operator	used to access array elements using indexing	if arr is an array, then we can access its values using arr[index], where index represents the array index starting from zero

Data Types


C Types
Void
Integral
Floating-point
Derived
Boolean
Character
Integer
Real
Imaginary
Complex



Character Types



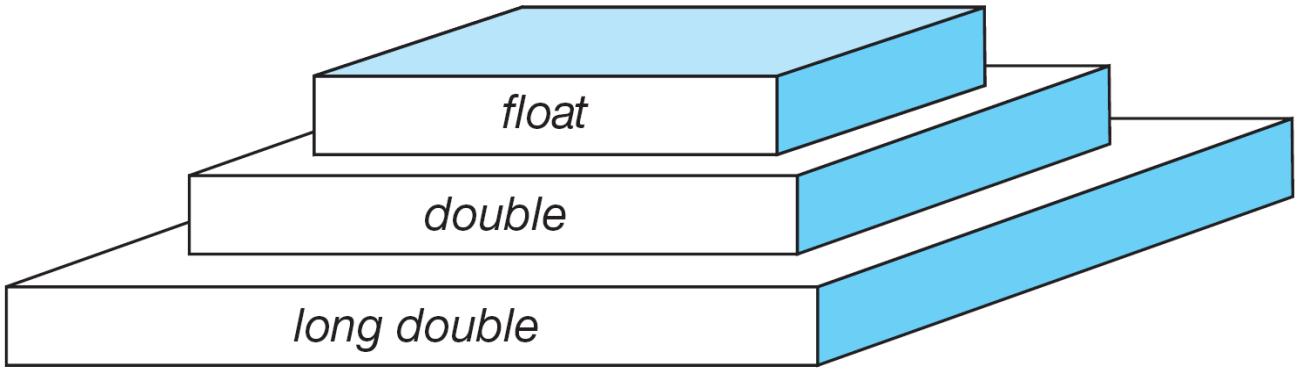
Integer Types

Note

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

Typical Integer Sizes and Values for Signed Integers



Floating-point Types

Note

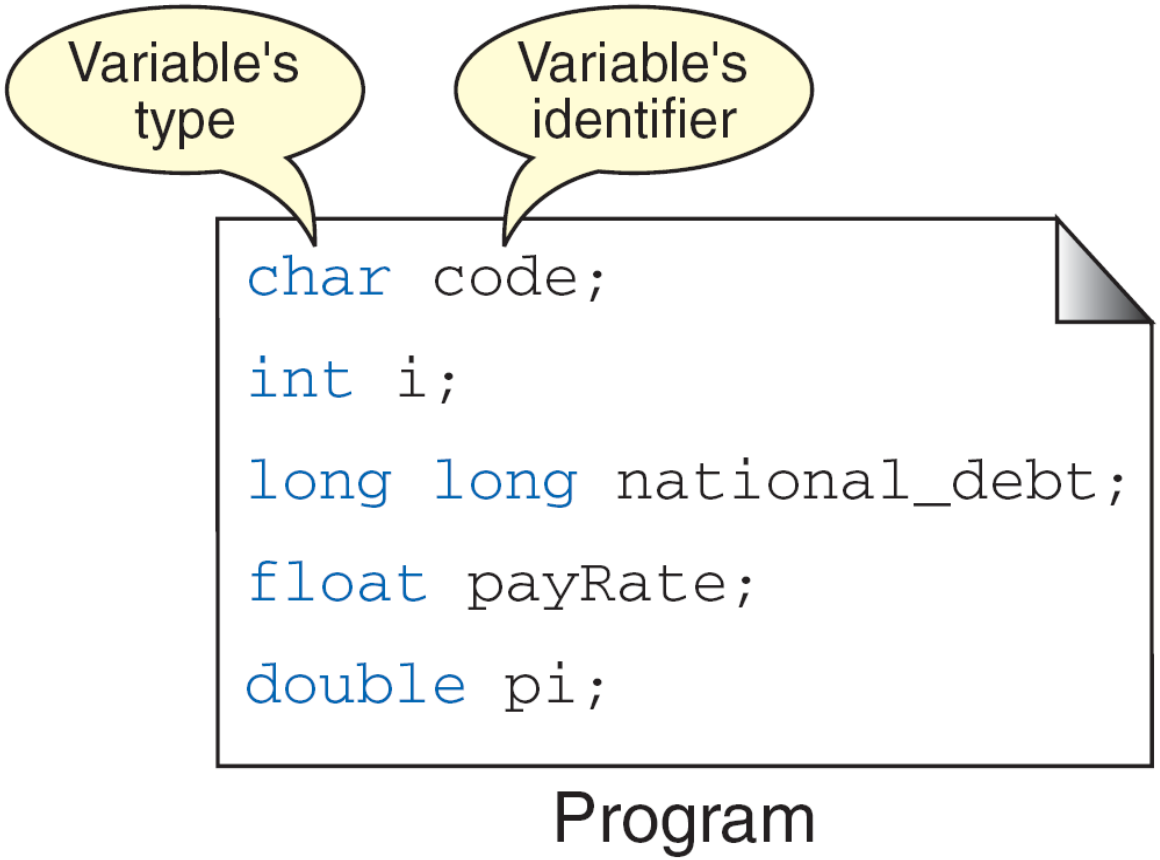
sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Category	Type	C Implementation
Void	Void	*void*
Integral	Boolean	*bool*
	Character	*char, wchar_t*
	Integer	*short int, int, long int, long long int*
Floating-Point	Real	*float, double, long double*
	Imaginary	*float imaginary, double imaginary, long double imaginary*
	Complex	*float complex, double complex, long double complex*

Type Summary

Variables

Variables are named memory locations that have a type, such as integer or character, which is inherited from their type. The type determines the values that a variable may contain and the operations that may be used with its values.


Variable's type
Variable's identifier
char code;
int i;
long long national_debt;
float payRate;
double pi;
Program

Variables

```
bool   fact;
short  maxItems;                 // Word separator: Capital
long   long national_debt;       // Word separator: underscore
float  payRate;                  // Word separator: Capital
double tax;
float  complex voltage;
char   code, kind;               // Poor style—see text
int    a, b;                     // Poor style—see text
```

Examples of Variable Declarations and Definitions

Constants

Constants are data values that cannot be changed during the execution of a program. Like variables, constants have a type. In this section, we discuss Boolean, character, integer, real, complex, and string constants.

ASCII Character	Symbolic Name
null character	`'\0'`
alert (bell)	`'\a'`
backspace	`'\b'`
horizontal tab	`'\t'`
newline	`'\n'`
vertical tab	`'\v'`
form feed	`'\f'`
carriage return	`'\r'`
single quote	`'\''`
double quote	`'\"'`
backslash	`'\\'`

Symbolic Names for Control Characters