Essentials of C Programming + Problem Solving
Brief History of C

- UNIX developed c. 1969 -- DEC PDP-7 Assembly Language
- A totally new language “C” a successor to “B”. c. 1971
- By 1973 UNIX OS almost totally written in “C”. 
Reference in C

B.W. Kernighan  D.M. Ritchie
Writing a C Program
# General Format

```c
#include <libraries.h>

main()
{
  declarations
  statements
}
```
A First Program

#include <stdio.h>

main() {
    printf("It's a delight to be here!\n");
}

The Use of printf() and scanf()

Format:
control string, other arguments

The control string is simply a string that may contain conversion specifications (or formats).
main() {
    char c;
    float x, y;

    c = 'A';
    printf("%c\n", c); /* the letter A is printed */

    x = 1.0;
    y = 2.0;

    /* The sum of x and y is 3.000000 */
    printf("The sum of x and y is %f.\n", x + y);
}

Examples
## `Printf()`

<table>
<thead>
<tr>
<th>Conversion character</th>
<th>How the corresponding argument is printed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>as a character</td>
</tr>
<tr>
<td>d</td>
<td>as a decimal integer</td>
</tr>
<tr>
<td>e</td>
<td>as a floating point number in scientific notation</td>
</tr>
<tr>
<td>f</td>
<td>as a floating point number</td>
</tr>
<tr>
<td>g</td>
<td>in the e-format or f-format, whichever is shorter</td>
</tr>
<tr>
<td>s</td>
<td>as a string</td>
</tr>
</tbody>
</table>
# Scanf()

<table>
<thead>
<tr>
<th>Conversion character</th>
<th>What characters in the input stream are converted to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c</code></td>
<td>to a character</td>
</tr>
<tr>
<td><code>d</code></td>
<td>to a decimal integer</td>
</tr>
<tr>
<td><code>f</code></td>
<td>to a floating point number ((float))</td>
</tr>
<tr>
<td><code>lf</code></td>
<td>to a floating point number ((long\ float\ or\ double))</td>
</tr>
<tr>
<td><code>S</code></td>
<td>to a string</td>
</tr>
</tbody>
</table>
Example

Refer to program age.cpp
Precision in Printf

- Precision is the number of decimal digits printed to the right of the decimal point.
- For floating values, precision can be controlled.
- Format: \%m.nf
  - where m is the field length
  - and n is the precision
- Refer to program add.cpp
Types, Operators and Expressions
Variable Names

Every variable name in C must start with a letter, the rest of the name can consist of letters, numbers and underscore characters.

C recognizes upper and lower case characters as being different.

It is conventional to avoid the use of capital letters in variable names (so that capital letters can be used for names of constants).

You cannot use any of C's keywords.
Types of Variables

C provides a wide range of data types

- **int** - an integer
- **float** - a single precision floating point (real) number
- **char** - a single byte of memory (enough to hold a character)
- **short** - an integer w/ (possibly) reduced range
- **long** - an integer w/ (possibly) increased range
- **unsigned** - an integer w/ no negative values
- **unsigned long**
- **double** - a double precision floating point number
Constants

- **Numerical Constant** (e.g. 1, -3, 2.5, 1.0e7)
- **Character Constant** (e.g. ‘a’, ‘b’, ‘c’)
  - ‘\n’ - newline
  - ‘\t’ - tab
  - ‘\\’ - backslash
  - ‘’' - single quote
  - ‘\0’ - null (used automatically to terminate strings)
- **String Constant** (e.g. “Hello World”)
- ‘a’ ≠ “a”
- Refer to program ascii.cpp
## Escape Sequences

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\a</code></td>
<td>alert (bell)</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>backspace</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>formfeed</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>newline</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>carriage return</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>horizontal tab</td>
</tr>
<tr>
<td><code>\v</code></td>
<td>vertical tab</td>
</tr>
<tr>
<td><code>\\</code></td>
<td>backslash</td>
</tr>
<tr>
<td><code>\?'</code></td>
<td>question mark</td>
</tr>
<tr>
<td><code>\'</code></td>
<td>single quote</td>
</tr>
<tr>
<td><code>\&quot;</code></td>
<td>double quote</td>
</tr>
<tr>
<td><code>\000</code></td>
<td>octal number</td>
</tr>
<tr>
<td><code>\xhh</code></td>
<td>hexadecimal number</td>
</tr>
</tbody>
</table>
The Use of `#define` (for constants)

- The C compiler has a preprocessor built into it.
- Just before compilation, the source code is modified.
- Example:

  ```c
  #define LIMIT 100
  #define PI 3.14
  ...
  printf("Limit is %d\n", LIMIT);
  ```
Expressions and Operators

- Assignment statement (e.g., \( y = ( m \times x ) + b \))

- Arithmetic Operators
  - + (Addition)
  - - (Subtraction)
  - * (Multiplication)
  - / (Division*)
  - \% (Modulo Reduction)

- Remember the precedence of operators
Shorthand Operators/Assignment Statements

++, --

\[ x = (a++) * (--b); \]
/* This is equivalent to
   b = b - 1;
   x = a * b;
   a = a + 1;    */

+=, -=, *=, /=

\[ x -= 3;    /* x = x - 3 */\]

What is the difference of n++ and ++n?
Relational, Equality and Logical Operators

Relational, Equality and Logical Operators yield either the \texttt{int} value 0 (false) or the \texttt{int} value 1 (true).

- Relational Operators: $<$, $>$, $\leq$, $\geq$
- Equality Operators: $==$, $!=$
- Logical Operators: $!$, $\&\&$, $||$
Control Statements
The Compound Statement

A compound statement is a series of statements surrounded by the braces { and }.

```c
{
    ++i;
    sum += x;
}
```
The Compound Statement

- A compound statement is a series of statements surrounded by the braces { and }.
  ```
  {
    ++i;
    sum += x;
  }
  ```

- The chief use is to group statements into an executable unit.

- When declarations come at the beginning of a compound statement, it is called a block.

- Wherever it is possible to place a statement, it is also possible to place a compound statement.
The Null Statement

- The null statement is written as a single semicolon.
- It causes no action to take place.

```
a = b + c;
;         /* this is a null statement */
```
The if else Statement

This is used to decide whether to do something at a special point, or to decide between two courses of action.

```c
if (result >= 60) {
    printf("Pass\n");
    printf("ok\n");
} else
    printf("Fail\n");
```

It is possible to use the `if` part without the else or use `else if` for the else part.

Refer to find_min program.
The Switch Statement

switch (expression) {
    case const-expr: statements
    case const-expr: statements
    default: statements
}


The while Statement

Used to break the sequential flow of statements.

While (expression)

statement

*Refer to find_sum program.
The for Statement

for (expression1; expression2; expression3)
statement;
The do-while Statement

do {
    statement;
} while(expression);
Functions and Program Structure
Structured programming is a problem solving strategy and a programming methodology that strives for simple flow of control and uses top-down design.
Main Points

Top-down design, also referred to as stepwise refinement, consists of repeatedly decomposing a problem into smaller problems.

**MAIN TASK** :

- Subtask1
- Subtask2
  - Subtask 2_a
  - Subtask 2_b
  - ...
- Subtask3
  - ...

Main Points

- A long program should be written as a collection of functions, each one being no longer than, say, a page in length.
- Each function should capture some small task of the overall problem.
Main Points

- A programmer creates a function by writing a function definition. A function definition consists of a header and a body.

- The header specifies:
  - name of the function
  - type of the return value
  - types of the parameters to be passed to the function.

- The body of the function definition consists of declarations and statements.
Main Points

- Turbo C supports
  - Newer function syntax (called function prototype)
    include <stdio.h>

    void prn_message(void);

    main() {
      prn_message();
    }
    void prn_message(void) {
      printf("Have a nice day\n");
    }

- Functions can call on other functions.
Main Points

- The execution of the program always begins with `main()`. When a function is called, program control is passed to the function.
- When a `return` statement is executed, or the end of the function is reached, control is passed back to the calling environment.
- If a `return` statement contains an expression, the value of the expression is passed back to the calling environment as well.
Sample Top-Down Design

Sample Task: Read a series of input numbers and printout the current sum, minimum and maximum.

Refer to program rd_n_prn.cpp.
Another Example

- The Game of Heads or Tails.
- Refer to program headtail.cpp.
Call by Value

Functions are invoked by writing their name and an appropriate list of arguments within parentheses.

All arguments are passed “call by value.”

Only a copy of the value of the argument is passed.
Call by Value: an Example

```c
main()
{
    int k=3, sum;
    int compute_sum(int);

    printf("%d\n", k);       /* 3 is printed */
    sum = compute_sum(n);
    printf("%d\n", k);       /* 3 is printed */
    printf("%d\n", sum);     /* 6 is printed */
}

int compute_sum(int k)  /* sum the integers from 1 to k */
{
    int sum = 0;

    for (; k > 0; --k)
        sum += k;
    printf("%d\n", k);       /* 0 is printed */
    return (sum);
}
```
Character Processing
Main Points

Basic input/output for characters is accomplished readily with the standard macros `getchar()` and `putchar()`
Main Points

- When doing character input, it is frequently necessary to test for the end-of-file mark. This is accomplished by using the symbolic constant `EOF` in the program.
- The symbolic constant `EOF` is defined in the system header `stdio.h`.
- In most systems, including Turbo C, the value of `EOF` is `-1`.
- In Turbo C running under DOS, an `EOF` can be entered by typing `Ctrl-Z` followed by a carriage return.
Examples

Refer to the following programs:

- copy.cpp
- countc.cpp
- countl.cpp
- countw.cpp
- lower.cpp
Exercises

- Write a program to count blanks, tabs, and newlines.
- Write a program to copy its input to its output, replacing each string of one or more blanks by a single blank.
- Write a program to copy its input to its output, replacing each tab by \t, each backspace by \b, and each backslash by \\.
  This makes tabs and backspaces visible in an unambiguous way.
Exercises

Write a program containing the function `upper` which converts all input to uppercase letters.
## The Macros in ctype.h

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha(c)</td>
<td>c is a letter</td>
</tr>
<tr>
<td>isupper(c)</td>
<td>c is an uppercase letter</td>
</tr>
<tr>
<td>islower(c)</td>
<td>c is a lowercase letter</td>
</tr>
<tr>
<td>isdigit(c)</td>
<td>c is a digit</td>
</tr>
<tr>
<td>isxdigit(c)</td>
<td>c is a hexadecimal digit</td>
</tr>
<tr>
<td>isspace(c)</td>
<td>c is a white space character</td>
</tr>
<tr>
<td>isalnum(c)</td>
<td>c is a letter or digit</td>
</tr>
<tr>
<td>ispunct(c)</td>
<td>c is a punctuation character</td>
</tr>
<tr>
<td>isprint(c)</td>
<td>c is a printable character</td>
</tr>
<tr>
<td>isgraph(c)</td>
<td>c is printable, but not a space</td>
</tr>
<tr>
<td>iscntrl(c)</td>
<td>c is a control character</td>
</tr>
<tr>
<td>isascii(c)</td>
<td>c is an ASCII code</td>
</tr>
</tbody>
</table>
## Functions in `ctype.h`

<table>
<thead>
<tr>
<th>Function</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toupper(c)</code></td>
<td>changes c from lowercase to uppercase</td>
</tr>
<tr>
<td><code>tolower(c)</code></td>
<td>changes c from uppercase to lowercase</td>
</tr>
<tr>
<td><code>toascii(c)</code></td>
<td>changes c to ASCII code</td>
</tr>
</tbody>
</table>
Arrays
Main Points

- Arrays can be used to deal with a large number of homogeneous values.
- A declaration such as
  ```c
  int   a[100];
  ```
  makes an array of int’s. The compiler allocates contiguous space memory for 100 int’s and numbers the elements of a from 0 to 99.
Initializing Arrays

float x[4] = {-1.1, 0.2, 33.0, 4.4};

int a[4] = {1, 2, 3},
b[] = {3, 4, 5, 6};
Example

Refer to program abc_cnt.cpp
Exercises

Write a program containing the function `squeeze`:

```c
void squeeze(char s[], int c)
```

which removes all the occurrences of the character `c` from the string `s`. 
Pointers and Arrays
Main Points

- Pointers are used in programs to access memory and manipulate addresses.
- The declaration
  ```c
  int  *p;
  ```
  declares \( p \) to be of type “pointer to \textit{int}”.
- A pointer variable typically takes as values either \texttt{NULL} or addresses of other variables.
- The legal range of values for any pointer includes the special address 0 (for \texttt{NULL}) and a set of positive integers that are interpreted as machine addresses.
Main Points

- The unary operator & gives the address of an object

- Some examples of assignment to pointers:
  
  ```
  p = &i;
p = 0;
p = NULL;  /* same as p = 0 */
p = (int *) 1507;  /* absolute address */
  ```
Example

```c
int i, *p;

p = &i;

scanf("%d", p) /*same as scanf("%d", &i);*/
```
The unary operator *

- Indirection or dereferencing operator
- Accesses the object the pointer points to

Example:

```c
int x = 1, y=2, z[10];
int *ip;  /* ip is a pointer to int */

ip = &x;  /* ip now points to x */
y = *ip;   /* y is now 1 */
*ip = 0;   /* x is now 0 */
ip = &z[0]; /* ip now points to z[0] */
```
Example

*ip = ip + 10;    /* increments ip by 10 */
y = *ip +1;       /* adds 1 to whatever ip
                 points to and assigns the result to y*/
*ip += 1;        /* increments by 1 what ip
+++*ip;          points to */
(*ip)++

iq = ip;         /* assuming iq is pointer to
int; copies the contents of ip into iq
making iq point to whatever ip pointed to */
void swap (int x, int y) { /* WRONG */
    int temp;

    temp = x;
    x = y;
    y = temp;
} /* only swaps copies of a and b */
Pointers and Function Arguments

Solution: calling program to pass pointers to the values to be changed:

```c
swap(&a, &b);
```

/* interchange *px and *py */

```c
void swap (int *px, int *py) {
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```
Arrays and Pointers

- In C, there is a strong relationship between arrays and pointers.
- Any operation that can be achieved by array subscripting can also be done with pointers.
Example

```c
int a[10];  /* defines an array a of size 10 */
int *pa;    /* pa is a pointer to int*/

pa = &a[0]; /* pa points to element 0 of a */
x = *pa;    /* copy the contents of a[0] into x */
*(pa+1)     /* refers to the content of a[1] */

pa = a;     /* equivalent to pa = &a[0] */
```
Note:

- $a[i]$ can be written as $*(a+i)$
- $&a[i]$ is identical to $a+i$
- $p_a[i]$ is identical to $*(p_a+i)$
- A pointer is a variable: $p_a = a$; $p_a++$ are legal
- An array name is not a variable: $a = p_a$; $a++$ are illegal
Main Points

When an array name is passed as an argument to a function, only a copy of the base address of the array is actually passed.

```c
int sum(int a[], int n) { /* n is array size */
    int i, s = 0;
    for (i = 0; i < n; ++i)
        s += a[i];
    return (s);
}
```
Main Points

- In the header of a function definition,
  \texttt{int a[]} \textit{is equivalent to} \texttt{int *a}

- As declarations within the body of a function, they are not equivalent. The first will create a constant pointer (and no storage), whereas the second will create a pointer variable.
Exercises

Write a function that orders the stored values of three characters. Suppose, for example, that \(c_1\), \(c_2\), and \(c_3\) are character variables having the values ‘C’, ‘B’ and ‘D’, respectively. Then the function call `order_chars(&c1, &c2, &c3)` should cause the stored values of \(c_1\), \(c_2\) and \(c_3\) to be ‘B’, ‘C’ and ‘D’ respectively. Write a program that thoroughly tests your function.
Exercises

Write a program that reads n integers into an array, and then prints the value of each distinct element along with the number of times that it occurs on a separate line. The values should be printed in descending order. Suppose for example, that you input the values -7  3  3  -7  5  5  3 as the elements of your array. Then your program should print:

5 occurs 2 times
3 occurs 3 times
-7 occurs 2 times
Dynamic Memory Allocations

- `malloc (object_size)`
- `calloc (n, object_size)`

```c
printf("\nEnter size of array: ");
scanf("%d", &n);
a = (int *) calloc(n, sizeof(int));
...```

Strings and Pointers
Main Points

- Strings are one-dimensional arrays of type `char`.
- The null character `\0` is used to delimit a string.
- Systems functions such as `printf()` will work properly only on null terminated strings.
Main Points

A function call such as `scanf("%s", w)` can be used to read a sequence of nonwhite space characters into the string `w`.

After all the characters have been read in, `scanf()` automatically ends the string with the null character.
Main Points

Strings may be initialized.

```c
char s[] = "abc";

/* equivalent to
   char s[] = {'a', 'b', 'c', '\0'};
*/

char *p = "abc";
```
Main Points

- Refer to program nice_day.
Main Points

- **String processing can be done by making use of array notation with subscripts and by making use of pointers and pointer arithmetic.**
- **Refer to program change.**
Main Points

- The standard library contains many useful string handling functions.
  - `strcpy(s1, s2)` - copy `s2` to `s1`, return `s1`
  - `strcmp(s1, s2)` - compare strings `s1` and `s2`.
Exercises

Write the function `strend(s,t)`, which returns 1 if the string `t` occurs at the end of the string `s`, and zero otherwise.
Pointer Arrays;
Pointers to Pointers
Main Points

- Pointers can be stored in arrays
- Example
  
  ```
  char *s[100]; /* an array of pointers to a string */
  ```
Pointers vs. Multi-dimensional Arrays

Given: int a[10][20];
    int *b[10];

* a[3][4] and b[3][4] are both legal references to a single int
* a is a true two-dimensional array; 200 int-sized locations have been set aside
* b only allocates 10 pointers but does not initialize them; initialization must be done explicitly
char *name[] = { "Illegal month", "Jan", "Feb", "Mar" };