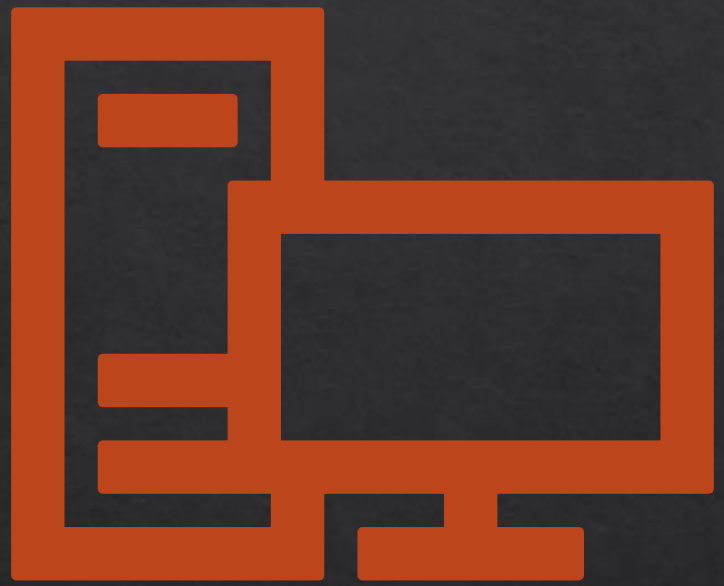
4. Programming in C

Prepared By: Ajay Singh



4.1 Review of C programming concept

- Variables: A variable is a named memory location that stores a value. In C, variables must be declared before they are used, and they must be of a specific data type (e.g. int, char, float, etc.).
- Data types: C provides a range of data types for representing different types of values, such as integers, characters, and floating-point numbers.
- Input and Output: C provides functions for reading and writing data
- Operators: C provides a range of operators that can be used to perform operations on values, such as arithmetic, logical, and comparison operations.
- Control structures: C provides a range of control structures that can be used to control the flow of a program, such as if-else statements, for loops, and while loops.

Prepared By: Ajay Singh

Introduction of C

- *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.*

Features of C

- Simple and Efficient
- Fast
- Portability
- Extensibility
- Function-Rich Libraries
- Dynamic Memory Management
- Modularity With Structured Language
- Mid-Level Programming Language
- Pointers
- Recursion

Prepared By: Ajay Singh

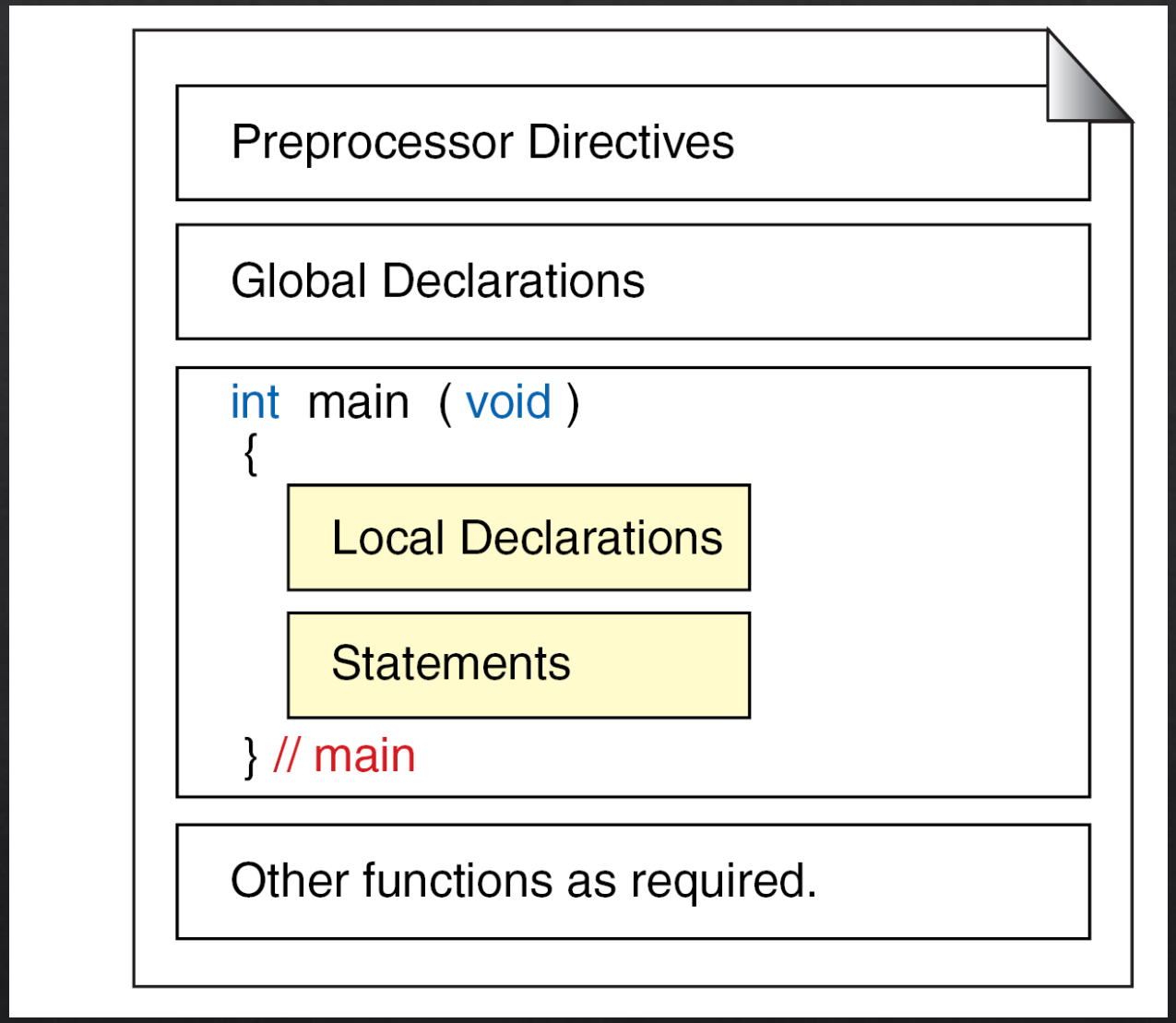
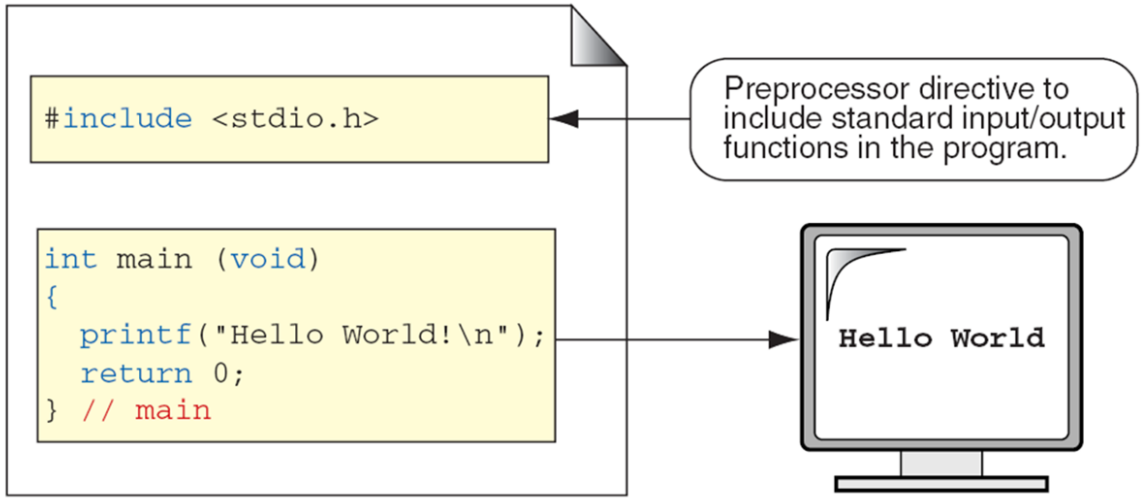


FIGURE Structure of a C Program

C Preprocessor and Header Files

- **A header file is a file containing C declarations and macro definitions to be shared between several source files. You request the use of a header file in your program by including it, with the C preprocessing directive '#include'.**



First C Program

- *It's time to write your first C program.*

Prepared By: Ajay Singh

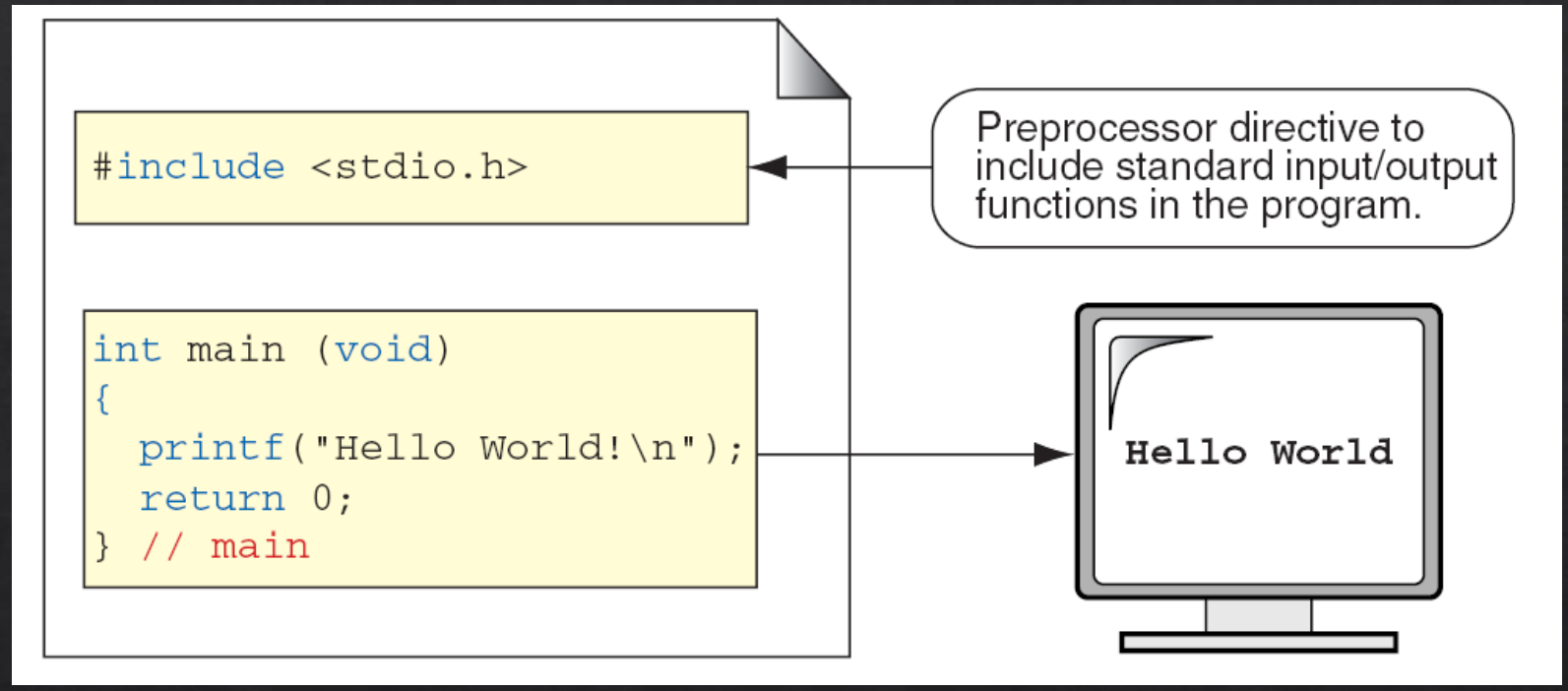


FIGURE The Greeting Program

PROGRAM The Greeting 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
```

```
/* This is a block comment that
   covers two lines.                    */

/*
** It is a very common style to put the opening token
** on a line by itself, followed by the documentation
** and then the closing token on a separate line. Some
** programmers also like to put asterisks at the beginning
** of each line to clearly mark the comment.
*/
```

FIGURE Examples of Block Comments

```
// This is a whole line comment

a = 5;              // This is a partial line comment
```

FIGURE Examples of Line Comments

Character set

Type of Character	Description	Characters
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special Characters	–	` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ‘ “ , . ?
White Spaces	–	Blank Spaces, Carriage Return, Tab, New Line

Identifiers

One feature present in all computer languages is the identifier. Identifiers allow us to name data and other objects in the program. Each identified object in the computer is stored at a unique address.

Prepared By: Ajay Singh

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.

Table Rules for Identifiers

Note

An identifier must start with a letter or underscore:
it may not have a space or a hyphen.

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

Table Examples of Valid and Invalid Names

Data Types

A type defines a set of values and a set of operations that can be applied on those values.

Topics discussed in this section:

Void Type
Integral Type
Floating-Point Types

Prepared By: Ajay Singh

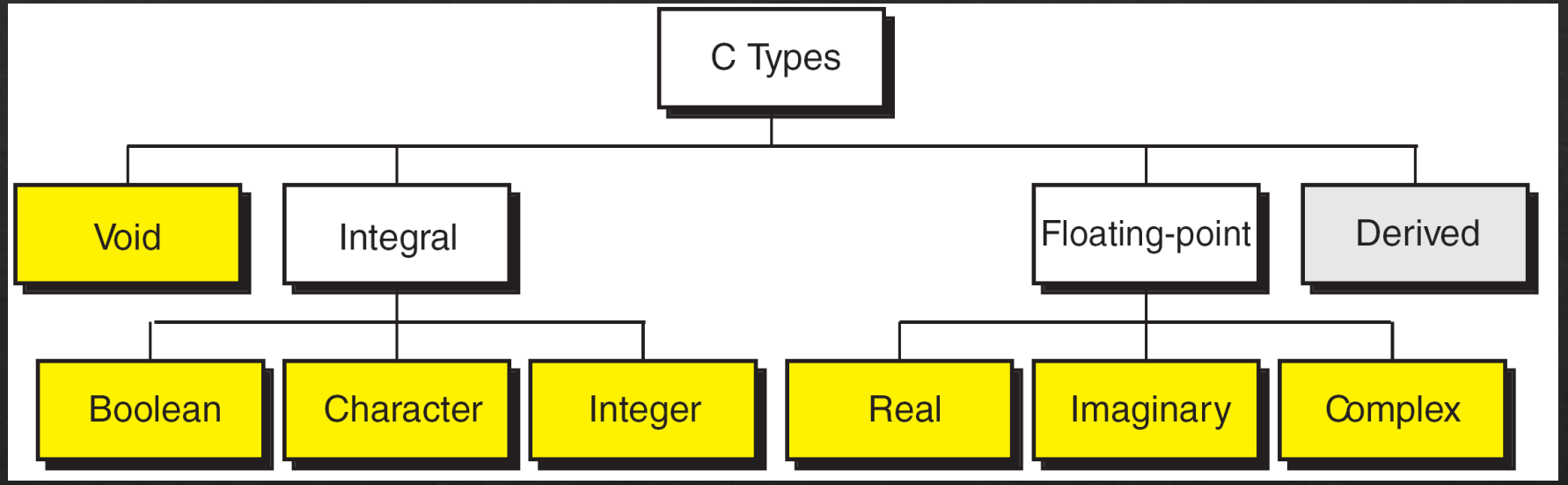


FIGURE Data Types

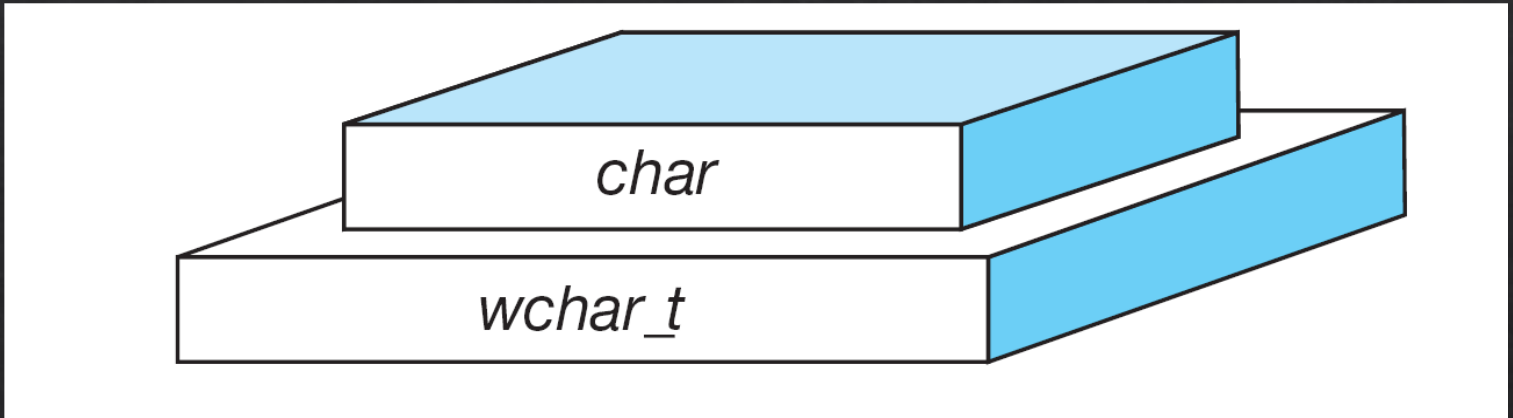


FIGURE Character Types

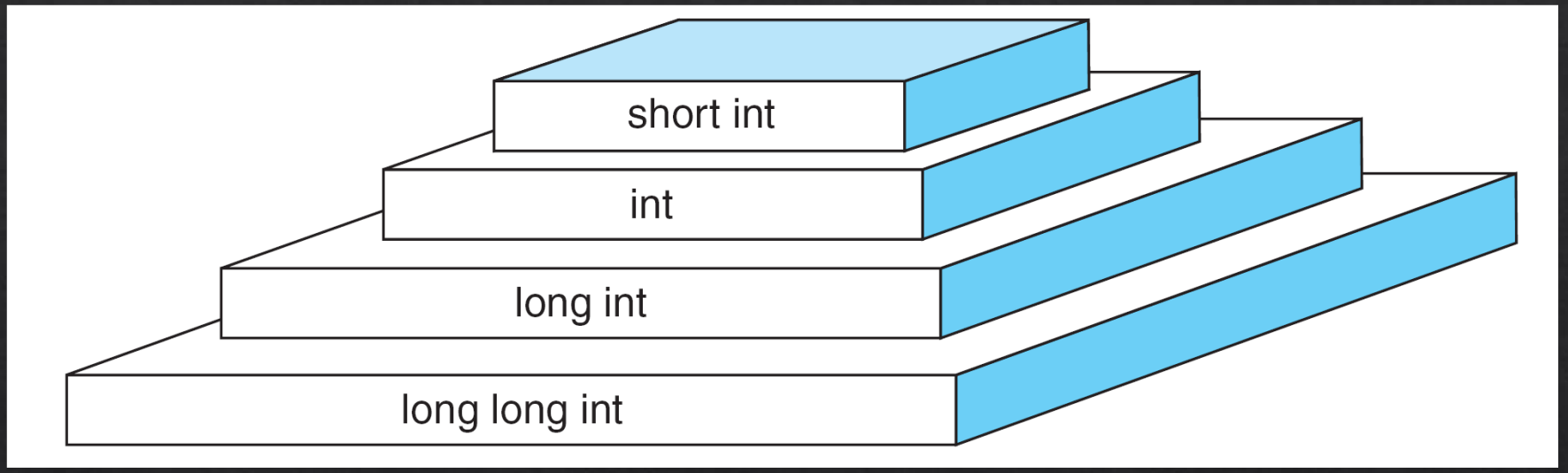


FIGURE Integer Types

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

Table **Typical Integer Sizes and Values for Signed Integers**

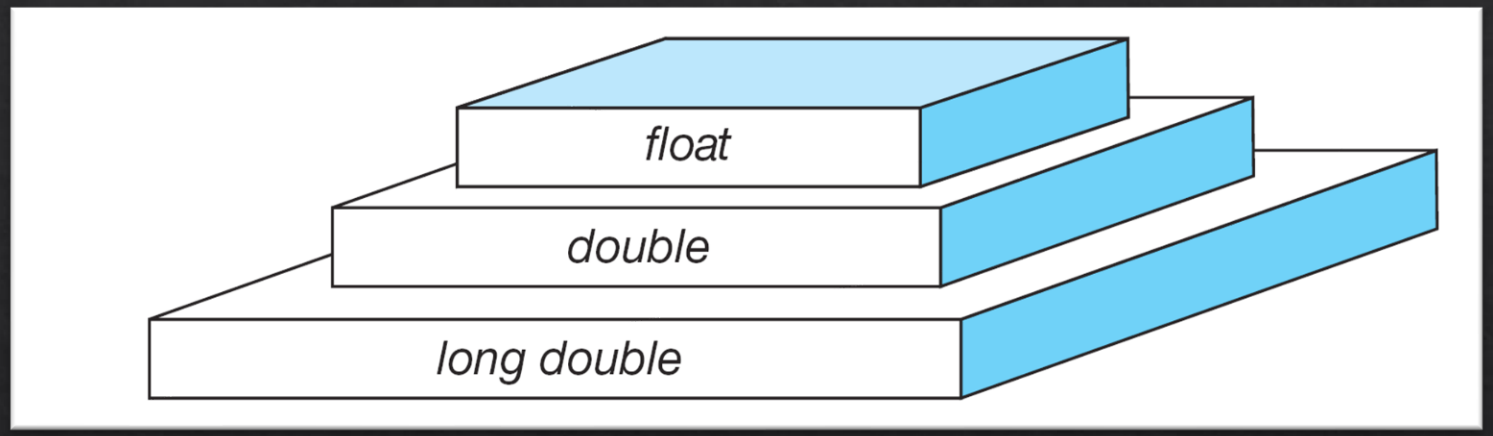


FIGURE Floating-point Types

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*

Table Type Summary

Prepared By: Ajay Singh

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.

Topics discussed in this section:

Variable Declaration

Variable Initialization

Prepared By: Ajay Singh

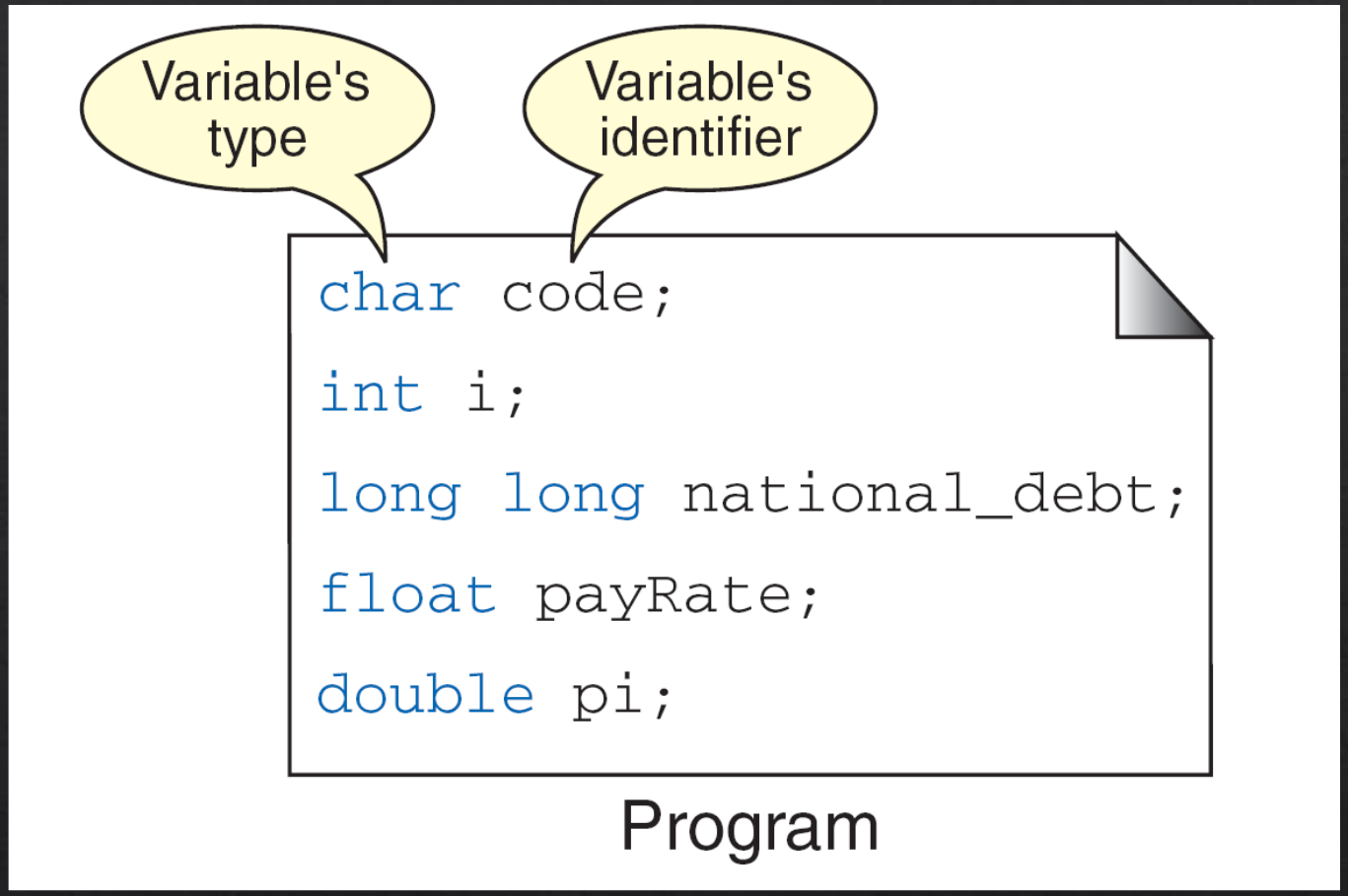


FIGURE Variables

```
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
```

Table **Examples of Variable Declarations and Definitions**

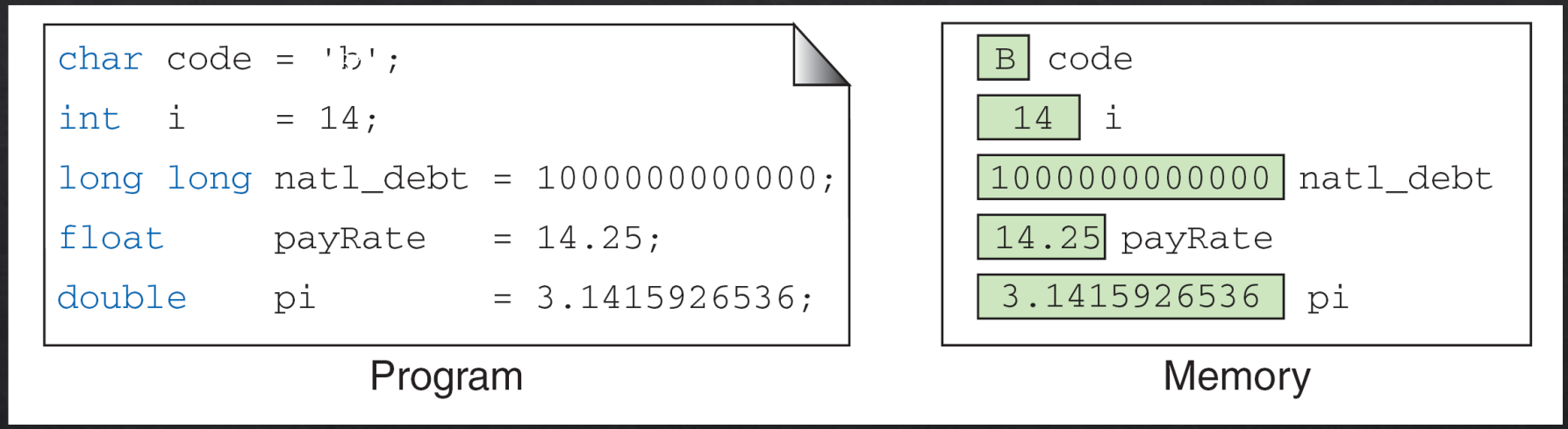


FIGURE Variable Initialization

Note

When a variable is defined, it is not initialized. We must initialize any variable requiring prescribed data when the function starts.

PROGRAM Print Sum of Three Numbers

```
1  /* This program calculates and prints the sum of
2     three numbers input by the user at the keyboard.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    int a;
12    int b;
13    int c;
14    int sum;
15
```

PROGRAM Print Sum of Three Numbers (continued)

```
16  // Statements
17     printf("\nWelcome. This program adds\n");
18     printf("three numbers. Enter three numbers\n");
19     printf("in the form: nnn nnn nnn <return>\n");
20     scanf("%d %d %d", &a, &b, &c);
21
22     // Numbers are now in a, b, and c. Add them.
23     sum = a + b + c;
24
25     printf("The total is: %d\n\n", sum);
26
27     printf("Thank you. Have a good day.\n");
28     return 0;
29  }  //  main
```

PROGRAM Print Sum of Three Numbers (continued)

```
Results:
Welcome. This program adds
three numbers. Enter three numbers
in the form: nnn nnn nnn <return>
11 22 33

The total is: 66

Thank you. Have a good day.
```

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.

Topics discussed in this section:

Constant Representation

Coding Constants

Prepared By: Ajay Singh

Note

A character constant is enclosed in single quotes.

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	'\\'

Table Symbolic Names for Control Characters

Representation	Value	Type
+123	123	int
–378	–378	int
–32271L	–32,271	long int
76542LU	76,542	unsigned long int
12789845LL	12,789,845	long long int

Table **Examples of Integer Constants**

Representation	Value	Type
0.	0.0	double
.0	0.0	double
2.0	2.0	double
3.1416	3.1416	double
–2.0f	–2.0	float
3.1415926536L	3.1415926536	long double

Table Examples of Real Constants

```
""                                                    // A null string
"h"
"Hello World\n"
"HOW ARE YOU"
"Good Morning!"
L"This string contains wide characters."
```

FIGURE Some Strings

FIGURE Null Characters and Null Strings

Note

Use single quotes for character constants.
Use double quotes for string constants.

PROGRAM Memory Constants

```
1  /* This program demonstrates three ways to use con-
   stants.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #define PI 3.1415926536
7
8  int main (void)
9  {
10 // Local Declarations
11    const double cPi = PI;
12
13 // Statements
14    printf("Defined constant PI: %f\n", PI);
15    printf("Memory constant cPi: %f\n", PI);
```

PROGRAM Memory Constants (continued)

```
16      printf("Literal constant:     %f\n", 3.1415926536);
17      return 0;
18  }   // main
```

```
Results:
Defined constant PI:   3.141593
Memory constant cPi:   3.141593
Literal constant:      3.141593
```

Operators in C

Prepared By: Ajay Singh

An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Table C Arithmetic Operators

PROGRAM 2-3 Operators in C

```c
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Prepared By: Ajay Singh

Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Prepared By: Ajay Singh

PROGRAM 2-3 Operators in C

```c
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
--d = 99.500000
```

Prepared By: Ajay Singh

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b

Prepared By: Ajay Singh

PROGRAM Operators in C

```c
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;      // c is 5
    printf("c = %d\n", c);
    c += a;     // c is 10
    printf("c = %d\n", c);
    c -= a;     // c is 5
    printf("c = %d\n", c);
    c *= a;     // c is 25
    printf("c = %d\n", c);
    c /= a;     // c is 5
    printf("c = %d\n", c);
    c %= a;     // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Prepared By: Ajay Singh

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

Prepared By: Ajay Singh

PROGRAM Operators in C

```c
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

Prepared By: Ajay Singh

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

Prepared By: Ajay Singh

PROGRAM Operators in C

```c
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

Output

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Prepared By: Ajay Singh