Unix Systems Programming (CSE 3041)

Dr. Trilok Nath Pandey

S 'O' A Deemed to be University
Dept. of C.S.E. ITER, Bhubaneswar

C Language Elements

Text Books

- **Jeri R. Hanly, & Elliot B. Koffman**
  Problem Solving and Program Design in C, 7th Edition
  Pearson Education
- **Kay A Robbins, & Steven Robbins**
  The Unix System Programming
  Communication,Concurrency, & Threads
  Pearson Education
- **Brain W. Kernighan, & Rob Pike**
  The Unix Programming Environment
  PHI

C Language Elements

- The C program has two parts: preprocessor directives and the main function.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.
- The two most common directives are: #include and #define

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.
- The two most common directives are: #include and #define

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.
- The two most common directives are: #include and #define
- Every C implementation contains collections of useful functions and symbols called libraries.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.
- The two most common directives are: #include and #define
- Every C implementation contains collections of useful functions and symbols called libraries.

C Language Elements

- The C program has two parts: preprocessor directives and the main function.
- The preprocessor directives are commands that give instructions to the C preprocessor, whose job it is to modify the text of a C program before it is compiled.
- A preprocessor directive begins with a number symbol # as its first non blank character.
- The two most common directives are: #include and #define
- Every C implementation contains collections of useful functions and symbols called libraries.
- Each library has a standard header file whose name ends with the symbols .h.

C Language Elements

C Language Elements

- C Language Elements in Miles-to-Kilometers Conversion Program

C Language Elements

- C Language Elements in Miles-to-Kilometers Conversion Program

C Language Elements

- C Language Elements in Miles-to-Kilometers Conversion Program

C Language Elements in Miles-to-Kilometers Conversion Program

```
/*
 * Converts distances from miles to kilometers.
 */
```

standard header file

comment

preprocessor directive

```
#include <stdio.h>          /* printf, scanf definitions */
#define KMS_PER_MILE 1.609  /* conversion constant       */
```

constant

reserved word

```
int
main(void)
{
      double miles, /* distance in miles
```

variable

```
             kms;   /* equivalent distance in kilometers */
```

comment

standard identifier

```
      /* Get the distance in miles. */
      printf("Enter the distance in miles> ");
      scanf("%lf", &miles);

      /* Convert the distance to kilometers. */
      kms = KMS_PER_MILE * miles;
```

special symbol

```
      /* Display the distance in kilometers. */
      printf("That equals %f kilometers.\n", kms);
```

reserved word

punctuation

```
      return (0);
}
```

special symbol

Syntax Displays for Preprocessor Directives

Syntax Displays for Preprocessor Directives

- Explains the construct's syntax and shows examples of its use.

Syntax Displays for Preprocessor Directives

- Explains the construct's syntax and shows examples of its use.

Syntax Displays for Preprocessor Directives

- Explains the construct's syntax and shows examples of its use.

#include Directive for Defining Identifiers from Standard Libraries

SYNTAX: `#include <standard header file>`

EXAMPLES:

```
#include <stdio.h>
#include <math.h>
```

INTERPRETATION: `#include` directives tell the preprocessor where to find the meanings of standard identifiers used in the program. These meanings are collected in files called *standard header files*. The header file `stdio.h` contains information about standard input and output functions such as `scanf` and `printf`. Descriptions of common mathematical functions are found in the header file `math.h`. We will investigate header files associated with other standard libraries in later chapters.

Function main

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.
- The declarations tell the compiler what memory cells are needed in the function.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.
- The declarations tell the compiler what memory cells are needed in the function.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.
- The declarations tell the compiler what memory cells are needed in the function.
- The executable statements (derived from the algorithm) are translated into machine language and later executed.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.
- The declarations tell the compiler what memory cells are needed in the function.
- The executable statements (derived from the algorithm) are translated into machine language and later executed.

Function main

- The two-line heading marks the beginning of the main function where program execution begins. Every C program has a main function.
- A function body has two parts: declarations and executable statements.
- The declarations tell the compiler what memory cells are needed in the function.
- The executable statements (derived from the algorithm) are translated into machine language and later executed.

main Function Definition

SYNTAX:
```
int
main(void)
{
        function body
}
```

(continued)

Reserved Words

Reserved Words

- reserved word a word that has special meaning in C

Reserved Words

- reserved word a word that has special meaning in C

Reserved Words

- reserved word a word that has special meaning in C
- All the reserved words appear in lowercase; they have special meaning in C and cannot be used for other purposes.

Reserved Words

- reserved word a word that has special meaning in C
- All the reserved words appear in lowercase; they have special meaning in C and cannot be used for other purposes.

Reserved Words

- reserved word a word that has special meaning in C
- All the reserved words appear in lowercase; they have special meaning in C and cannot be used for other purposes.

ANSI C RESERVED WORDS

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Standard Identifiers

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.
- Unlike reserved words, standard identifiers can be redefined and used by the programmer for other purposes however,we don't recommend this practice.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.
- Unlike reserved words, standard identifiers can be redefined and used by the programmer for other purposes however,we don't recommend this practice.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.
- Unlike reserved words, standard identifiers can be redefined and used by the programmer for other purposes however,we don't recommend this practice.
- If you redefine a standard identifier, C will no longer be able to use it for its original purpose.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.
- Unlike reserved words, standard identifiers can be redefined and used by the programmer for other purposes however,we don't recommend this practice.
- If you redefine a standard identifier, C will no longer be able to use it for its original purpose.

Standard Identifiers

- The other words in the example are identifiers that come in two varieties: standard and user-defined.
- standard identifier a word having special meaning but one that a programmer may redefine (but redefinition is not recommended!)
- the standard identifiers printf and scanf are names of operations defined in the standard input/output library.
- Unlike reserved words, standard identifiers can be redefined and used by the programmer for other purposes however,we don't recommend this practice.
- If you redefine a standard identifier, C will no longer be able to use it for its original purpose.

User-Defined Identifiers

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.
- An identifier defined in a C standard library should not be redefined.(advice from the authors rather than ANSI C syntax.)

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.
- An identifier defined in a C standard library should not be redefined.(advice from the authors rather than ANSI C syntax.)

User-Defined Identifiers

- We choose our own identifiers (called user-defined identifiers) to name memory cells that will hold data and program results and to name operations that we define.
- The syntax rules and some valid identifiers follow.
- An identifier must consist only of letters, digits, and underscores.
- An identifier cannot begin with a digit.
- A C reserved word cannot be used as an identifier.
- An identifier defined in a C standard library should not be redefined.(advice from the authors rather than ANSI C syntax.)
- Valid Identifiers : letter_1, letter_2, inches, cent, CENT_PER, Hello, variable

User-Defined Identifiers

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.
- lets consider the two identifiers
  per_capita_meat_consumption_in_1980
  per_capita_meat_consumption_in_1995

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.
- lets consider the two identifiers
  per_capita_meat_consumption_in_1980
  per_capita_meat_consumption_in_1995

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.
- lets consider the two identifiers
  per_capita_meat_consumption_in_1980
  per_capita_meat_consumption_in_1995
- would be viewed as identical by a C compiler that considered only the first 31 characters to be significant

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.
- lets consider the two identifiers
  per_capita_meat_consumption_in_1980
  per_capita_meat_consumption_in_1995
- would be viewed as identical by a C compiler that considered only the first 31 characters to be significant

User-Defined Identifiers

- Although the syntax rules for identifiers do not place a limit on length, some ANSI C compilers do not consider two names to be different unless there is a variation within the first 31 characters.
- lets consider the two identifiers
  per_capita_meat_consumption_in_1980
  per_capita_meat_consumption_in_1995
- would be viewed as identical by a C compiler that considered only the first 31 characters to be significant

Reserved Words	Standard Identifiers	User-Defined Identifiers
`int, void, double, return`	`printf, scanf`	`KMS_PER_MILE, main, miles, kms`

Variable Declarations and Data Types

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.
- They also tell the compiler what kind of information will be stored in each variable and how that information will be represented in memory.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.
- They also tell the compiler what kind of information will be stored in each variable and how that information will be represented in memory.

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.
- They also tell the compiler what kind of information will be stored in each variable and how that information will be represented in memory.
- double miles; /* input - distance in miles. */

Variable Declarations and Data Types

- variable a name associated with a memory cell whose value can change.
- The memory cells used for storing a program's input data and its computational results are called variables because the values stored in variables can change (and usually do) as the program executes.
- They also tell the compiler what kind of information will be stored in each variable and how that information will be represented in memory.
- double miles; /* input - distance in miles. */
- double kms; /* output - distance in kilometers */

Variable Declarations and Data Types

Variable Declarations and Data Types

Syntax Display for Declarations

SYNTAX:

```
int variable_list;
double variable_list;
char variable_list;
```

EXAMPLES:

```
int count,
    large;
double x, y, z;
char first_initial;
char ans;
```

INTERPRETATION: A memory cell is allocated for each name in the *variable_list*. The type of data (`double`, `int`, `char`) to be stored in each variable is specified at the beginning of the statement. One statement may extend over multiple lines. A single data type can appear in more than one variable declaration, so the following two declaration sections are equally acceptable ways of declaring the variables `rate`, `time`, and `age`.

```
double rate, time;
int age;
```

```
double   rate;
int      age;
double   time;
```

Data Types

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.
- **Data Type double :** In C, the data type double is used to represent real numbers (for example, 3.14159, 0.0005, 150.0)

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.
- **Data Type double :** In C, the data type double is used to represent real numbers (for example, 3.14159, 0.0005, 150.0)

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.
- **Data Type double :** In C, the data type double is used to represent real numbers (for example, 3.14159, 0.0005, 150.0)
- **Data Type char :** Data type char represents an individual character value—a letter, a digit, or a special symbol. Each type char value is enclosed in apostrophes (single quotes) as shown here.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.
- **Data Type double :** In C, the data type double is used to represent real numbers (for example, 3.14159, 0.0005, 150.0)
- **Data Type char :** Data type char represents an individual character value—a letter, a digit, or a special symbol. Each type char value is enclosed in apostrophes (single quotes) as shown here.

Data Types

- **Data Type** It is a set of values and operations that can be performed on those values.
- **Data Type int :** The int data type is used to represent integers in C. Because of the finite size of a memory cell, not all integers can be represented by type int.
- ANSI C specifies that the range of data type int must include at least the values - 32767 through + 32767.
- **Data Type double :** In C, the data type double is used to represent real numbers (for example, 3.14159, 0.0005, 150.0)
- **Data Type char :** Data type char represents an individual character value—a letter, a digit, or a special symbol. Each type char value is enclosed in apostrophes (single quotes) as shown here.
- ’A’ ’z’ ’2’ ’9’ ’*’ ’:’ ’”’ ’ ’

The ASCII Code

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.
- The ASCII code(American Standard Code for Information Interchange) is the most common.

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.
- The ASCII code(American Standard Code for Information Interchange) is the most common.

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.
- The ASCII code(American Standard Code for Information Interchange) is the most common.
- In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for the symbol $\sim$).

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.
- The ASCII code(American Standard Code for Information Interchange) is the most common.
- In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for the symbol $\sim$).

The ASCII Code

- You should know that a character is represented in memory as an integer. The value stored is determined by the code used by your C compiler.
- The ASCII code(American Standard Code for Information Interchange) is the most common.
- In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for the symbol $\sim$).
- The other codes represent nonprintable control characters. Sending a control character to an output device causes the device to perform a special operation such as returning the cursor to column one, advancing the cursor to the next line, or ringing a bell.

The ASCII Code

The ASCII Code

The charts in this appendix show the following character sets: ASCII (American Standard Code for Information Interchange), EBCDIC (Extended Binary Coded Decimal Interchange Code), and CDC[†] Scientific. Only printable characters are shown. The integer code for each character is shown in decimal. For example, in ASCII, the code for 'A' is 65, and the code for 'z' is 122. The blank character is denoted by □.

Right Digit / Left Digit(s)	ASCII									
	0	1	2	3	4	5	6	7	8	9
3			□	!	"	#	$	%	&	'
4	(	)	*	+	,	–	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	/	]	^	–	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}				

Executable Statements

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.
- The C compiler translates the executable statements into machine language; the computer executes the machine language version of these statements when we run the program.

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.
- The C compiler translates the executable statements into machine language; the computer executes the machine language version of these statements when we run the program.

Executable Statements

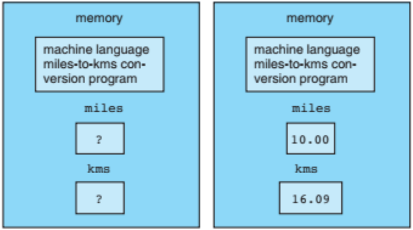
- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.
- The C compiler translates the executable statements into machine language; the computer executes the machine language version of these statements when we run the program.
- Memory (a) Before and (b) After Execution of a Program

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.
- The C compiler translates the executable statements into machine language; the computer executes the machine language version of these statements when we run the program.
- Memory (a) Before and (b) After Execution of a Program

Executable Statements

- The executable statements follow the declarations in a function. They are the C statements used to write or code the algorithm and its refinements.
- The C compiler translates the executable statements into machine language; the computer executes the machine language version of these statements when we run the program.
- Memory (a) Before and (b) After Execution of a Program



(a) (b)

Assignment Statements

Assignment Statements

- An assignment statement stores a value or a computational result in a variable, and is used to perform most arithmetic operations in a program.

Assignment Statements

- An assignment statement stores a value or a computational result in a variable, and is used to perform most arithmetic operations in a program.

Assignment Statements

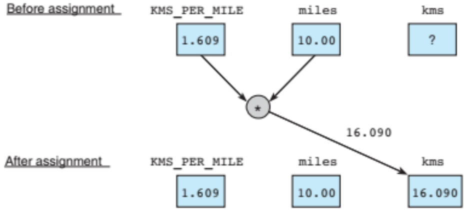
- An assignment statement stores a value or a computational result in a variable, and is used to perform most arithmetic operations in a program.
- kms = KMS_PER_MILE * miles;

Assignment Statements

- An assignment statement stores a value or a computational result in a variable, and is used to perform most arithmetic operations in a program.
- kms = KMS_PER_MILE * miles;

Assignment Statements

- An assignment statement stores a value or a computational result in a variable, and is used to perform most arithmetic operations in a program.
- kms = KMS_PER_MILE * miles;



Assignment Statement

FORM: *variable* = *expression*;

EXAMPLE: `x = y + z + 2.0;`

INTERPRETATION: The *variable* before the assignment operator is assigned the value of the *expression* after it. The previous value of *variable* is destroyed. The *expression* can be a variable, a constant, or a combination of these connected by appropriate operators (for example, +, −, /, and *).

Input/Output Operations and Functions

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation
- In C a function call is used to call or activate a function.

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation
- In C a function call is used to call or activate a function.

Input/Output Operations and Functions

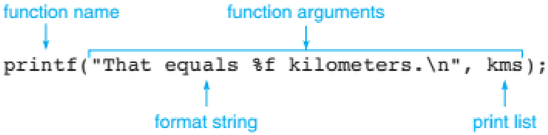
- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation
- In C a function call is used to call or activate a function.
- The printf Function

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation
- In C a function call is used to call or activate a function.
- The printf Function

Input/Output Operations and Functions

- **input operation :** an instruction that copies data from an input device into memory
- **output operation :** an instruction that displays information stored in memory
- **input/output function :** a C function that performs an input or output operation
- In C a function call is used to call or activate a function.
- The printf Function



Input/Output Operations and Functions

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line
- **print list:** in a call to printf, the variables or expressions whose values are displayed placeholder a symbol beginning with % in a format string that indicates where to display the output value

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line
- **print list:** in a call to printf, the variables or expressions whose values are displayed placeholder a symbol beginning with % in a format string that indicates where to display the output value

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line
- **print list:** in a call to printf, the variables or expressions whose values are displayed placeholder a symbol beginning with % in a format string that indicates where to display the output value
- **placeholder:** a symbol beginning with % in a format string that indicates where to display the output value

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line
- **print list:** in a call to printf, the variables or expressions whose values are displayed placeholder a symbol beginning with % in a format string that indicates where to display the output value
- **placeholder:** a symbol beginning with % in a format string that indicates where to display the output value

Input/Output Operations and Functions

- **function argument :** function argument enclosed in parentheses following the function name; provides information needed by the function
- **format string :** format string in a call to printf, a string of characters enclosed in quotes ("), which specifies the form of the output line
- **print list:** in a call to printf, the variables or expressions whose values are displayed placeholder a symbol beginning with % in a format string that indicates where to display the output value
- **placeholder:** a symbol beginning with % in a format string that indicates where to display the output value
- **newline escape sequence :** the character sequence \n, which is used in a format string to terminate an output line

Input/Output Operations and Functions

Input/Output Operations and Functions

- **Multiple placeholder:** Format strings can have multiple placeholders. If the print list of a printf call has several variables, the format string should contain the same number of placeholders. C matches variables with placeholders in left-to-right order.

Input/Output Operations and Functions

- **Multiple placeholder:** Format strings can have multiple placeholders. If the print list of a printf call has several variables, the format string should contain the same number of placeholders. C matches variables with placeholders in left-to-right order.

Input/Output Operations and Functions

- **Multiple placeholder:** Format strings can have multiple placeholders. If the print list of a printf call has several variables, the format string should contain the same number of placeholders. C matches variables with placeholders in left-to-right order.

Placeholders in Format Strings

Placeholder	Variable Type	Function Use
%c	char	printf/scanf
%d	int	printf/scanf
%f	double	printf
%lf	double	scanf

Input/Output Operations and Functions

- **Multiple placeholder:** Format strings can have multiple placeholders. If the print list of a printf call has several variables, the format string should contain the same number of placeholders. C matches variables with placeholders in left-to-right order.

Placeholders in Format Strings

Placeholder	Variable Type	Function Use
`%c`	`char`	`printf/scanf`
`%d`	`int`	`printf/scanf`
`%f`	`double`	`printf`
`%lf`	`double`	`scanf`

Syntax Display for printf Function Call

SYNTAX:

```
printf(format string, print list);
printf(format string);
```

EXAMPLES:

```
printf("I am %d years old, and my gpa is %f\n",
       age, gpa);
printf("Enter the object mass in grams> ");
```

INTERPRETATION: The `printf` function displays the value of its *format string* after substituting in left-to-right order the values of the expressions in the *print list* for their placeholders in the *format string* and after replacing escape sequences such as `\n` by their meanings.

Input/Output Operations and Functions

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf("Enter the distance in miles ");
  scanf("%lf", &miles);

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf(”Enter the distance in miles ”);
  scanf(”%lf”, &miles);

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf(”Enter the distance in miles ”);
  scanf(”%lf”, &miles);
- **The scanf Function:** The statement scanf(”%lf”, &miles);

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf(”Enter the distance in miles ”);
  scanf(”%lf”, &miles);
- **The scanf Function:** The statement scanf(”%lf”, &miles);

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf(”Enter the distance in miles ”);
  scanf(”%lf”, &miles);
- **The scanf Function:** The statement scanf(”%lf”, &miles);
- calls function scanf (pronounced “scan-eff”) to copy data into the variable miles.It copies the data from the standard input device.

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf(”Enter the distance in miles ”);
  scanf(”%lf”, &miles);
- **The scanf Function:** The statement scanf(”%lf”, &miles);
- calls function scanf (pronounced “scan-eff”) to copy data into the variable miles.It copies the data from the standard input device.

Input/Output Operations and Functions

- **Displaying Prompts :** When input data are needed in an interactive program, you should use the printf function to display a prompting message, or prompt, that tells the program user what data to enter. The printf statement below
- printf("Enter the distance in miles ");
  scanf("%lf", &miles);
- **The scanf Function:** The statement scanf("%lf", &miles);
- calls function scanf (pronounced "scan-eff") to copy data into the variable miles.It copies the data from the standard input device.
- In most cases the standard input device is the keyboard; consequently, the computer will attempt to store in miles whatever data the program user types at the keyboard.

Input/Output Operations and Functions

Input/Output Operations and Functions

- Notice that in a call to scanf, the name of each variable that is to be given a value is preceded by the ampersand character (&). The & is the C **address-of operator**.

Input/Output Operations and Functions

- Notice that in a call to scanf, the name of each variable that is to be given a value is preceded by the ampersand character (&). The & is the C **address-of operator**.
- In the context of this input operation, the & operator tells the scanf function where to find each variable into which it is to store a new value.

Input/Output Operations and Functions

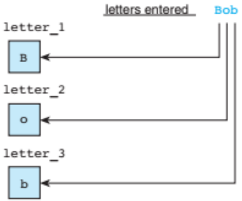
- Notice that in a call to scanf, the name of each variable that is to be given a value is preceded by the ampersand character (&). The & is the C **address-of operator.**
- In the context of this input operation, the & operator tells the scanf function where to find each variable into which it is to store a new value.
- What is the purpose of the following function call scanf("%c%c%c", &letter_1, &letter_2, &letter_3);

Input/Output Operations and Functions

- Notice that in a call to scanf, the name of each variable that is to be given a value is preceded by the ampersand character (&). The & is the C **address-of operator.**
- In the context of this input operation, the & operator tells the scanf function where to find each variable into which it is to store a new value.
- What is the purpose of the following function call scanf("%c%c%c", &letter_1, &letter_2, &letter_3);

Input/Output Operations and Functions

- Notice that in a call to scanf, the name of each variable that is to be given a value is preceded by the ampersand character (&). The & is the C **address-of operator**.
- In the context of this input operation, the & operator tells the scanf function where to find each variable into which it is to store a new value.
- What is the purpose of the following function call scanf("%c%c%c", &letter_1, &letter_2, &letter_3);



Input/Output Operations and Functions

Input/Output Operations and Functions

- Syntax Display for scanf Function Call

Input/Output Operations and Functions

- Syntax Display for scanf Function Call

Input/Output Operations and Functions

- Syntax Display for scanf Function Call

Syntax Display for scanf Function Call

SYNTAX: `scanf(`*format string, input list*`);`

EXAMPLE: `scanf("%c%d", &first_initial, &age);`

INTERPRETATION: The `scanf` function copies into memory data typed at the keyboard by the program user during program execution. The *format string* is a quoted string of placeholders, one placeholder for each variable in the *input list*. Each `int`, `double`, or `char` variable in the *input list* is preceded by an ampersand (`&`). Commas are used to separate variable names. The order of the placeholders must correspond to the order of the variables in the *input list*.

You must enter data in the same order as the variables in the *input list*. You should insert one or more blank characters or carriage returns between numeric items. If you plan to insert blanks or carriage returns between character data, you must include a blank in the format string before the `%c` placeholder.

Input/Output Operations and Functions

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);
- transfers control from your program to the operating system.

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);
- transfers control from your program to the operating system.

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);
- transfers control from your program to the operating system.
- The value in parentheses, 0, is considered the result of function main's execution, and it indicates that your program executed without error.

Input/Output Operations and Functions

- **The return Statement :** The last line in the main function return (0);
- transfers control from your program to the operating system.
- The value in parentheses, 0, is considered the result of function main's execution, and it indicates that your program executed without error.

- **The return Statement :** The last line in the main function return (0);
- transfers control from your program to the operating system.
- The value in parentheses, 0, is considered the result of function main's execution, and it indicates that your program executed without error.

Syntax Display for return Statement

SYNTAX: `return` *expression*`;`

EXAMPLE: `return (0);`

INTERPRETATION: The `return` statement transfers control from a function back to the activator of the function. For function `main`, control is transferred back to the operating system. The value of *expression* is returned as the result of the function execution.

General Form of a C Program

General Form of a C Program

preprocessor directives
main function heading
{
  declarations
  executable statements
}

General Form of a C Program

preprocessor directives
main function heading
{
  declarations
  executable statements
}

- A simple C program defines the main function after the preprocessor directives.

General Form of a C Program

preprocessor directives
main function heading
{
  declarations
  executable statements
}

- A simple C program defines the main function after the preprocessor directives.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.
- These variables are followed by the statements that are translated into machine language and are eventually executed.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.
- These variables are followed by the statements that are translated into machine language and are eventually executed.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.
- These variables are followed by the statements that are translated into machine language and are eventually executed.
- The statements we have looked at so far perform computations or input/output operations.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.
- These variables are followed by the statements that are translated into machine language and are eventually executed.
- The statements we have looked at so far perform computations or input/output operations.

General Form of a C Program


```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

- A simple C program defines the main function after the preprocessor directives.
- An open curly brace ({) signals the beginning of the main function body
- Within this body, we first see the declarations of all the variables to be used by the main function.
- These variables are followed by the statements that are translated into machine language and are eventually executed.
- The statements we have looked at so far perform computations or input/output operations.

Program Style Using Comments

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name
- the date of the current version

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name
- the date of the current version

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name
- the date of the current version
- a brief description of what the program does

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name
- the date of the current version
- a brief description of what the program does

Program Style Using Comments

- Each program should begin with a header section that consists of a series of comments specifying
- the programmer's name
- the date of the current version
- a brief description of what the program does

Program Comment

SYNTAX: `/* comment text */`

EXAMPLES:

```
/* This is a one-line or partial-line comment */
/*
 * This is a multiple-line comment in which the stars
 * not immediately preceded or followed by slashes
 * have no special syntactic significance, but simply
 * help the comment to stand out as a block. This
 * style is often used to document the purpose of a
 * program.
 */
```

INTERPRETATION: A slash-star indicates the start of a comment; a star-slash indicates the end of a comment. Comments are listed with the program but are otherwise ignored by the C compiler. A comment may be put in a C program anywhere a blank space would be valid.

Note: ANSI C does not permit the placement of one comment inside another.

Program Style Using Comments

Program Style Using Comments

- If you write the program for a class assignment, you should also list the class identification and your instructor's name.

Program Style Using Comments

- If you write the program for a class assignment, you should also list the class identification and your instructor's name.

Program Style Using Comments

- If you write the program for a class assignment, you should also list the class identification and your instructor's name.
- Before you implement each step in the initial algorithm, you should write a comment that summarizes the purpose of the algorithm step.

Program Style Using Comments

- If you write the program for a class assignment, you should also list the class identification and your instructor's name.
- Before you implement each step in the initial algorithm, you should write a comment that summarizes the purpose of the algorithm step.

Program Style Using Comments

- If you write the program for a class assignment, you should also list the class identification and your instructor's name.
- Before you implement each step in the initial algorithm, you should write a comment that summarizes the purpose of the algorithm step.
- This comment should describe what the step does rather than simply restate the step in English.For example, the comment

Program Style Using Comments

Program Style Using Comments


```
/*
 * Programmer: William Bell    Date completed: May 9, 2003
 * Instructor: Janet Smith     Class: CIS61
 *
 * Calculates and displays the area and circumference of a
 * circle
 */
```

Program Style Using Comments


```
/*
 * Programmer: William Bell     Date completed: May 9, 2003
 * Instructor: Janet Smith      Class: CIS61
 *
 * Calculates and displays the area and circumference of a
 * circle
 */
```

```
/* Convert the distance to kilometers. */
kms = KMS_PER_MILE * miles;
```

is more descriptive and hence preferable to

```
/* Multiply KMS_PER_MILE by miles and store result in kms. */
kms = KMS_PER_MILE * miles;
```

Practice Questions ???

Practice Questions ???

1. Change the following comments so they are syntactically correct.

```
/* This is a comment? *\
/* This one /* seems like a comment */ doesn't it */
```

2. Correct the syntax errors in the following program, and rewrite the program so that it follows our style conventions. What does each statement of your corrected program do? What output does it display?

```
/*
 * Calculate and display the difference of two input values
 *)
#include <stdio.h>
int
main(void) {int X, /* first input value */ x, /* second
  input value */
sum; /* sum of inputs */
scanf("%i%i"; X; x); X + x = sum;
printf("%d + %d = %d\n"; X; x; sum); return (0);}
```

Arithmetic Expressions

Arithmetic Expressions

- To solve most programming problems, you will need to write arithmetic expressions that manipulate type int and double data.

Arithmetic Expressions

- To solve most programming problems, you will need to write arithmetic expressions that manipulate type int and double data.

Arithmetic Expressions

- To solve most programming problems, you will need to write arithmetic expressions that manipulate type int and double data.

Arithmetic Operator	Meaning	Examples
+	addition	`5 + 2 is 7` `5.0 + 2.0 is 7.0`
-	subtraction	`5 - 2 is 3` `5.0 - 2.0 is 3.0`
*	multiplication	`5 * 2 is 10` `5.0 * 2.0 is 10.0`
/	division	`5.0 / 2.0 is 2.5` `5 / 2 is 2`
%	remainder	`5 % 2 is 1`

Mixed-Type Assignment Statement

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=). Either a type double or a type int expression may be assigned to a type double variable.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=). Either a type double or a type int expression may be assigned to a type double variable.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=). Either a type double or a type int expression may be assigned to a type double variable.
- if m and n are type int and p, x, and y are type double, the statements that follow assign the values shown in the boxes.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=). Either a type double or a type int expression may be assigned to a type double variable.
- if m and n are type int and p, x, and y are type double, the statements that follow assign the values shown in the boxes.

Mixed-Type Assignment Statement

- An expression that has operands of both type int and double is a mixed-type expression. The data type of such a mixed-type expression will be double.
- When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=). Either a type double or a type int expression may be assigned to a type double variable.
- if m and n are type int and p, x, and y are type double, the statements that follow assign the values shown in the boxes.

```
m = 3;
n = 2;
p = 2.0;
x = m / p;
y = m / n;
```

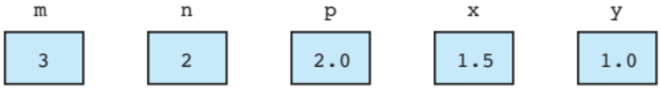
Mixed-Type Assignment Statement

Mixed-Type Assignment Statement

Mixed-Type Assignment Statement

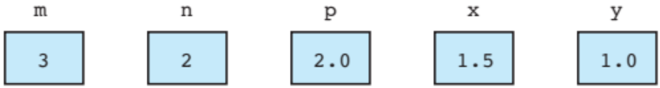
- In a mixed-type assignment such as $y = m/n$;

Mixed-Type Assignment Statement



- In a mixed-type assignment such as $y = m/n$;

Mixed-Type Assignment Statement

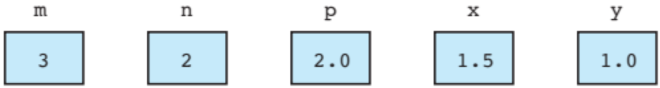


- In a mixed-type assignment such as $y = m/n$;
- A common error is to assume that because y is type double, the expression will be evaluated as if m and n were also type double instead of type int.

Mixed-Type Assignment Statement

- In a mixed-type assignment such as $y = m/n$;
- A common error is to assume that because y is type double, the expression will be evaluated as if m and n were also type double instead of type int.

Mixed-Type Assignment Statement



- In a mixed-type assignment such as $y = m/n$;
- A common error is to assume that because y is type double, the expression will be evaluated as if m and n were also type double instead of type int.
- Remember, the expression is evaluated before the assignment is made, and the type of the variable being assigned has no effect whatsoever on the expression value.

Type Conversion through Casts

Type Conversion through Casts

- C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a type cast.

Type Conversion through Casts

- C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a type cast.

Type Conversion through Casts

- C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a type cast.
- Assignment of a type double expression to a type int variable causes the fractional part of the expression to be lost since it cannot be represented in a type int variable.

Type Conversion through Casts

- C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a type cast.
- Assignment of a type double expression to a type int variable causes the fractional part of the expression to be lost since it cannot be represented in a type int variable.

Type Conversion through Casts

- C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a type cast.
- Assignment of a type double expression to a type int variable causes the fractional part of the expression to be lost since it cannot be represented in a type int variable.
- The expression in the assignment statements
  $x = 9 * 0.5;$
  $n = 9 * 0.5;$
  evaluates to the real number 4.5.

Type Conversion through Casts

Type Conversion through Casts

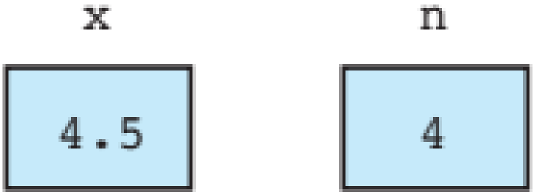
- If x is of type double, the number 4.5 is stored in x, as expected. If n is of type int, only the integral part of the expression value is stored in n, as shown.

Type Conversion through Casts

- If x is of type double, the number 4.5 is stored in x, as expected. If n is of type int, only the integral part of the expression value is stored in n, as shown.

Type Conversion through Casts

- If x is of type double, the number 4.5 is stored in x, as expected. If n is of type int, only the integral part of the expression value is stored in n, as shown.



Type Conversion through Casts

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.
- For example, you could use the following to find out the code your implementation uses for a question mark:

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.
- For example, you could use the following to find out the code your implementation uses for a question mark:

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.
- For example, you could use the following to find out the code your implementation uses for a question mark:

```
qmark_code = (int)'?';
printf("Code for ? = %d\n", qmark_code);
```

Type Conversion through Casts

- In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an int.
- Use a type cast to show that this happens.
  $n = (int)(9 * 0.5);$
- **Characters as Integers**Since characters are represented by integer codes, C permits conversion of type char to type int and vice versa.
- For example, you could use the following to find out the code your implementation uses for a question mark:

```
qmark_code = (int)'?';
printf("Code for ? = %d\n", qmark_code);
```

- You can perform arithmetic operations on characters. For example, the expression 'A' + 1 adds 1 to the code for 'A' and its value is the next character after 'A' which is 'B' in ASCII.

Rules for Evaluating Expressions

Rules for Evaluating Expressions

- **Parentheses rule:** All expressions in parentheses must be evaluated separately.Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

Rules for Evaluating Expressions

- **Parentheses rule:** All expressions in parentheses must be evaluated separately.Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

Rules for Evaluating Expressions

- **Parentheses rule:** All expressions in parentheses must be evaluated separately.Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- **Operator precedence rule:** Operators in the same expression are evaluated in the following order:
  unary +, - first
  *, /, % next
  binary +, - last

Rules for Evaluating Expressions

- **Parentheses rule:** All expressions in parentheses must be evaluated separately.Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- **Operator precedence rule:** Operators in the same expression are evaluated in the following order:
  unary +, - first
  *, /, % next
  binary +, - last

Rules for Evaluating Expressions

- **Parentheses rule:** All expressions in parentheses must be evaluated separately.Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- **Operator precedence rule:** Operators in the same expression are evaluated in the following order:
  unary +, - first
  *, /, % next
  binary +, - last
- **Associativity rule:** Unary operators in the same subexpression and at the same precedence level such as + and - are evaluated right to left i.e right associativity.Binary operators in the same subexpression and at the same precedence level such as + and - are evaluated left to right i.e left associativity.

Writing Mathematical Formulas in C

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed
- Use parentheses when required to control the order of operator evaluation

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed
- Use parentheses when required to control the order of operator evaluation

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed
- Use parentheses when required to control the order of operator evaluation
- Two arithmetic operators can be written in succession if the second is a unary

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed
- Use parentheses when required to control the order of operator evaluation
- Two arithmetic operators can be written in succession if the second is a unary

Writing Mathematical Formulas in C

- Always specify multiplication explicitly by using the operator * where needed
- Use parentheses when required to control the order of operator evaluation
- Two arithmetic operators can be written in succession if the second is a unary

Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a+b}{c+d}$	`(a + b) / (c + d)`
4. $\frac{1}{1+x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Numerical Inaccuracies

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error an error :**resulting from applying an arithmetic operation to operands of vastly different magnitudes; effect of smaller operand is lost

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error an error :**resulting from applying an arithmetic operation to operands of vastly different magnitudes; effect of smaller operand is lost

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error an error :**resulting from applying an arithmetic operation to operands of vastly different magnitudes; effect of smaller operand is lost
- If two very small numbers are multiplied, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called **arithmetic underflow**.

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error an error :**resulting from applying an arithmetic operation to operands of vastly different magnitudes; effect of smaller operand is lost
- If two very small numbers are multiplied, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called **arithmetic underflow.**

Numerical Inaccuracies

- **Representational error :** an error due to coding a real number as a finite number of binary digits
- The representational error (sometimes called round-off error) will depend on the number of bits used in the mantissa: the more bits, the smaller the error.
- **cancellation error an error :**resulting from applying an arithmetic operation to operands of vastly different magnitudes; effect of smaller operand is lost
- If two very small numbers are multiplied, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called **arithmetic underflow.**
- if two very large numbers are multiplied, the result may be too large to be represented. This phenomenon, called **arithmetic overflow**

Formatting Numbers in Program Output

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.
- This number specifies the field width—the number of columns to use for the display of the value.
  printf(" Results: % 3d meters = % 4d ft. % 2d in.\n ", meters, feet, inches);

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.
- This number specifies the field width—the number of columns to use for the display of the value.
  printf(" Results: % 3d meters = % 4d ft. % 2d in.\n ", meters, feet, inches);

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.
- This number specifies the field width—the number of columns to use for the display of the value.
  printf(" Results: % 3d meters = % 4d ft. % 2d in.\n ", meters, feet, inches);
- **Formatting Values of Type double :**To describe the format specification for a type double value, we must indicate both the total field width needed and the number of decimal places desired.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.
- This number specifies the field width—the number of columns to use for the display of the value.
  printf(" Results: % 3d meters = % 4d ft. % 2d in.\n ", meters, feet, inches);
- **Formatting Values of Type double :**To describe the format specification for a type double value, we must indicate both the total field width needed and the number of decimal places desired.

Formatting Numbers in Program Output

- C displays all numbers in its default notation unless you instruct it to do otherwise.
- **Formatting Values of Type int :** You simply add a number between the % and the d of the %d placeholder in the printf format string.
- This number specifies the field width—the number of columns to use for the display of the value.
  printf(" Results: % 3d meters = % 4d ft. % 2d in.\n ", meters, feet, inches);
- **Formatting Values of Type double :**To describe the format specification for a type double value, we must indicate both the total field width needed and the number of decimal places desired.
- The total field width should be large enough to accommodate all digits before and after the decimal point.

Formatting Type double Values

Formatting Type double Values

- The form of the format string placeholder is %n.mf where n is a number representing the total field width, and m is the desired number of decimal places.

Formatting Type double Values

- The form of the format string placeholder is %n.mf where n is a number representing the total field width, and m is the desired number of decimal places.

Formatting Type double Values

- The form of the format string placeholder is %n.mf where n is a number representing the total field width, and m is the desired number of decimal places.

Value	Format	Displayed Output	Value	Format	Displayed Output
`3.14159`	`%5.2f`	`▮3.14`	`3.14159`	`%4.2f`	`3.14`
`3.14159`	`%3.2f`	`3.14`	`3.14159`	`%5.1f`	`▮▮3.1`
`3.14159`	`%5.3f`	`3.142`	`3.14159`	`%8.5f`	`▮3.14159`
`.1234`	`%4.2f`	`0.12`	`-.006`	`%4.2f`	`-0.01`
`-.006`	`%8.3f`	`▮▮-0.006`	`-.006`	`%8.5f`	`-0.00600`
`-.006`	`%.3f`	`-0.006`	`-3.14159`	`%.4f`	`-3.1416`

Interactive Mode, Batch Mode, and Data Files

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user
- **Input Redirection**

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user
- **Input Redirection**

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user
- **Input Redirection**
  We assume here that the standard input device is associated with a batch data file instead of with the keyboard.

Interactive Mode, Batch Mode, and Data Files

- There are two basic modes of computer operation: batch mode and interactive mode
- In **interactive mode**, the program user interacts with the program and types in data while it is running.
- In **batch mode**, the program scans its data from a data file prepared beforehand instead of interacting with its user
- **Input Redirection**
  We assume here that the standard input device is associated with a batch data file instead of with the keyboard.
- In most systems, this association can be accomplished relatively easily through input/output redirection using operating system commands.

Interactive Mode, Batch Mode, and Data Files

- **Input Redirection**

Interactive Mode, Batch Mode, and Data Files

- **Input Redirection**

- **Input Redirection**
- For example, in the UNIX® and MS-DOS® operating systems, you can instruct your program to take its input from file mydata.txt instead of from the keyboard by placing the symbols $\langle$ mydata.txt at the end of the command line that causes your compiled and linked program to execute.

- **Output Redirection**

Interactive Mode, Batch Mode, and Data Files

- **Output Redirection**

Interactive Mode, Batch Mode, and Data Files

- **Output Redirection**
- You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer (using an operating system command) to obtain a hard copy of the program output. In UNIX or MS-DOS, use the symbols $\rangle$ myoutput.txt to redirect output from the screen to file myoutput.txt.

Interactive Mode, Batch Mode, and Data Files

- **Output Redirection**
- You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer (using an operating system command) to obtain a hard copy of the program output. In UNIX or MS-DOS, use the symbols $\rangle$ myoutput.txt to redirect output from the screen to file myoutput.txt.

- **Output Redirection**
- You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer (using an operating system command) to obtain a hard copy of the program output. In UNIX or MS-DOS, use the symbols $\rangle$ myoutput.txt to redirect output from the screen to file myoutput.txt.
- .\metric $\langle$mydata.txt $\rangle$myoutput.txt

- **Output Redirection**
- You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer (using an operating system command) to obtain a hard copy of the program output. In UNIX or MS-DOS, use the symbols ⟩ myoutput.txt to redirect output from the screen to file myoutput.txt.
- .\metric ⟨mydata.txt ⟩myoutput.txt

Interactive Mode, Batch Mode, and Data Files

- **Output Redirection**
- You can also redirect program output to a disk file instead of to the screen. Then you can send the output file to the printer (using an operating system command) to obtain a hard copy of the program output. In UNIX or MS-DOS, use the symbols $\rangle$ myoutput.txt to redirect output from the screen to file myoutput.txt.
- .\metric $\langle$mydata.txt $\rangle$myoutput.txt
- which takes program input from data file mydata.txt and sends program output to output file myoutput.txt.

Common Programming Errors

Common Programming Errors

- **debugging** removing errors from a program

Common Programming Errors

- **debugging** removing errors from a program

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- **Run-time errors** are detected and displayed by the computer during the execution of a program.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- **Run-time errors** are detected and displayed by the computer during the execution of a program.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- **Run-time errors** are detected and displayed by the computer during the execution of a program.
- A run-time error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- **Run-time errors** are detected and displayed by the computer during the execution of a program.
- A run-time error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.

Common Programming Errors

- **debugging** removing errors from a program
- A **syntax error** occurs when your code violates one or more grammar rules of C and is detected by the compiler as it attempts to translate your program.
- **Run-time errors** are detected and displayed by the computer during the execution of a program.
- A run-time error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero.
- **Logic errors** occur when a program follows a faulty algorithm. Because logic errors usually do not cause run-time errors and do not display error messages,they are very difficult to detect.

CASE STUDY Supermarket Coin Processor

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **PROBLEM :**

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **PROBLEM :**

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **PROBLEM :**
- You are drafting software for the machines placed at the front of supermarkets to convert change to personalized credit slips. In this draft, the user will manually enter the number of each kind of coin in the collection, but in the final version, these counts will be provided by code that interfaces with the counting devices in the machine.

CASE STUDY Supermarket Coin Processor

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **ANALYSIS:**

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **ANALYSIS:**

CASE STUDY Supermarket Coin Processor

- This case study demonstrates the manipulation of type int data (using / and %) and type char data.
- **ANALYSIS:**
- To solve this problem, you need to get the customer's initials to use in personalizing the credit slip along with the count of each type of coin (dollars, quarters, dimes,nickels, pennies). From the counts, you can determine the total value of the coins in cents. Once you have that figure, you can do an integer division using 100 as the divisor to get the dollar value; the remainder of this division will be the leftover change. In the data requirements, list the total value in cents (total_cents) as a program variable, because it is needed as part of the computation process but is not a required problem output.

DATA REQUIREMENTS

DATA REQUIREMENTS

- Problem Inputs

DATA REQUIREMENTS

- Problem Inputs

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies
  Problem Outputs

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies
  Problem Outputs
  int total_dollars /* total dollar value */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies
  Problem Outputs
  int total_dollars /* total dollar value */
  int change /* leftover change */

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies
  Problem Outputs
  int total_dollars /* total dollar value */
  int change /* leftover change */
  Additional Program Variables

DATA REQUIREMENTS

- Problem Inputs
  char first, middle, last /* a customer's initials */
  int dollars /* number of dollars */
  int quarters /* number of quarters */
  int dimes /* number of dimes */
  int nickels /* number of nickels */
  int pennies /* number of pennies
  Problem Outputs
  int total_dollars /* total dollar value */
  int change /* leftover change */
  Additional Program Variables
  int total_cents /* total value in cents */

DESIGN

DESIGN

- INITIAL ALGORITHM

DESIGN

- INITIAL ALGORITHM

DESIGN

- INITIAL ALGORITHM

  1. Get and display the customer's initials.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:
  Step 3 Refinement 3.1 Find the equivalent value of each kind of coin in pennies and add these values.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:
  Step 3 Refinement 3.1 Find the equivalent value of each kind of coin in pennies and add these values.
  Step 4 Refinement

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:
  Step 3 Refinement 3.1 Find the equivalent value of each kind of coin in pennies and add these values.
  Step 4 Refinement
  4.1 total_dollars is the integer quotient of total_cents and 100.

DESIGN

- INITIAL ALGORITHM

  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:

  Step 3 Refinement 3.1 Find the equivalent value of each kind of coin in pennies and add these values.

  Step 4 Refinement

  4.1 total_dollars is the integer quotient of total_cents and 100.

  4.2 change is the integer remainder of total_cents and 100.

DESIGN

- INITIAL ALGORITHM
  1. Get and display the customer's initials.
  2. Get the count of each kind of coin.
  3. Compute the total value in cents.
  4. Find the value in dollars and change.
  5. Display the value in dollars and change.

  Steps 3 and 4 may need refinement. Their refinements are:
  Step 3 Refinement 3.1 Find the equivalent value of each kind of coin in pennies and add these values.
  Step 4 Refinement
  4.1 total_dollars is the integer quotient of total_cents and 100.
  4.2 change is the integer remainder of total_cents and 100.

IMPLEMENTATION

IMPLEMENTATION

- Now Write the C Program.