1. Introduction to C

1.1 Programming

- **computer program**

  A computer program consists of instructions that a computer executes (or *runs*), like multiplying numbers or printing a number to a screen.

1.2 A first program

- **Code**

  Code is the textual representation of a program.

- **line**

  A line is a row of text.

- **main**

  The program starts by executing a function called main.

- **braces**

  "**{**" and "**}**" are called braces, denoting a list of statements.

- **statement**

  A statement is a program instruction.

- **semicolon**

- **memory**

  A memory is composed of numerous individual locations, each able to store a value.

- **compiler**

  A compiler is a tool that converts a program into low-level machine instructions (0s and 1s) understood by a particular computer.

1.3 Basic output

- **printf**

  The printf construct supports printing.

- **string literal**

  Text in double quotes " " is known as a string literal.

- **newline**

  A newline character, \n, in the string literal starts a new output line.

- **%d**

  The %d in the string literal indicates that a decimal number (hence the d) should be printed there, with that number specified by the variable that follows.

1.4 Basic input

- **scanf**

  Reading a decimal number input is achieved using the statement: scanf ("%d", &variableName).

1.5 Comments and whitespace

- **comment**

A comment is text added to code by a programmer, intended to be read by humans to better understand the code, but ignored by the compiler.

- **single-line comment**

  A single-line comment uses the // symbols.

- **multi-line comment**

  A multi-line comment uses the /* and */ symbols.

- **block comment**

  A multi-line comment is also known as a block comment.

- **Whitespace**

  Whitespace refers to blank spaces between items within a statement, and to blank lines between statements.

1.6 Errors and warnings

- **syntax error**

One kind of mistake, known as a syntax error, is to violate a programming language's rules on how symbols can be combined to create a program.

- **compile-time error**

  Because a syntax error is detected by the compiler, a syntax error is known as a type of compile-time error.

- **logic error**

  A logic error is an error that occurs while a program runs, also called a runtime error or bug.

- **runtime error**

  A logic error is an error that occurs while a program runs, also called a runtime error or bug.

- **bug**

  A logic error is an error that occurs while a program runs, also called a runtime error or bug.

-

warning

A compiler will sometimes report a warning, which doesn't stop the compiler from creating an executable program, but indicates a possible logic error.

1.7 Computers and programs

- **bits**

  0s and 1s are known as bits (*b*inary dig*its*).

- **processors**

  To support different calculations, circuits called processors were created to process (aka *execute*) a list of desired calculations.

- **instruction**

- **memory**

  A memory is a circuit that can store 0s and 1s in each of a series of thousands of addressed locations.

-

program

- **application**

- **app**

- **machine instructions**

  Instructions represented as 0s and 1s are known as machine instructions.

- **executable program**

  A sequence of machine instructions together form an executable program.

- **assembly**

- **high-level languages**

- **compilers**

  To support high-level languages, programmers created compilers, which are programs that automatically translate high-level language programs into executable programs.

1.8 Computer tour

- **Input/output devices**

- **screen**

  A screen (or monitor) displays items to a user.

- **keyboard**

  A keyboard allows a user to provide input to the computer.

- **Storage**

-

disk

A disk (aka *hard drive*) stores files and other data, such as program files, song/movie files, or office documents.

- **Memory**

- **RAM**

  RAM (random-access memory) temporarily holds data read from storage, and is designed such that any address can be accessed much faster than disk, in just a few clock ticks (see below) rather than hundreds of ticks.

- **byte**

  A byte is 8 bits.

- **Processor**

- **processor**

The processor runs the computer's programs, reading and executing instructions from memory, performing operations, and reading/writing data from/to memory.

- **operating system**

  The operating system allows a user to run other programs and which interfaces with the many other peripherals.

- **cache**

  A processor may contain a small amount of RAM on its own chip, called cache memory, accessible in one clock tick rather than several, for maintaining a copy of the most-used instructions/data.

- **Clock**

- **clock**

  A processor's instructions execute at a rate governed by the processor's clock, which ticks at a specific frequency.

- **transistors**

Engineers created smaller switches called transistors, which in 1958 were integrated onto a single chip.

- **integrated circuit**

- **Moore's Law**

  Moore's Law: The doubling of IC capacity roughly every 18 months, which continues today.

1.9 Language history

- **C**

  In 1978, Brian Kernighan and Dennis Ritchie at AT&T Bell Labs (which used computers extensively for automatic phone call routing) published a book describing a new high-level language with the simple name C.

- **C++**

  In 1985, Bjarne Stroustrup published a book describing a C-based language called C++, adding constructs to support a style of programming known as object-oriented programming, along with other improvements.

1.10 Problem solving

- **problem solving**

  Much of programming is about problem solving: Creating a methodical solution to a given task.

1.11 C example: Salary Calculation

No terms in this section.

1.12 C example: Married-couple names

No terms in this section.

2. Variables / Assignments

2.1 Variables (int)

- **variable**

  A variable represents a memory location used to store data.

- **declares**

- **address**

2.2 Assignments

-

assignment statement

An assignment statement like `numApples = 8;` stores (i.e. assigns) the right-side item's current value (in this case, 8) into the variable on left side (numApples).

- ### expression

  An expression may be a number like 80, a variable name like numApples, or a simple calculation like numApples + 1.

2.3 Identifiers

- ### identifier

  A name created by a programmer for an item like a variable or function is called an identifier.

- ### underscore

- ### reserved word

  A reserved word is a word that is part of the language, like int, short, or double.

-

keyword

- **case sensitive**

  Identifiers are case sensitive, meaning upper and lower case letters differ.

- **naming conventions**

- **Lower camel case**

  Lower camel case abuts multiple words, capitalizing each word except the first, as in numApples or peopleOnBus.

2.4 Arithmetic expressions (int)

- **expression**

  An expression is a combination of items, like variables, literals, and operators, that evaluates to a value.

- **literal**

  A literal is a specific value in code like 2.

- **operator**

  An operator is a symbol for a built-in language calculation like + for addition.

- **Arithmetic operators**

- **unary minus**

  - used as negative is known as unary minus.

- **divide-by-zero error**

  A divide-by-zero error occurs at runtime if a divisor is 0, causing a program to terminate.

- **precedence rules**

  The compiler evaluates an expression's arithmetic operators using the order of standard mathematics, such order known in programming as precedence rules.

- **compound operators**

  Special operators called compound operators provide a shorthand way to update a variable, such as userAge += 1 being shorthand for userAge = userAge + 1.

- **+**

- **addition**

- **-**

- **subtraction**

- ***

- **multiplication**

- **/**

- **division**

- **%**

- **modulo (remainder)**

2.5 Floating-point numbers (double)

- **double**

  A variable declared as type double stores a floating-point number.

- **floating-point literal**

  A floating-point literal is a number with a fractional part, even if that fraction is 0, as in 1.0, 0.0, or 99.573.

- **%lf**

  Scanf and printf use %lf to specify a double type in the string literal, in contrast to %d for an int type.

- **scientific notation**

  A floating-point literal using scientific notation is written using an e preceding the power-of-10 exponent, as in 6.02e23 to represent 6.02×10^{23}.

- **floating-point divide-by-zero**

  A floating-point divide-by-zero occurs at runtime if a divisor is 0.0. Dividing by zero results in infinity or -infinity depending on the signs of the operands.

2.6 Constant variables

- **const**

- **constant variable**

  An initialized variable whose value cannot change is called a constant variable.

2.7 Using math functions

- **math library**

  Thus, the language comes with a standard math library that has about 20 math operations available for floating-point values, listed later in this section.

- **function**

  A function is a list of statements that can be executed by referring to the function's name.

- **argument**

  An input value to a function appears between parentheses and is known as an argument.

- **function call**

  Invoking a function is a function call.

- **abs()**

- pow
- cos
- sqrt
- sin
- tan
- exp
- acos

- log
- asin
- log10
- atan
- atan2
- ceil
- cosh
-

fabs

- sinh
- floor
- tanh
- fmod
- abs
- frexp
- ldexp

- **modf**

2.8 Type conversions

- **type conversion**

  A type conversion is a conversion of one data type to another, such as an int to a double.

- **implicit conversion**

  The compiler automatically performs several common conversions between int and double types, such automatic conversion known as implicit conversion.

- **(type)**

- **type casting**

  Explicit conversion by the programmer of one type to another is known as type casting.

2.9 Binary

- **binary number**

  Because each memory location is composed of bits (0s and 1s), a processor stores a number using base 2, known as a binary number.

- **decimal number**

  For a number in the more familiar base 10, known as a decimal number, each digit must be 0-9 and each digit's place is weighed by increasing powers of 10.

- **base 2**

  In base 2, each digit must be 0-1 and each digit's place is weighed by increasing powers of 2.

2.10 Characters

- **char**

  A variable of char type can store a single character, like the letter m or the symbol %.

- **character literal**

A character literal is surrounded with single quotes, as in 'm' or '%'.

- **ASCII**

  ASCII is an early standard for encoding characters as numbers.

- **escape sequence**

  An escape sequence is a two-character sequence starting with \ that represents a special character.

2.11 String basics

- **string**

  A sequence of characters is called a string.

- **character array**

- **C string**

  This material may refer to a character array as a string or a C string.

- **null character**

  Strings are always terminated with a special character called the null character, '\0'.

- **whitespace character**

  A whitespace character is a character used to print spaces in text, and includes spaces, tabs, and newline characters.

- **strcpy**

  The programmer assigns a value to a string using the function strcpy(str1, str2), which copies each character in str2 into corresponding locations of str1.

2.12 Integer overflow

- **overflow**

  An overflow occurs when the value being assigned to a variable is greater than the maximum value the variable can store.

- **compiler warning**

The compiler may not report a syntax error (the syntax is correct), but may output a compiler warning message that indicates a potential problem.

2.13 Numeric data types

- **long long**

  Long long is used for integers expected to exceed about 2 billion.

- **overflow**

  An overflow occurs when the value being assigned to a variable is greater than the maximum value the variable can store.

2.14 Unsigned

No terms in this section.

2.15 Random numbers

- **rand()**

  The rand() function, in the C standard library, returns a random integer each time the function is called, in the range 0 to RAND_MAX.

- **seed**

For the first call to rand(), no previous random integer exists, so the function uses a built-in integer known as the seed.

- **time()**

  The function time() returns the number of seconds since Jan 1, 1970.

2.16 The printf and scanf functions

- **format string**

  The format string defines the format of the text that will be printed along with any number of placeholders, known as format specifiers, for printing numeric values and text stored in variables.

- **format specifier**

  A format specifier is a placeholder that defines the type of value that will be printed in its place.

2.17 Debugging

- **Debugging**

  Debugging is the process of determining and fixing the cause of a problem in a computer program.

- **Troubleshooting**

  Troubleshooting is another word for debugging.

2.18 Style guidelines

- **style guide**

  Each programming team, whether a company or a classroom, may have its own style for writing code, sometimes called a style guide.

- **K&R style**

  K&R style for braces and indents is named after C language creators Kernighan and Ritchie.

- **Stroustrup style**

  Stroustrup style for braces and indents is named after C++ language creator Bjarne Stroustrup.

2.19 C example: Salary calculation with variables

No terms in this section.

2.20 C example: Married-couple names with variables

No terms in this section.

3. Branches

3.1 If-else

- **branching**

  Branching directs a program to execute either one statement group or another, depending on an expression's value.

- **Braces**

  Braces { }, sometimes redundantly called curly braces, represent a grouping, such as a grouping of statements.

3.2 Relational and equality operators

- **relational operator**

- **equality operator**

- **Boolean**

3.3 Multiple if-else branches

- **nested if-else**

  A branch's statements can include any valid statements, including another if-else statement, such occurrence known as nested if-else statements.

3.4 Logical operators

- **logical operator**

  A logical operator treats operands as being true or false, and evaluates to true or false.

- **bool**

  Bool (short for Boolean) data type is for variables that should store only values true or false.

- **Binary operators**

  Binary operators take two operands (from the left and right) and evaluate to true or false.

- **bitwise**

3.5 Switch statements

- **switch**

  A switch statement can more clearly represent multi-branch behavior involving a variable being compared to constant values.

- **case**

- **default case**

- **break**

3.6 Boolean data types

- **Boolean**

  Boolean refers to a quantity that has only two possible values, true or false.

- **bool**

The language has the built-in data type bool for representing Boolean quantities.

3.7 String comparisons

- **strcmp**

  The strcmp function returns 0 if the strings are equal, and some non-zero value otherwise.

3.8 String access operations

- **index**

  Each string character has a position number called an index.

- **null character**

  So that code can detect where a string ends, the compiler ends a string with a null character, written as '\0'.

- **strlen**

3.9 Character operations

-

ctype.h library

Including the ctype.h library via `#include <ctype.h>` provides access to several functions for working with characters.

- **isalpha**
- **toupper**
- **isdigit**
- **tolower**
- **isspace**

3.10 Conditional expressions

- **conditional expression**

- **ternary operator**

3.11 Floating-point comparison

- **epsilon**

  The difference threshold indicating that floating-point numbers are equal is often called the epsilon.

3.12 Short circuit evaluation

- **Short circuit evaluation**

  Short circuit evaluation skips evaluating later operands if the result of the logical operator can already be determined. .

3.13 C example: Salary calculation with branches

No terms in this section.

3.14 C example: Search for name using branches

- **core generic top-level domain (core gTLD)**

  A core generic top-level domain (core gTLD) name is one of the following Internet domains: .com, .net, .org, and .info.

4. Loops

4.1 Loops

- **loop**

  A loop is a construct that repeatedly executes specific code as long as some condition is true.

4.2 While loops

- **while loop**

  A while loop is a program construct that executes a list of sub-statements repeatedly as long as the loop's expression evaluates to true.

- **loop body**

  The sub-statements inside the braces, known as the loop body.

- **iteration**

  Each execution of the loop body is called an iteration.

- **infinite loop**

An infinite loop is a loop that will always execute (i.e., execute infinitely) because the loop's expression always evaluates to true.

4.3 More while examples

No terms in this section.

4.4 Counting

- **loop variable**

  A loop variable counts the number of iterations of a loop.

- **++i**

  Because i = i + 1 is so common in programs, the programming language provides a shorthand version ++i.

- **increment operator**

  The ++ is known as the increment operator.

- **decrement operator**

  The decrement operator, as in --i, is equivalent to i = i - 1.

4.5 For loops

- **for loop**

  A for loop statement collects three parts—the loop variable initialization, loop expression, and loop variable update—all at the top of the loop.

4.6 Nested loops

- **nested loop**

  A nested loop is a loop that appears in the body of another loop.

- **inner loop**

- **outer loop**

4.7 Developing programs incrementally

- **incrementally**

  Experienced programmers develop programs incrementally, meaning they create a simple program version, and then growing the program little-by-little into successively more-complete versions.

- **FIXME comment**

  A FIXME comment is commonly used to indicate program parts to be fixed or added.

4.8 Break and continue

- **break statement**

  A break statement in a loop causes an immediate exit of the loop.

- **continue statement**

  A continue statement in a loop causes an immediate jump to the loop condition check.

4.9 Enumerations

- **enumeration type**

  An enumeration type declares a name for a new type and possible values for that type.

- **state machine**

4.10 C example: Salary calculation with loops

No terms in this section.

4.11 C example: Domain name validation with loops

- **top-level domain**

  A top-level domain (TLD) name is the last part of an Internet domain name like .com in example.com.

- **core generic top-level domain**

  A core generic top-level domain (core gTLD) is a TLD that is either .com, .net, .org, or .info.

- **second-level domain**

  A second-level domain is a single name that precedes a TLD as in apple in apple.com .

5. Arrays

5.1 Array concept

- **array**

  An array is a special variable having one name, but storing a list of data items, with each item directly accessible.

- **vector**

  Some languages use a construct similar to an array called a vector.

- **element**

  Each item in an array is known as an element.

- **index**

  In an array, each element's location number is called the index; myArray[2] has index 2.

5.2 Arrays

- **array**

  An array is an ordered list of items of a given data type.

- **element**

  Each item in an array is called an element.

- **brackets**

  [ ] are brackets.

- **braces**

  { } are braces.

- **index**

  In an array access, the number in brackets is called the index of the corresponding element.

- **const**

5.3 Array iteration drill

No terms in this section.

5.4 Iterating through arrays

No terms in this section.

5.5 Multiple arrays

No terms in this section.

5.6 Swapping two variables

- **Swapping**

  Swapping two variables x and y means to assign y's value to x, and x's value to y.

5.7 Loop-modifying or copying/comparing arrays

No terms in this section.

5.8 Debugging example: Reversing an array

No terms in this section.

5.9 Two-dimensional arrays

- **row-major order**

  The compiler maps two-dimensional array elements to one-dimensional memory, each row following the previous row, known as row-major order.

5.10 Char arrays / C strings

- **string**

  A programmer can use an array to store a sequence of characters, known as a string.

- **null character**

A string in a char array must end with a special character known as a null character, written as '\0'.

- **null-terminated string**

  An array of characters ending with a null character is known as a null-terminated string.

5.11 String library functions

- **string.h**
- **strcpy()**
- **strncpy()**
- **strcat()**
- **strncat()**

- **strchr()**
- **strlen()**
- **strcmp()**
- **scanf() and fgets() in stdio.h**
- **scanf**
- **fgets**

5.12 Char library functions: ctype

- **Character checking functions**

- **isalpha(c)**
- **isdigit(c)**
- **isalnum(c)**
- **isspace(c)**
- **islower(c)**
- **isupper(c)**
- **isblank(c)**

- **isxdigit(c)**
- **ispunct(c)**
- **isprint(c)**
- **iscntrl(c)**
- **Character conversion functions**
- **toupper(c)**
- **tolower(c)**

5.13 Arrays and strings

No terms in this section.

5.14 C example: Salary calculation with arrays

No terms in this section.

5.15 C example: Domain name validation with arrays

- **top-level domain**

  A top-level domain (TLD) name is the last part of an Internet domain name like .com in example.com.

- **core generic top-level domain**

  A core generic top-level domain (core gTLD) is a TLD that is either .com, .net, .org, or .info.

- **restricted top-level domain**

  A restricted top-level domain is a TLD that is either .biz, .name, or .pro.

- **second-level domain**

  A second-level domain is a single name that precedes a TLD as in apple in apple.com.

6. User-Defined Functions

6.1 User-defined function basics

- **function**

  A function is a named list of statements.

- **function call**

  Invoking a function's name, known as a function call, causes the function's statements to execute.

- **function definition**

  A function definition consists of the new function's name and a block of statements.

- **block**

  A block is a list of statements surrounded by braces.

- **return**

  Return causes execution to jump back to the original calling location.

6.2 Parameters

- **parameter**

  Programmers can influence a function's behavior via an input to the function known as a parameter.

- **argument**

  The value passed to a parameter is known as an argument.

6.3 Return

- **return statement**

  A function may return a value using a return statement.

- **void**

  A return type of void indicates that a function does not return any value, in which case the return statement should simply be: `return;`.

- **hierarchical function calls**

- **nested function calls**

6.4 Reasons for defining functions

- **module**

- **function stubs**

6.5 Functions with branches/loops

No terms in this section.

6.6 Unit testing (functions)

- **Unit testing**

  Unit testing is the process of individually testing a small part or unit of a program, typically a function.

- **testbench**

  A unit test is typically conducted by creating a testbench, a.k.a. test harness, which is a separate program whose sole purpose is to check that a function returns correct output values for a variety of input values.

- **test vector**

  Each unique set of input values is known as a test vector.

- **border cases**

6.7 How functions work

- **stack frame**

6.8 Functions: Common errors

No terms in this section.

6.9 Pass by pointer

- **pass by value**

  Normal parameters are pass by value, meaning the argument's value is copied into a local variable for the parameter.

- **pass by reference**

  Defining a parameter as a pointer to enable updating the argument variable is commonly known as pass by reference.

- **pass by pointer**

6.10 Functions with array parameters

- **const**

  The keyword const can be prepended to a function's array parameter to prevent the function from modifying the parameter.

6.11 Functions with C string parameters

No terms in this section.

6.12 Functions with array parameters: Common errors

No terms in this section.

6.13 Scope of variable/function definitions

- **scope**

  The name of a defined variable or function item is only visible to part of a program, known as the item's scope.

- **global variable**

  A variable declared outside any function is called a global variable, in contrast to a *local variable* declared inside a function.

- **side effects**

  If a function updates a global variable, the function has effects that go beyond its parameters and return value, known as side effects,.

- **function declaration**

  A function declaration specifies the function's return type, name, and parameters, ending with a semicolon where the opening brace would have gone.

- **function prototype**

  A function declaration is also known as a function prototype.

6.14 Preprocessor and include

- **preprocessor**

  The preprocessor is a tool that scans the file from top to bottom looking for any lines that begin with #, known as a hash symbol. Each such line is not a program statement, but rather directs the preprocessor to modify the file in some way before compilation continues, each such line being known as a preprocessor directive.

-

hash symbol

The preprocessor is a tool that scans the file from top to bottom looking for any lines that begin with #, known as a hash symbol. Each such line is not a program statement, but rather directs the preprocessor to modify the file in some way before compilation continues, each such line being known as a preprocessor directive.

- ### preprocessor directive

  The preprocessor is a tool that scans the file from top to bottom looking for any lines that begin with #, known as a hash symbol. Each such line is not a program statement, but rather directs the preprocessor to modify the file in some way before compilation continues, each such line being known as a preprocessor directive.

- ### #include

- ### include directive

6.15 Separate files

- ### Header file guards

  Header file guards are preprocessor directives cause the compiler to only include the contents of the header file once.

6.16 C example: Salary calculation with functions

No terms in this section.

6.17 C example: Domain name validation with functions

- **top-level domain**

  A top-level domain (TLD) name is the last part of an Internet domain name like .com in example.com.

- **core generic top-level domain**

  A core generic top-level domain (core gTLD) is a TLD that is either .com, .net, .org, or .info.

- **restricted top-level domain**

  A restricted top-level domain is a TLD that is either .biz, .name, or .pro.

- **second-level domain**

  A second-level domain is a single name that precedes a TLD as in apple in apple.com .

7. Structs

7.1 Grouping data: struct

- **struct**

  The struct construct defines a new type, which can be used to declare a variable with subitems.

- **typedef**

  A typedef defines a new type name for an existing type.

- **data member**

  Each struct subitem is called a data member.

- **member access**

  For a declared variable, each struct data member can be accessed using ".", known as a member access operator.

- **dot notation**

7.2 Structs and functions

No terms in this section.

7.3 Structs and arrays

No terms in this section.

7.4 Structs, arrays, and functions: A seat reservation example

No terms in this section.

7.5 Separate files for structs

- **StructName.h**

- **StructName.c**

8. Pointers

8.1 Why pointers: Pass by pointer example

- **pass by pointer**

- **pointer**

  The & passes the variable's memory address, known as a pointer, rather than the variable's value.

-

- **pass by reference**

8.2 Pointer basics

- **pointer**

  A pointer is a variable that contains a memory address, rather than containing data like most variables introduced earlier.

- **dereferencing**

- **NULL**

8.3 The malloc and free functions

- **malloc()**

- **standard library**

- **void pointer**

- **free()**

8.4 Pointers with structs

- **member access operator**

8.5 String functions with pointers

- **strchr()**
- **strrchr()**
- **strstr()**

8.6 The malloc function for arrays and strings

- **statically allocated**

- **dynamically allocated**

8.7 The realloc function

- **realloc**

8.8 Vector ADT

- **abstract data type (ADT)**

  Abstract data type (ADT), which is a data type whose creation and update are supported by specific well-defined operations.

- **information hiding**

  Internal implementation of the data and operations are hidden from the ADT user, a concept known as information hiding.

- **segmentation fault**

- **access violation**

- **bad access**

8.9 Why pointers: A list example

- **vector insert/erase performance problem**

- **linked list**

8.10 A first linked list

- **list node**

8.11 Memory regions: Heap/Stack

- **Code**

- **Static memory**

- **The stack**

- **automatic memory**

- **The heap**

- **free store**

8.12 Memory leaks

- **memory leak**

  A program that allocates memory but then loses the ability to access that memory, typically due to failure to properly destroy/free dynamically allocated memory, is said to have a memory leak.

- **garbage collection**

  Some programming languages, such as Java, use a mechanism called garbage collection wherein a program's executable includes automatic behavior that at various intervals finds all unreachable allocated memory locations (e.g., by comparing all reachable memory with all

previously-allocated memory), and automatically freeing such unreachable memory.

8.13 C example: Employee list using vector ADT

No terms in this section.

9. Input / Output

9.1 The stdout file pointer

- **FILE***

  A FILE*, called a "file pointer," is a pointer to a FILE structure that allows programs to read and write to files. FILE* is available via `#include <stdio.h>`.

- **stdout**

- **format string**

- **format specifiers**

9.2 The stdin file pointer

-

- **fscanf()**

- **format string**

- **format specifier**

- **stdin**

9.3 Output formatting

- **output formatting**

- **format sub-specifiers**

- **fflush()**

9.4 Input parsing

- **scanf**
- **fscanf**
- **input parsing**

  A programmer can control the way that input is read when using scanf(), a task known as input parsing.

- **format sub-specifiers**
- **sscanf()**
- **format string**
- **format specifier**

9.5 File input and output

- **fopen()**
- **read mode**
- **feof()**
- **write mode**

10. Recursion

10.1 Recursion: Introduction

- **algorithm**

  An algorithm is a sequence of steps for solving a problem.

- **recursive algorithm**

A recursive algorithm solves a problem by breaking that problem into smaller subproblems, solving these subproblems, and combining the solutions.

- **recursive**

  An algorithm that is defined by repeated applications of the same algorithm on smaller problems is a recursive algorithm.

- **base case**

  At some point, a recursive algorithm must describe how to actually do something, known as the base case.

10.2 Recursive functions

- **recursive function**

  A function that calls itself is a recursive function.

10.3 Recursive algorithm: Search

- **binary search**

  A binary search algorithm begins at the midpoint of the range and halves the range after each guess.

- **base case**

10.4 Adding output statements for debugging

- **conditional compilation**

10.5 Creating a recursive function

- **base case**

10.6 Recursive math functions

- **Fibonacci sequence**

  The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, etc.; starting with 0, 1, the pattern is to compute the next number by adding the previous two numbers.

- **greatest common divisor**

  The greatest common divisor (GCD) is the largest number that divides evenly into two numbers.

10.7 Recursive exploration of all possibilities

No terms in this section.

10.8 Stack overflow

- **stack frame**

  Each function call places a new stack frame on the stack, for local parameters, local variables, and more function items.

- **stack overflow**

  Deep recursion could fill the stack region and cause a stack overflow, meaning a stack frame extends beyond the memory region allocated for stack, .

10.9 C example: Recursively output permutations

No terms in this section.

11. Additional Material

11.1 Do-while loops

- **do-while loop**

  A do-while loop is a loop construct that first executes the loop body's statements, then checks the loop condition.

11.2 Engineering examples

No terms in this section.

11.3 Command-line arguments

- **Command-line arguments**

  Command-line arguments are values entered by a user when running a program from a command line.

- **argc**

  When a program is run, the system passes an int parameter argc to main(), indicating the number of command-line arguments (argc is short for argument count).

- **argv**

  When a program is run, the system passes a second parameter argv to main() (argv is short for argument vector), defined as an array of strings: char* argv[].

- **usage message**

  A usage message lists a program's expected command-line arguments.

11.4 The #define directive

- **#define**

The #define directive, of the form #define MACROIDENTIFIER replacement, instructs the processor to replace any occurrence of MACROIDENTIFIER in the subsequent program code by the replacement text.

- **macro**

  #define is sometimes called a macro.

- **#undef**

- **#ifdef**

- **#if**

- **#else**

- **#elif**

- **#pragma**
- **#line**
- **#error**

11.5 Modular compilation

- **single-step compilation**
- **modular compilation**
- **object file**

  An object file contains machine instructions for the compiled code along with placeholders, often referred to as references, for calls to functions or accesses to variables or classes defined in other source files or libraries.

-

- **linker**

- **linking**

11.6 Makefiles

- **project management tools**

- **make**

  Make is one project management tool that is commonly used on Unix and Linux computer systems.

- **makefile**

  The make utility uses a makefile to recompile and link a program whenever changes are made to the source or header files.

- **Make rules**

  Make rules are used to specify dependencies between a target file (e.g., object files and executable) and a set of prerequisite files that are needed to generate the target file (e.g., source and header files).

- **make recipe**

11.7 Binary file I/O

- **binary file mode**

11.8 Engineering examples using functions

No terms in this section.

11.9 Command-line arguments and files

No terms in this section.

11.10 Additional practice: Output art

- **ASCII art**

11.11 Additional practice: Grade calculation

No terms in this section.

11.12 Additional practice: Health data

No terms in this section.

11.13 Additional practice: Tweet decoder

No terms in this section.

11.14 Additional practice: Dice statistics

No terms in this section.

12. New Content

12.1 Variable name scope

- **scope**

  A declared name is only valid within a region of code known as the name's scope.

- **block**

  A block is a brace-enclosed {...} sequence of statements, such as found with an if-else, for loop, or while loop.

12.2 zyBooks built-in programming window

No terms in this section.