C Programming Essentials

Unit 4: System Programming

CHAPTER 12: C STANDARD LIBRARY

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- In this Chapter, you'll learn about the standard library functions in C. More specifically, what are they, different library functions in C and how to use them in your program.

What is C Standard Library

- C Standard library functions or simply C Library functions are inbuilt functions in C programming.
- The prototype and data definitions of these functions are present in their respective header files. To use these functions we need to include the header file in our program.

Advantages of Using C library functions

1. They work

One of the most important reasons you should use library functions is simply because they work. These functions have gone through multiple rigorous testing and are easy to use.

2. The functions are optimized for performance

Since, the functions are "standard library" functions, a dedicated group of developers constantly make them better. In the process, they are able to create the most efficient code optimized for maximum performance.

Advantages of Using C library functions

3. It saves considerable development time

Since the general functions like printing to a screen, calculating the square root, and many more are already written. You shouldn't worry about creating them once again.

4. The functions are portable

With ever-changing real-world needs, your application is expected to work every time, everywhere. And, these library functions help you in that they do the same thing on every computer.

LECTURE 1

Introduction

C Standard Library

- The **C standard library** (aka **libc**) is a standardized collection of **header** files and **library** routines, which are used to implement common operations, such as input/output and string handling etc.
- C does not have built in keywords for these tasks, so nearly all C programs rely on the standard library.

libc

<assert.h>
<ctype.h>
<errno.h>
<sys/file.h>
<float.h>
<limits.h>
<locale.h>
<math.h>
<setjmp.h>
<signal.h>
<stdarg.h>
<stddef.h>
<stdio.h>
<stdlib.h>
<string.h>
<time.h>

libc

<**assert.h**> : Contains the assert macro, helpful in detecting logical errors and other types of bug in debugging versions

<**ctype.h**> : to classify characters by their types or to convert between upper and lower case

<**errno.h**> : For testing error codes

<**float.h**> : Contains macros that expand to various limits and parameters of the standard floating-point types

libc

<**limits.h**> : constants specifying the implementation-specific properties of the integer types

<**locale.h**> : to set and select locale

<**math.h**> : common math functions

<**setjmp.h**> : macros setjmp and longjmp

<**signal.h**> : various exceptional conditions

<**stdarg.h**> : to allows functions to accept an variable number of arguments

libc

<**stddef.h**> : some useful types and macros

<**stdio.h**> : input/output functionaliteis

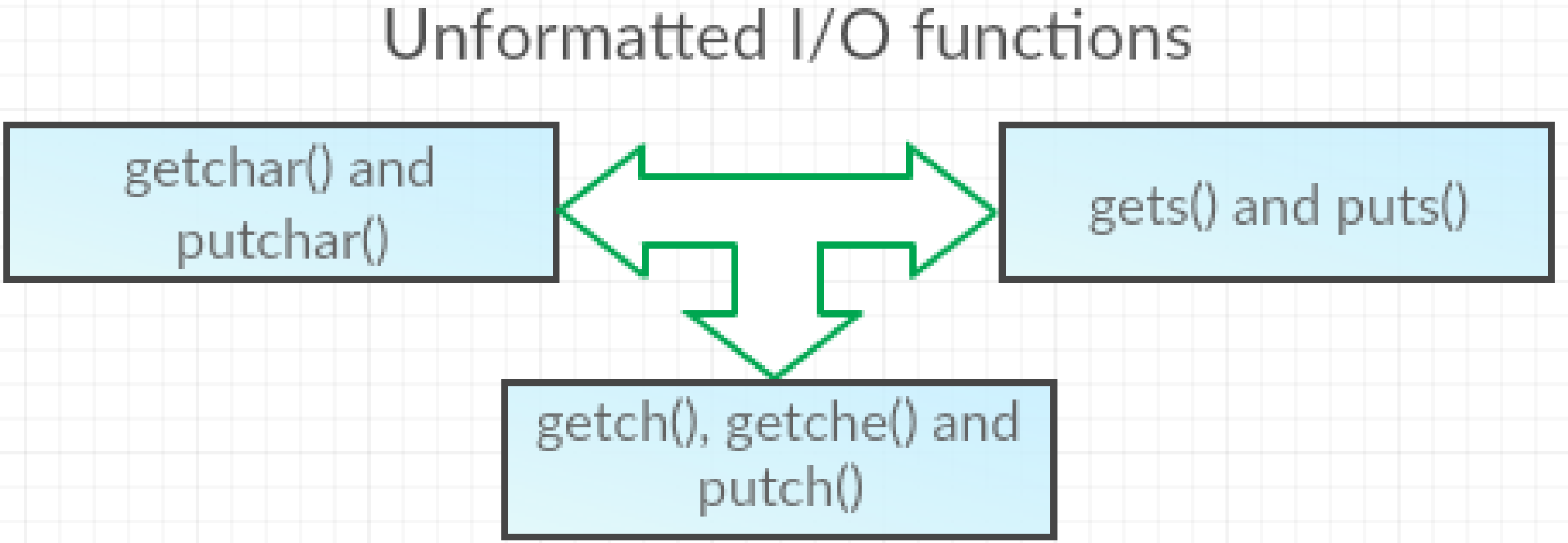
<**stdlib.h**>: conversion, pseudo-random numbers, memory allocation, process control, environment, signalling, searching, and sorting.

<**string.h**> : string manipulation and memory handling

<**time.h**> : time/date formats and manipulation

LECTURE 1

Basic I/O Operations <stdio.h>



Unformatted I/O Functions

Output

putchar:

putchar will display a single character at a time on the screen. To print many characters, then use loop statements.

Example: putchar('a');

puts: (println in Java)

puts() will display the string followed by a new line(\n).

Example: puts(splessons)

Demo Program: putchar_eg.c

```
Enter a character: b
b
```

```c
#include <stdio.h>
#include <conio.h>
void main( ){
   int a;
   printf("Enter a character: ");
   a=getchar();
   putchar(a);
   getch(); // wait for one keyboard stroke
}
```

Input

getchar():

Input can also be given using the `gets()` and `getchar()` functions. `getchar()` function reads characters one by one until the `return()` statement is approached. To read all the characters at a time then use loops. It reads the data and stores in a variable.

Example:

```
int ch;
ch=getchar();
```

Input

gets():

gets() function reads entire line given by the user through keyboard from stdin file and assigns a null character at the end.

Example:

```
gets(name);
```

Demo Program gets_eg.c

```
#include<stdio.h>
void main() {
    char sp[20];
    printf("Enter a string: ");
    gets(sp); // input.nextLine()
    puts(sp); // System.out.println(sp);
    getch();
}
```

```
Enter a string: How are you?
How are you?
```

Input without Print

getch():

getch function scans only one character at a time from the keyboard. The output is not projected on the screen.

Example:

```
char c;
c=getch();
```

Input with Print

getche():

getche function scans only one character at a time from the keyboard. The output is projected to the screen.

Example:

```
char c;
c=getche();
```

Output a character

putch():

putch function displays only one character at a time to the screen.

Example:

```
char c;
c=putch();
```

Formatted Functions		
Type	**Input**	**Output**
char	scanf()	printf()
int	scanf()	printf()
float	scanf()	printf()
string	scanf()	printf()

Unformatted Functions		
Type	**Input**	**Output**
char	getch() getche() getchar()	putch() putchar()
int	-	-
float	-	-
string	gets()	puts()

stdio.h

- Formatted standard I/O
  - printf - formatted output conversion
  - scanf - input format conversion
- Formatted Conversion to strings
  - sprintf - formatted output conversion

- File I/O
  - fclose - close a stream
  - feof - check and reset stream status
  - ferror - check and reset stream status
  - fgetc - input of characters and strings
  - fgets - input of characters and strings
  - fopen - stream open functions
  - fprintf - formatted output conversion
  - fputc - output of characters and strings
  - fread - binary stream input/output
  - fseek - reposition a stream
  - fwrite - binary stream input/output

printf

Format Specifer:

The format specifier follows the following prototype:

%[flags][width][.precision][length]specifier

specifier	argument
d or i	Singed INT
u	Unsinged INT
o	Unsigned Octal
x or X	Unsigned Hexadecimal (lower/upper case)
e or E	Floating Point
g or G	Shortest Representation
a or A	Hexadecimal Floating Point (lower/upper case)
c	Character
s	String of characters
p	Pointer address

flag	description
-	Left-justify (default: right)
+	Force print + sign with positives
(space)	Add space, if no sign before value
#	Precede o, x, or X with 0
0	Left pad number with zeros

.precision	description
.number	specifies number of digits to write
.*	not specified in cstring, additional INT value given

width	description
(number)	Minimum number of characters to print
*	Not specified in cstring, additional INT value given

```c
/* Example for printf() */
#include <stdio.h>
int main(){p
   printf ("Integers: %i %u \n", -3456, 3456);
   printf ("Characters: %c %c \n", 'z', 80);
   printf ("Decimals: %d %ld\n", 1997, 32000L);
   printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
   printf ("floats: %4.2f %+.0e %E \n", 3.14159, 3.14159, 3.14159);
   printf ("Preceding with empty spaces: %10d \n", 1997);
   printf ("Preceding with zeros: %010d \n", 1997);
   printf ("Width: %*d \n", 15, 140);
   printf ("%s \n", "Educative");
   return 0;
}
```

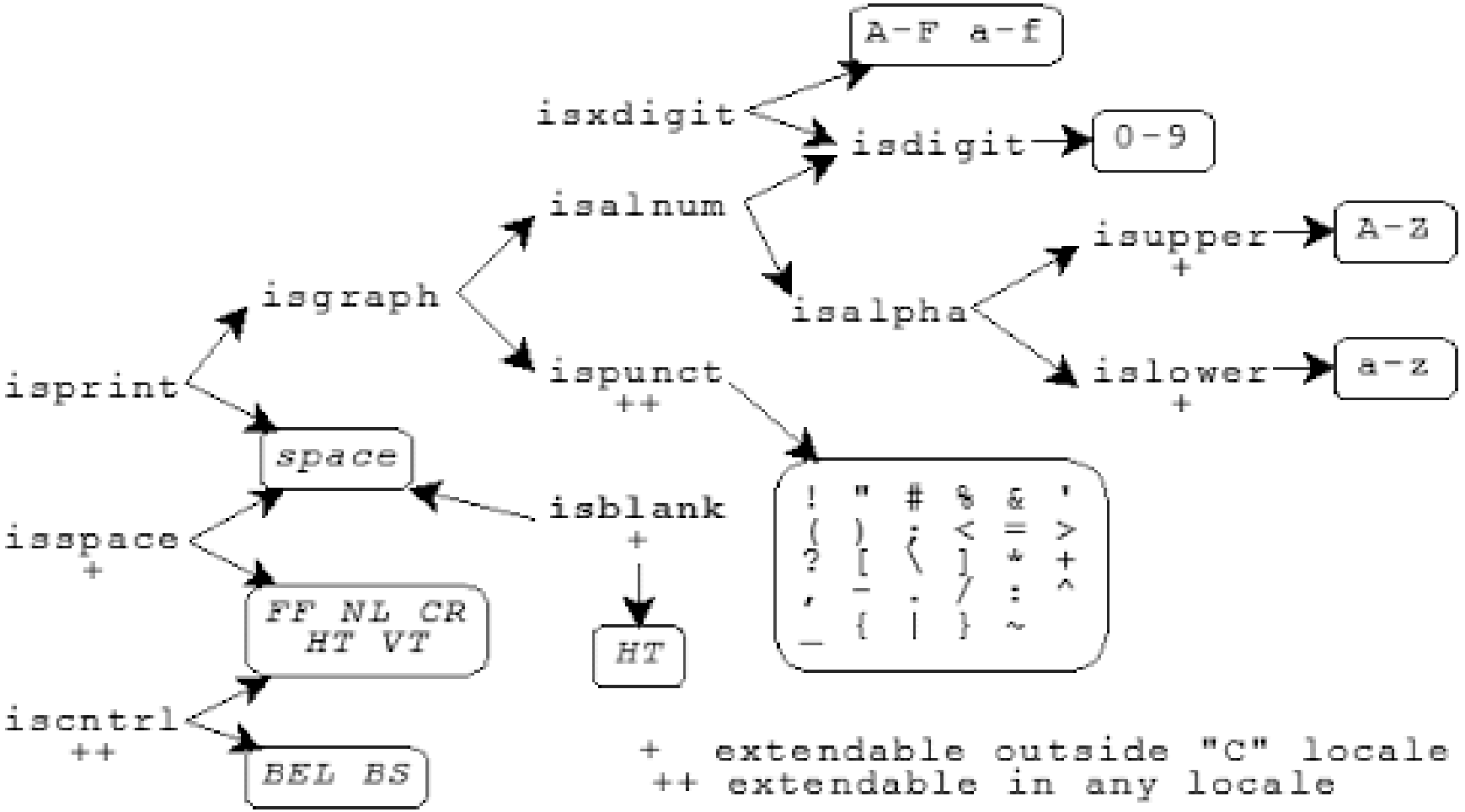
```
Integers: -3456 3456
Characters: z P
Decimals: 1997 32000
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+00 3.141590E+00
Preceding with empty spaces:       1997
Preceding with zeros: 0000001997
Width:             140
Educative
```

LECTURE 1

Logic Function <ctype.h>

ctype.h

- isalnum - character classification functions
- isalpha - character classification functions
- isdigit - character classification functions
- islower - character classification functions
- isspace - character classification functions
- isupper - character classification functions
- tolower - convert uppercase or lowercase
- toupper - convert uppercase or lowercase


A-F a-f
isxdigit
isdigit
0-9
isalnum
isupper
+
A-Z
isalpha
isgraph
islower
+
a-z
isprint
ispunct
++
space
isblank
+
isspace
+
FF NL CR
HT VT
HT
iscntrl
++
BEL BS
! " # % & '
( ) ; < = >
? [\ ] * +
, _ . / : ^
` { | } ~
+ extendable outside "C" locale
++ extendable in any locale

Character Functions

Character macros and functions	
Function or macro	Effect
toupper(c)	Changes c from lowercase to uppercase
tolower(c)	Changes c from uppercase to lowercase
toascii(c)	Changes c to ASCII code

LECTURE 1

Mathematical Libraries <math.h><limits.h> <random.h>

math.h

HUGE_VAL symbolic constant for a positive double expression

double **sin**(double x)

double **cos**(double x)

double **tan**(double x)

double **asin**(double x)

double **acos**(double x)

double **atan**(double x)
arc tangent of x in the range [-π/2,+π/2] radians

math.h

double **atan2**(double y, double x)
arc tangent of y/x in the range [-π,+π] radians

double **sinh**(double x)

double **cosh**(double x)

double **tanh**(double x)

double **exp**(double x)

double **log**(double x)

double **log10**(double x)

math.h

double **pow**(double x, double y)

double **sqrt**(double x)

double **ceil**(double x)

double **floor**(double x)

double **fabs**(double x)

double **ldexp**(double x, int n): $x.2^n$

math.h

double **frexp**(double x, int *exp)

- splits x into a normalized fraction in the interval [1/2, 1), which is returned, and a power of 2, which is stored in *exp. If x is zero, both parts of the result are zero.

double **modf**(double x, double *iptr)

- splits x into integral and fractional parts, each with the same sign as x. Integral part is stored in *iptr, and the fractional part is returned.

double **fmod**(double x, double y)

- floating-point remainder of x/y.

math.h

Domain Error (EDOM) :

- If an argument is outside the domain over which the function is defined

Range Error (ERANGE) :

- If the result cannot be represented as a double.
- If the result overflows, function returns HUGE_VAL with the right sign, and errno is set to ERANGE
- If the result underflows, function returns zero, whether errno is set to ERANGE is implementation defined

limits.h

- The `limits.h` header determines various properties of the various variable types. The macros defined in this header, limits the values of various variable types like char, int and long.
- These limits specify that a variable cannot store any value beyond these limits, for example an unsigned character can store up to a maximum value of `255`.

limits.h

Macro	Value	Description
CHAR_BIT	8	Defines the number of bits in a byte.
SCHAR_MIN	-128	Defines the minimum value for a signed char.
SCHAR_MAX	+127	Defines the maximum value for a signed char.
UCHAR_MAX	255	Defines the maximum value for an unsigned char.
CHAR_MIN	-128	Defines the minimum value for type char and its value will be equal to SCHAR_MIN if char represents negative values, otherwise zero.
CHAR_MAX	+127	Defines the value for type char and its value will be equal to SCHAR_MAX if char represents negative values, otherwise UCHAR_MAX.
MB_LEN_MAX	16	Defines the maximum number of bytes in a multi-byte character.
SHRT_MIN	-32768	Defines the minimum value for a short int.
SHRT_MAX	+32767	Defines the maximum value for a short int.
USHRT_MAX	65535	Defines the maximum value for an unsigned short int.
INT_MIN	-2147483648	Defines the minimum value for an int.
INT_MAX	+2147483647	Defines the maximum value for an int.
UINT_MAX	4294967295	Defines the maximum value for an unsigned int.
LONG_MIN	-9223372036854775808	Defines the minimum value for a long int.
LONG_MAX	+9223372036854775807	Defines the maximum value for a long int.
ULONG_MAX	18446744073709551615	Defines the maximum value for an unsigned long int.

```c
#include <stdio.h>
#include <limits.h>

int main() {
   printf("The number of bits in a byte %d\n", CHAR_BIT);
   printf("The minimum value of SIGNED CHAR = %d\n", SCHAR_MIN);
   printf("The maximum value of SIGNED CHAR = %d\n", SCHAR_MAX);
   printf("The maximum value of UNSIGNED CHAR = %d\n", UCHAR_MAX);
   printf("The minimum value of SHORT INT = %d\n", SHRT_MIN);
   printf("The maximum value of SHORT INT = %d\n", SHRT_MAX);
   printf("The minimum value of INT = %d\n", INT_MIN);
   printf("The maximum value of INT = %d\n", INT_MAX);
   printf("The minimum value of CHAR = %d\n", CHAR_MIN);
   printf("The maximum value of CHAR = %d\n", CHAR_MAX);
   printf("The minimum value of LONG = %ld\n", LONG_MIN);
   printf("The maximum value of LONG = %ld\n", LONG_MAX);
   return(0);
}
```

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch12\math>testlimits
The number of bits in a byte 8
The minimum value of SIGNED CHAR = -128
The maximum value of SIGNED CHAR = 127
The maximum value of UNSIGNED CHAR = 255
The minimum value of SHORT INT = -32768
The maximum value of SHORT INT = 32767
The minimum value of INT = -2147483648
The maximum value of INT = 2147483647
The minimum value of CHAR = -128
The maximum value of CHAR = 127
The minimum value of LONG = -2147483648
The maximum value of LONG = 2147483647
```

srand(), rand()

Description

- The C library function **int rand(void)** returns a pseudo-random number in the range of 0 to *RAND_MAX*.
- RAND_MAX is a constant whose default value may vary between implementations but it is granted to be at least 32767.

srand(), rand()

Description

- The C library function **int rand(void)** returns a pseudo-random number in the range of 0 to *RAND_MAX*.
- RAND_MAX is a constant whose default value may vary between implementations but it is granted to be at least 32767.

int rand(void)

int rand(void):

Return Value:

- This function returns an integer value between 0 and RAND_MAX.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main () {
   int i, n;
   time_t t;
   n = 5;

   /* Intializes random number generator */
   srand((unsigned) time(&t));
   /* Print 5 random numbers from 0 to 49 */
   for( i = 0 ; i < n ; i++ ) {
      printf("%d\n", rand() % 50);
   }
   return(0);
}
```

```
C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch12\math>test
20
41
45
41
20

C:\Eric_Chou\C Course\C Programming Essentials\CDev\Ch12\math>test
24
22
41
36
34
```

LECTURE 1

String and Memory Management <stdlib.h> <string.h>

<stdlib.h>

String Data Conversion:

- atof - convert a string to a double
- atoi - convert a string to an integer
- atol - convert a string to an integer
- atoll - convert a string to an integer

Memory Allocation:

- calloc - allocate and free dynamic memory
- free - allocate and free dynamic memory
- malloc - allocate and free dynamic memory
- random - random number generator
- realloc - allocate and free dynamic memory
- srandom - random number generator

Other built-in Typecase functions in C:

- Typecasting functions in C language performs data type conversion from one type to another.
- itoa() function converts int data type to string data type.
- Click on each function name below for description and example programs.

String Type Casting

Typecast function	Description
atof()	atof( ) function converts string to float
atoi()	atoi( ) function converts string to int
atol()	atol( ) function converts string to long
itoa()	itoa( ) function converts int to string
ltoa()	ltoa( ) function converts long to string

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    int a=54325;
    char buffer[20];
    itoa(a,buffer,2);   // here 2 means binary
    printf("Binary value = %s\n", buffer);

    itoa(a,buffer,10);   // here 10 means decimal
    printf("Decimal value = %s\n", buffer);
    itoa(a,buffer,16);   // here 16 means Hexadecimal
    printf("Hexadecimal value = %s\n", buffer);
    return 0;
}
```

```
Binary value = 1101010000110101
Decimal value = 54325
Hexadecimal value = d435
```

string.h (memory handling)

void* **memcpy** (void* dest, const void* src, size_t num)

void* **memmove** (void* dest, const void* src, size_t num) /*works even when the objects overlap*/

int **memcmp**(const void* buffer1, const void* buffer2, size_t num)

void* **memchr** (const void* buffer, int c, size_t num)

void* **memset** (void* buffer, int c, size_t num)

strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

string.h (string manipulation)

char* **strcpy** (char* *dest*, const char* *src*)
copy src to dest including '\0'

char* **strncpy** (char* *dest*, const char* *src*, sizet_t *num*)
pad with '\0's if src has fewer than num chars

char* **strcat** (char* *dest*, const char* *src*)

char* **strncat** (char* *dest*, const char* *src*, sizet_t *num*)

concatenate at most num chars, terminate dest with '\0' and return dest

string.h (string manipulation)

int **strcmp** (const char* *string1*, const char* *string2*)
returns <0 if string1<string2, 0 if string1==string2, or >0 if string1>string2

int **strncmp** (const char* *string1*, const char* *string2*, sizet_t *num*)

char* **strchr** (const char* *string*, int *c*)
return pointer to first occurrence of c in string or NULL if not present.

char* **strrchr** (const char* *string*, int *c*)
last occurrence of *c* in *string*

string.h (string manipulation)

`size_t` **`strspn`** `(const char* string1, const char* string2)`
return length of prefix of string1 consisting of chars in string2

`size_t` **`strcspn`** `(const char* string1, const char* string2)`
return length of prefix of string1 consisting of chars not in string2

`char*` **`strpbrk`** `(const char* string1, const char* string2)`
return pointer to first occurrence in string1 of any character of string2, or NULL if none is present.

string.h (string manipulation)

```
char* strstr (const char* string1, const char* string2)
```

return pointer to first occurrence of string2 in string1, or NULL if not present

```
size_t strlen (const char* string)
```

string.h (string manipulation)

```
char* strerror(int errnum)
```

Returns a pointer to a string with the error message corresponding to the *errnum* error number. Subsequent calls to this function will overwrite its content. This funcion can be called with the global variable, errno, declared in errno.h to get the last error produced by a call to a C library function.

string.h (string manipulation)

```
char *strtok(const char* string,const char* delimiters)
```

If *string* is not NULL, the function scans *string* for the first occurrence of any character included in *delimiters*. If a member of *delimiters* is found, the function overwrites the delimiter in *string* by a null-character and returns a pointer to the token, i.e. the part of the scanned string previous to the member of *delimiters*. After a first call to strtok, the function may be called with NULL as the string parameter, and it will continue from where the last call to strtok found a member of *delimiters*. Delimiters may vary from one call to another.