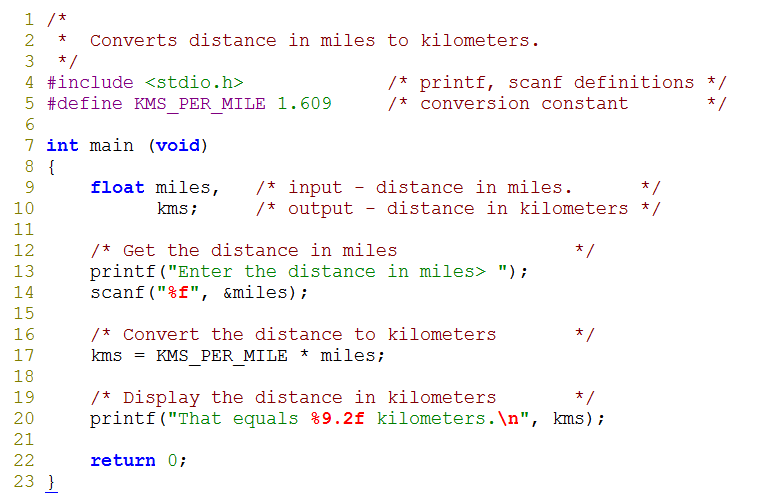
CS1010

Compiled by: Jin Zhe

NUS

Programming Methodology

This set of notes is compiled from Mr. Aaron Tan’s lecture slides for AY2011/12 S1. All credits goes to him.



**Preprocessor Directives:**

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

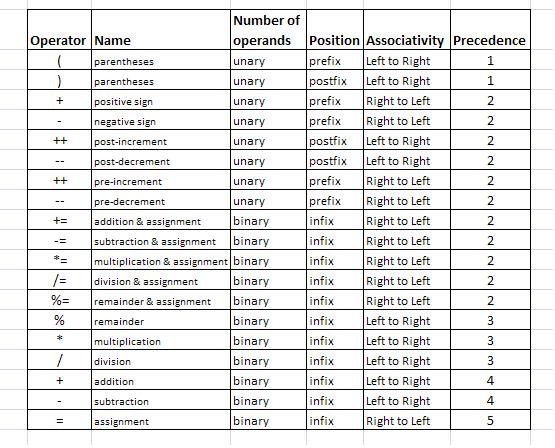
**Note:** x = 1.23456789

printf(“**%09.2lf**”, x); prints 000001.23 printf(“**%04.6lf**”, x); prints 1.234568 printf(“**%05lf**”, x); prints 1.234568 printf(“**%09.21lf**”, x); prints 1.234567799999999992977

Week2 - 43

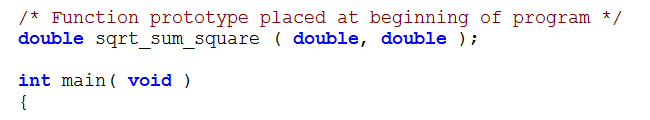
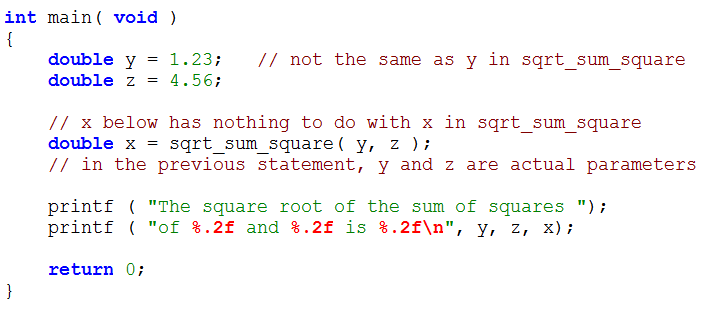
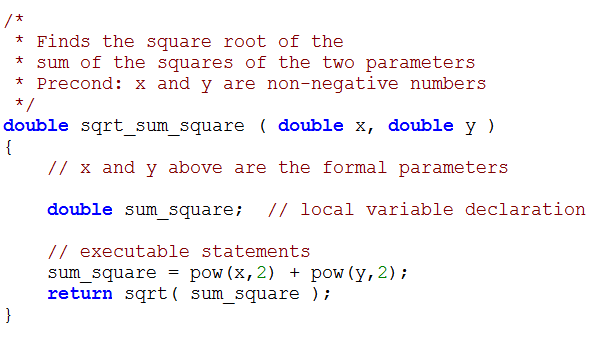
|  |  |  |  |
| --- | --- | --- | --- |
| **Placeholder** | **Variable Type** | **Function Use** | **Comments** |
| %c | char | printf / scanf | individual character, which is a letter, a digit, or a special symbol  'A' 'z' '2' '9' '\*' '?' ' ' etc. |
| %d | int | printf / scanf | 32 bits; value between -2,147,483,648 (-231) to +2,147,483,647 (231 – 1) |
| %f | float or double | printf |  |
| %f | float | scanf | 1.175494351 E – 38  to 3.402823466 E + 38 |
| %lf | double | scanf | 2.2250738585072014 E – 308 to 1.7976931348623158 E + 308 |
| %e | float or double | printf (for scientific notation) |  |

**Arithmetic operators: Associativity & Precedence**

****

**Note:** Priority goes to highest precedence first then down the list in order.

**Functions**

**Function Syntax:**

*Function interface comment*

***ftype*** *fname (****formal parameter declaration list****)*

**{**

*local* ***variable*** *declarations*

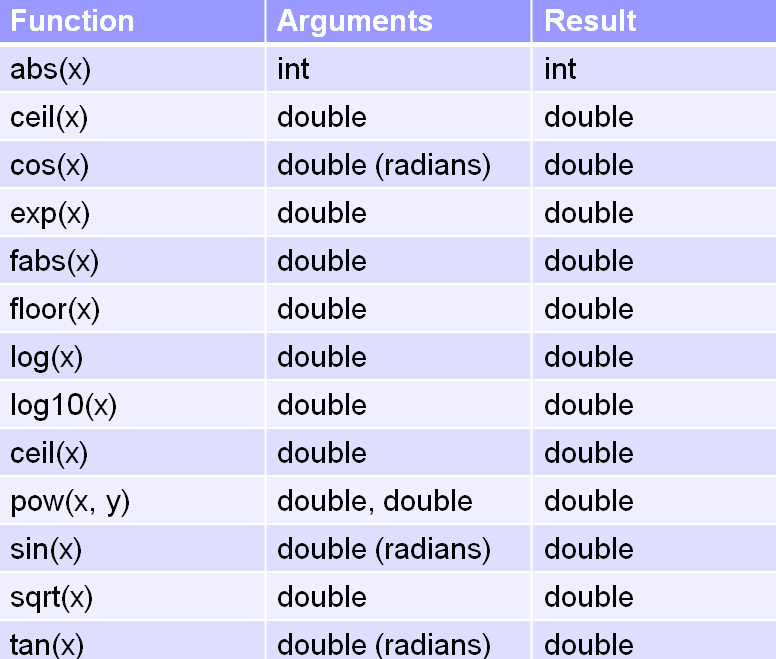
*executable statements*

*// include* ***return*** *statements, if any*

**}**

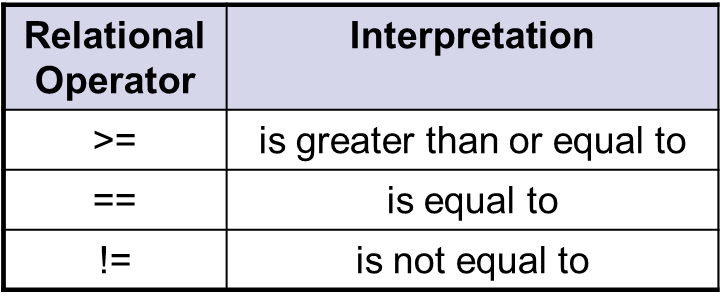
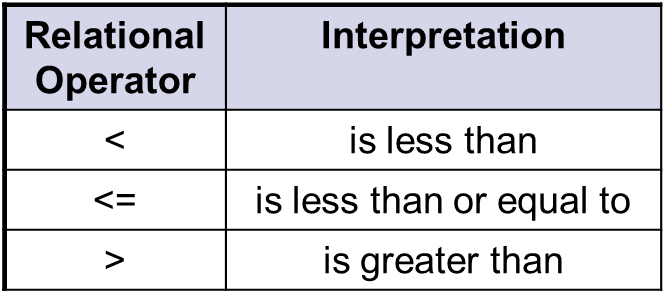
**Math library functions:**

**Note:** gcc compilations with math.h must include **“–lm**”



**Operators:**

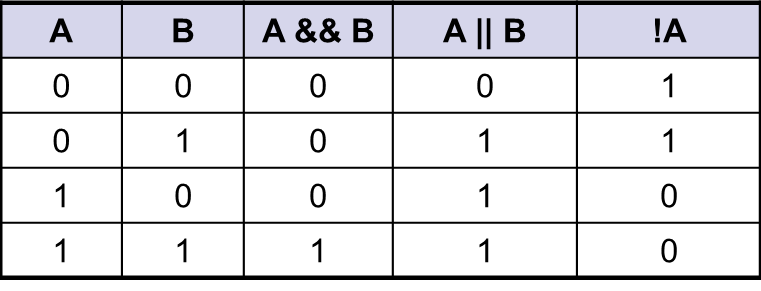
*Relational operator*

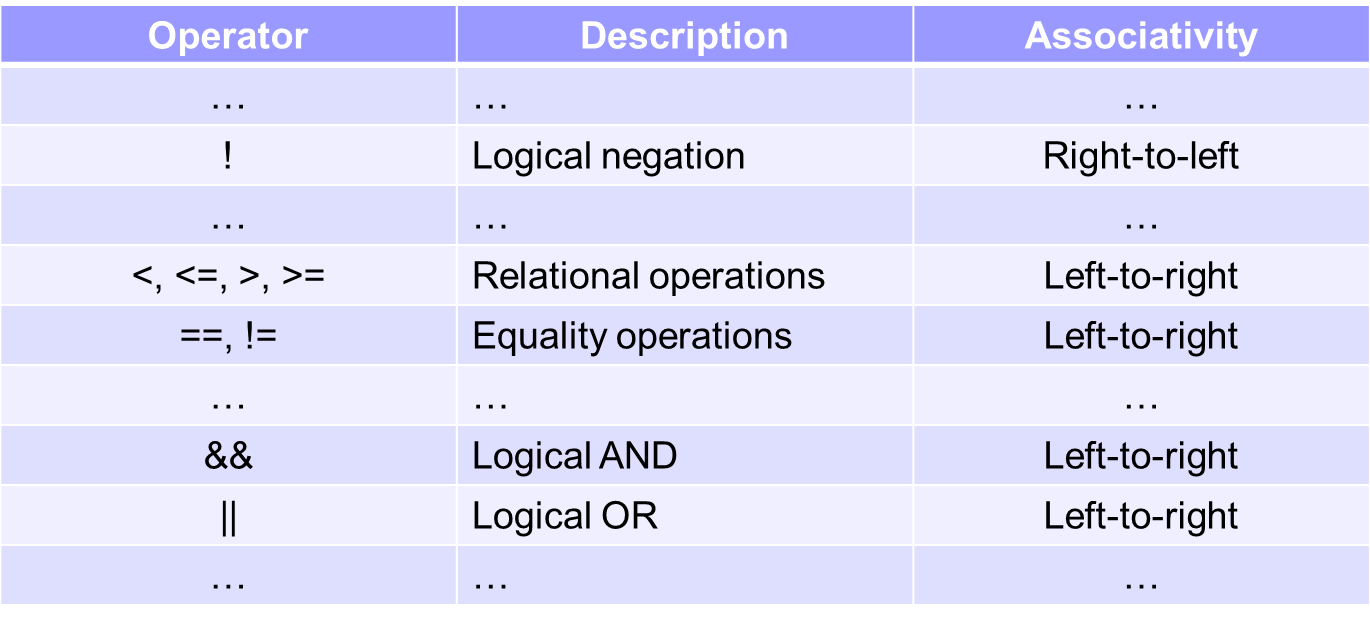
 

*Logical operator*

**Wrong**: a\*b == (1 || 2 || 3)

**Correct:** (a\*b == 1) || (a\*b == 2) || (a\*b == 3)





**Selection statements:**

***if*** statement

**if** **(** **condition is true ) {**

**/\* Execute statements inside the body \*/**

**}**

***if-else*** statements

**if (** **condition is true ) {**

**/\* Execute these statements if TRUE \*/**

**}**

**else {**

**/\* execute these statements if FALSE \*/**

**}**

***Switch*** statement

**Expression:** a + b, 3\*a-1, etc.; A variable is also an expression.

**Note:** Value for case restricted to *discrete* types only; “Break;” preferred over “**return**” to satisfy switch rule.

**switch (** <expression> **) {**

**case** value1:

Code to execute if <expression> == value1

**break;**

**case** value2:

Code to execute if <expression> == value2

**break;**

...

**default:**

Code to execute if <expression> does not equal the

value following any of the cases

**break;**

**}**

**Loops:**

***while*** loop

**Note:** The number of times a loop repeats is referred to as *iterations*.

**while** ( condition )

{

// loop body

}

***do-while*** loop

do

{

// loop body

} while ( condition );

**Note:** Do-while loops execute loop body at least once; pay attention to closing semicolon after while statement.

***for*** loop

for ( initialization; condition; update )

{

// loop body

}

(Initialize loop variable; repeat loop while condition true; update of loop variable per round)

**E.g. “for** (x=1; x<=10 ; x++)” will run 10 times

**Note:** for complex nested loops, recommended to place process under a function to recall

**Pointers**

**#include <stdio.h>**

**void function(int \*, int \*, int \*);**

**int main(void)**

**{**

**int a = 9, b = -2, c = 5;**

**function(&a, &b, &c);**

**//Function Body**

**return 0;**

**}**

**void function(int \*x, int \*y, int \*z)**

**{**

**//Function Body**

**}**

**Note**:

Pointers are used for functions where more than one return value is needed.

**\*p** is a pointer to a variable **x**. **P** shares same address as **&x**. **\*p** shares same value as **x.** Altering any will correspondingly change the other.

**Modular Programming**

**f1.c** (Source file):

* Functions, as per function prototypes
* Other Module local function, variables and constants

**f1.h** (Header file):

* Function definitions
* Function prototypes

Case 1: Source files compiled and linked in one step

libm.a

math.h

main.c

f3.c

f3.h

f2.c

f2.h

f1.c

f1.h

Library  
file(s)

a.out

gcc main.c f1.c f2.c f3.c -lm

Compilation

and Linking

Executable  
file

Source  
files  
.c & .h

Case 2: Source files are compiled separately and then linked

f1.h

f2.h

f3.h

f1.c

f2.c

f3.c

main.c

Library  
file

math.h

f1.o

gcc –c f1.c

Libm.a

f2.o

gcc –c f2.c

a.out

gcc main.o f1.o f2.o f3.o -lm

Executable  
file

f3.o

gcc –c f3.c

Linking

gcc –c main.c

main.o

Object  
files

Compilation

Source  
files  
.c & .h

Note:

* Use **#include"…" to include your own header files** and **#include <…> to include system header files**.
* **Include “.h” files only in “.c” files**, otherwise duplicate inclusions may happen.

**Array**

**Element type array name[array size];**

**E.g. int array[n];**

Note: array refers to &array[0]. Array element type follows that of Array.

array[0]

array[1]

array[2]

array[3]

array [n-1]

**…**

**int arr[10]; // size of arr is 10**

**#define M 5**

**#define N 10**

**double foo[M\*N+8]; // size of foo is 58**

**int i;**

**float bar[i]; // variable-length array**

**// is illegal in cs1010!**

Not supported by ISO C90 standard.

gcc –pedantic will generate warning.

Array initialization

**Initialized at time of declaration:**

**// a[0]=54, a[1]=9, a[2]=10**

**int a[3] = {54, 9, 10};**

**// size of b is 3 with b[0]=1, b[1]=2, b[2]=3**

**int b[] = {1, 2, 3};**

**// c[0]=17, c[1]=3, c[2]=10, c[3]=0, c[4]=0**

**int c[5] = {17, 3, 10};**

**// c[0]=0**

**int c[n] = {0};**

**Incorrect initialization:**

**int e[2] = {1, 2, 3}; // warning issued: excess elements**

**int f[5];**

**f[5] = {8, 23, 12, -3, 6}; // too late to do this;**

**// compilation error**

**Initialization using “for loop”:**

**int i, array[n];**

**for (i=0; i<=(n-1); i++) //i<n may also be used**

**Array[i]=i;**

Array assignment

**Using loop:**

**int i;**

**for (i = 0; i < N; i++)**

**dest[i] = source[i];**

**#define N 10**

**int source[N] = { 10, 20, 30, 40, 50 };**

**int dest[N];**

**dest = source; // illegal!**

Syntax

**Function prototype:**

“[]” and “arr[]” are all acceptable and equivalent

**int sumArray(int [], int);**

*Alternate syntax using pointers (not advised)*

**int sumArray(int \*, int);**

**Function heading:**

“arr[n]” is acceptable but compiler ignores n

**int sumArray(int arr[], int size) { ... }**

*Alternate syntax using pointers (not advised)*

**int sumArray(int \*arr, int size) { ... }**

**Main function:**

Note:

Common mistake: foo[8]

For this specific function size cannot be more than 8

**int main(void) {**

**...**

**printf("sum is %d\n", sumArray(foo, 8));**

**...**

**}**

**//Function definition**

**int sumArray(int arr[], int size) {**

**...**

**}**

**Multi-Dimensional Array**

Initialization

*At time of declaration:*

**// nesting one-dimensional initializers**

**int a[3][5] = { {4, 2, 1, 0, 0},**

**{8, 3, 3, 1, 6},**

**{0, 0 ,0, 0, 0} };**

**// the first dimension can be unspecified**

**int b[][5] = { {4, 2, 1, 0, 0},**

**{8, 3, 3, 1, 6},**

**{0, 0, 0, 0, 0} };**

**// initializer without nesting**

**int c[3][5] = { 4, 2, 1, 0, 0, 8, 3, 3, 1, 6, 0, 0,**

**0, 0, 0 };**

**// initializer with implicit zero values**

**int d[3][5] = { {4, 2, 1},**

**{8, 3, 3, 1, 6} };**

**// initialize only first element to zero**

**int d[3][5] = { {0} };**

*Dual dimensional arrays are stored in row major order. E.g. arr[0][0]…arr[0][N],arr[1][0]…arr[1][N]..*

*User assignment using for loop*

**int main (void)**

**int r, //Loop counter for rows**

**c, //Loop counter for column**

**arr[ROWS][COLS]; //Array**

**{**

**//function body**

**for (r=0; r<ROWS; r++) //Across the columns then down the rows**

**for (c=0; c<COLS; c++)**

**scanf(“%d”, &arr[r][c]);**

**//function body**

**}**

Syntax

Function prototype:

**int arrayFunct(int [][MAXVALUE], int, int);**

Main Function:

**int arrayFunct(arr, 5, 5);**

Function Definition:

**int arrayFunct(int arr[][MAXVALUE], int row\_size, int col\_size) {...}**

**Recursion**

Base case(s)

*Principle*

**// Pre-cond: n >= 0**

**int fib(int n)**

**{**

**if (n < 2)**

**return n;**

**else**

**return fib(n-1) + fib(n-2);**

**}**

**// Pre-cond: n >= 0**

**int factorial(int n)**

**{**

**if (n == 0)**

**return 1;**

**else**

**return n \* factorial(n-1);**

**}**

Recurrent Relation

*Order of statements matter in recurrent relation:*

**≠**

**void mystery2(int n)**

**{**

**if (n>0) {**

**mystery2(n/10);**

**printf("%d", n%10);**

**}**

**}**

**void mystery1(int n)**

**{**

**if (n>0) {**

**printf("%d", n%10);**

**mystery1(n/10);**

**}**

**}**

Note:

**int countValue(int value, int arr[], int size) {**

**if (size == 0)**

**return 0;**

**if (value == arr[size-1])**

**return 1 + countValue(value, arr, size-1);**

**else**

**return countValue(value, arr, size-1);**

**}**

Can be simplified to:

**int countValue(int value, int arr[], int size) {**

**if (size == 0)**

**return 0;**

**else**

**return (value == arr[size-1]) +**

**countValue(value, arr, size-1);**

**}**

*Terminologies:*

Tail recursion is one in which recursive call is the last operation in the code

**Auxiliary function**

**int countValue(int value, int arr[], int size) //Driver Function**

**{**

**return countValue\_recur(value, arr, 0, size);**

**}**

Driver function is used to eliminate additional redundant parameters from the caller’s point of view.

**Characters**

* Characters are denoted as symbols enclosed in single quotes.

***ASCII***



***Character functions***

**#include <stdio.h>**

**int main(void)**

**{**

**char ch;**

**printf("Enter a character: ");**

**ch = getchar();**

**printf("The character entered is ");**

**putchar(ch);**

**putchar('\n');**

**return 0;**

**}**

*Reads a character ‘ch’ from stdin*

*Equivalent to scanf(“%c”, &ch);*

*Prints a character ’ch’ to stdout*

*Equivalent to printf(“%c”, ch);*

**d**

***<ctype.h> character functions:***

Note:

tolower(ch) and toupper(ch) do NOT change ch!

**// Week8\_CharacterDemo3.c**

**#include <stdio.h>**

**#include <ctype.h>**

**int main(void)**

**{**

**char ch;**

**printf("Enter a character: ");**

**ch = getchar();**

**if (isalpha(ch)) {**

**if (isupper(ch)) {**

**printf("'%c' is a uppercase-letter.\n", ch);**

**printf("Converted to lowercase: %c\n", tolower(ch));**

**}**

**if (islower(ch)) {**

**printf("'%c' is a lowercase-letter.\n", ch);**

**printf("Converted to uppercase: %c\n", toupper(ch));**

**}**

**}**

**if (isdigit(ch)) printf("'%c' is a digit character.\n", ch);**

**if (isalnum(ch)) printf("'%c' is an alphanumeric character.\n", ch);**

**if (isspace(ch)) printf("'%c' is a whitespace character.\n", ch);**

**if (ispunct(ch)) printf("'%c' is a punctuation character.\n", ch);**

**return 0;**

**}**

**Strings**

* A string is an array of characters, terminated by a null character '\0' (which has ASCII value of zero)
* Without **‘\0’**, array is not considered as a string.
* A string beginning with **‘\0’** prints nothing.

*Declaration and initialization*

**char fruit\_name1[MAX+1] = "apple"; //’\0’ automatically added**

**char fruit\_name2[MAX+1] = {'a','p','p','l','e','\0'};**

*Assignment*

**str[0] = 'e';**

**str[1] = 'g';**

**str[2] = 'g';**

**str[3] = '\0';**

*Reading strings from stdin*

**gets(str); //Reads in entire line(NOT RECOMMENDED)**

**fgets(str, size, stdin; //Reads size-1 char, or until newline**

**scanf("%s", str); //Reads until white space (IMPROPER)**

*Note on fgets:*

* + *On interactive input, fgets() also reads in the newline character* ***‘\n’***
  + *Hence, we may need to replace it with '\0' if necessary*

**fgets(str, size, stdin); //E.g. fgets(str,LENGTH+1,stdin);  
len = strlen(str);  
if (str[len – 1] == '\n')  
 str[len – 1] = '\0';**

*Printing strings to stdout*

**puts(str); // terminates with newline**

**printf("%s\n", str);**

*<string.h> string functions*

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**int main(void) {**

**int i, len, count = 0;**

**char str[31];**

**printf("Enter a string (at most 30 characters): ");**

**fgets(str,31,stdin);**

**len = strlen(str); //strlen() returns number of char in string**

**if (str[len – 1] == '\n')**

**str[len – 1] = '\0';**

**len = strlen(str); // check length again**

**for (i=0; i<len; i++) {**

**switch (toupper(str[i])) {**

**case 'A': case 'E':**

**case 'I': case 'O': case 'U': count++;**

**}**

**}**

**printf("Number of vowels: %d\n", count);**

**return 0;**

**}**

*Pointer to string*

name is a character array of 12 elements. namePtr is a pointer to a character.

Both have strings assigned.

Difference is name sets aside space for 12 characters, but namePtr is a char pointer variable that is initialized to point to a string constant of 9 characters.

**#include <stdio.h>**

**#include <string.h>**

**int main(void)**

**{**

**char name[12] = "Chan Tan";**

**char \*namePtr = "Chan Tan";**

**printf("name = %s\n", name);**

**printf("namePtr = %s\n", namePtr);**

**printf("Address of 1st array location for name = %p\n", name);**

**printf("Address of 1st array location for namePtr = %p\n",namePtr);**

**strcpy(name, "Lee Hsu");**

**namePtr = "Lee Hsu";**

**printf("name = %s\n", name);**

**printf("namePtr = %s\n", namePtr);**

**printf("Address of 1st array location for name = %p\n", name);**

**printf("Address of 1st array location for namePtr = %p\n",namePtr);**

**}**

name updated using strcpy().

namePtr assigned to another string using =

Address of first array element for name remains constant, string assigned to namePtr changes on new assignment.

*Multidimensional Array of strings*

**//Declaration**

**// where MAXNUM is the maximum number of names**

**// and STRSIZE is the size of each name**

**char fruits[MAXNUM][STRSIZE+1];**

**//Initialization**

**char fruits[][6] = {"apple", "mango", "pear"};**

**char fruits[3][6] = {"apple", "mango", "pear"};**

**//Output**

**printf("fruits: %s %s\n", fruits[0], fruits[1]);**

**printf("character: %c\n", fruits[2][1]);**

*Array of pointers to strings*

**//Declaration**

**char \*fruits[3]; //each array element is a pointer to a string**

**//Assignment**

**fruits[0] = "apple";**

**fruits[1] = "banana";**

**fruits[2] = "cherry";**

**//Declare and initialize**

**char \*fruits[] = {"apple", "banana", "cherry"};**

**fruits[0] = "pear"; // new assignment**

**//Output**

**for (i=0; i<3; i++)**;

**printf("%s\n", fruits[i]);**

*Common <string.h> functions*

**strlen(s1);**

Determines the length of the string **s**.  Returns the number of characters in the string **before** the '\0'

**strcmp(s1, s2);**

Compare the ASCII values of the corresponding characters in strings s1 and s2.

Returns:

* a negative integer if s1 is lexicographically less than s2, or
* a positive integer if s1 is lexicographically greater than s2, or
* 0 if s1 and s2 are equal

**strncmp(str1, str2, n);**

Compare first n characters of s1 and s2.

**strcpy(str1, str2);**

**Or strcpy(str, ”string”)**

Copy the string pointed to by s2 into array pointed to by s1.

Function returns new s1.

Function not limited to size of str1, i.e. size of str2 may be more and str1 will then take on the new size.

**strncpy(str1, str2, n);**

Copy first *n* characters of the string s2 into s1.

**strstr(str1, str2);**

Returns a pointer to the first instance of string s2 in s1.

Returns a NULL pointer if s2 is not found in s1.

**atoi(str);**

Converts string to integer value. I.e.

Char str=“123”;

Int value;

Value = atoi(str) //Value==123

**itoa(integer, str, 10);**

constructs a string representation of an integer.

**strcat(str1, str2);**

Appends the string **s2** to the end of character array **s1**.  The first character from **s2** overwrites the '\0' of **s1**. The value of **s1** is returned.

**strtok(str, “tokens”)**

Truncates string. The contents of this string are modified and broken into smaller strings using tokens as divider. See below for example

*strtok*

*/\* strtok example \*/*

*#include <stdio.h>*

*#include <string.h>*

*int* main ()

{

*char* str[] ="- This, a sample string.";

*char* \* pch;

printf ("Splitting string \"%s\" into tokens:\n",str);

pch = strtok (str," ,.-");

*while* (pch != NULL)

{

printf ("%s\n",pch);

pch = strtok (NULL, " ,.-");

}

*return* 0;

}

//Output

Splitting string "- This, a sample string." into tokens:

This

a

sample

string

**Searching/Sorting**

***Selection Sort***

**// To sort arr in increasing order**

**void selectionSort(int arr[], int size)**

**{**

**int i, start\_index, min\_index, temp;**

**for (start\_index = 0; start\_index < size-1; start\_index++)**

**{**

**// each iteration of the for loop is one pass**

**// find the index of minimum element**

**min\_index = start\_index;**

**for (i = start\_index+1; i < size; i++)**

**if (arr[i] < arr[min\_index])**

**min\_index = i;**

**// swap minimum element with element at start\_index**

**temp = arr[start\_index];**

**arr[start\_index] = arr[min\_index];**

**arr[min\_index] = temp;**

**}**

**}**

***Bubble Sort***

**// To sort arr in increasing order**

**void bubbleSort(int arr[], int size)**

**{**

**int i, limit, temp;**

**for (limit = size-2; limit >= 0; limit--)**

**{**

**// limit is where the inner loop variable i should end**

**for (i=0; i<=limit; i++)**

**{**

**if (arr[i] > arr[i+1]) // swap arr[i] with arr[i+1]**

**{**

**temp = arr[i];**

**arr[i] = arr[i+1];**

**arr[i+1] = temp;**

**}**

**}**

**}**

**}**

**Structures**

**Declaration:**

* Declared before function prototypes
* 3 methods of declaration:

*Method 1 : Anonymous structure type (Seldom used)*

**struct**

**{**

**char name[12];**

**int age;**

**char gender;**

**} player1, player2;**

*Method 2: Name structure with tag, then use tag name to declare variable*

**struct player\_t**

**{**

**char name[12];**

**int age;**

**char gender;**

**};**

**struct player\_t player1, player2;**

*Method 3: Use typedef to define and name structure type (Most preferred)*

**typedef struct**

**{**

**char name[12];**

**int age;**

**char gender;**

**} player\_t;**

**//Body**

**player\_t player1, player2;**

**Initialization:**

**typedef struct**

**{**

**char name[12];**

**int age**

**char gender;**

**} player\_t;**

**//Body**

**player\_t player1 = { "Brusco", 23, 'M' };**

**Accessing/Reading in structure members:**

* Use dot operator

*Accessing structure members*

Note:

* ‘&’ not needed for reading in string as player.name already points to the address of the first character.
* For nested structures, multiple dot operators may be used
* Only variables of the same type may be assigned to each other. *E.g. player2=player1;*

**player\_t player2;**

**strcpy(player2.name, "July");**

**player2.age = 21;**

**player2.gender = 'F';**

*Reading in structure members*

**player\_t player1;**

**printf("Enter name, age and gender: ");**

**scanf("%s %d %c", player1.name,**

**&player1.age, &player1.gender);**

**Pointers to structures:**

* When a structure is used in a function, no part of the original variable in main function is altered
* To alter the original structure variable and its members, a pointer to the structure may be used

**/\*Main function\*/**

**// to change player1’s name and age**

**change\_name\_and\_age(&player1);**

.

.

**.**

**/\*Function definition\*/**

**// to change a player’s name and age**

**void change\_name\_and\_age(player\_t \*player\_p)**

**{**

**char new\_name[12];**

**int new\_age;**

**printf("Enter new name and age: ");**

**scanf("%s %d", new\_name, &new\_age);**

**strcpy( (\*player\_p).name, new\_name );**

**(\*player\_p).age = new\_age;**

**}**

Passing in the address of player1

Pointer of structure type player\_t

Take note of parenthesis

*Arrow operator:*

**(\*player\_p).name equivalent to player\_p->name**

**Input/Output**

**Text Stream:**

* Consists of a sequence of characters organized into lines
* Each line contains 0 or more characters followed by a newline character ‘\n’
* Text streams stored in files can be viewed/edited easily
* Example: source code of C program

**3 Standard Streams:**

1. **stdin** points to a default input stream (keyboard)
2. **stdout** points to a default output stream (screen)
3. **stderr** points to a default output stream for error messages (screen)
   * *printf* writes output to stdout / *scanf* reads input from stdin

**File Operations:**

*Useful constants:*

**NULL**: null pointer constant

**EOF**: used to represent end of file or error condition

*Opening a file*

//Function prototype

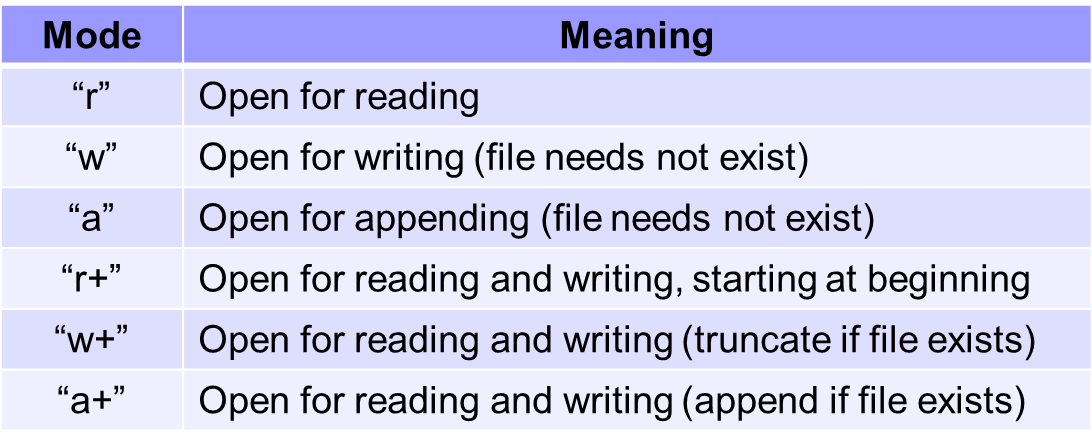
**FILE \*fopen(const char \*filename, const char \*mode);**

//Function usage

**FILE \*infile;**

**infile = fopen("demo1.in", "r");**

* *Returns* ***NULL*** *if error; otherwise, returns a pointer of* ***FILE*** *type*



**(Overwrite existing data)**

**(File must already exist)**

*Closing a file*

//Function prototype

**int fclose(FILE \*fp);**

//Function usage

**FILE \*infile;**

**fclose(infile);**

* Returns EOF if error detected

**I/O functions:**

*Formatted I/O*

* *fprintf, fscanf*
* Uses format strings to control conversion between character and numeric data
* **printf(" … ");** ≡ **fprintf(stdout, " … ");**
  + **fprintf** returns a negative value if an error occurs; otherwise, returns the number of characters written
* **scanf(" … ");** ≡ **fscanf(stdin, " … ");**
  + **fscanf** returns EOF if an input failure occurs before any data items can be read; otherwise, returns the number of data items that were read and stored

*Character I/O*

* Reads and writes single characters
* *fputc, putc , putchar , fgetc , getc , getchar , ungetc*
  + **fputc** and **putchar** return EOF if a write error occurs; otherwise, return character written
  + **fgetc** and **getchar** return EOF if a read error occurs or end of file is reached; otherwise, return character read. (**feof** or **ferror** to distinguish)
  + **putchar** ≡ **fputc(stdout), getchar** ≡ **getc(stdin)**

**int ch = 'A';**

**FILE \*fp;**

**putchar(ch); // writes ch to stdout**

**fp = fopen( ... );**

**fputc(ch, fp); // writes ch to fp**

**int ch;**

**FILE \*fp;**

**ch = getchar(); // reads a char from stdin**

**fp = fopen( ... );**

**ch = fgetc(fp); // reads a char from fp**

*Line I/O*

* *fputs, puts , fgets , gets* 
  + ***fputs*** *and* ***puts*** *return EOF if a write error occurs; otherwise, return a non-negative number*
  + ***fgets*** *and* ***gets*** *store a null character at the end of the string*
  + ***fgets*** *and* ***gets*** *return a null pointer if a read error occurs or end-of-file is encountered before storing any character; otherwise, return first argument*
* Reads and writes lines
* Used mostly for text streams

**FILE \*fp;**

**// writes to stdout with newline character appended**

**puts("Hello world!");**

**fp = fopen( ... );**

**// writes to fp without newline character appended**

**fputs("Hello world!", fp);**

**char s[100];**

**FILE \*fp;**

**gets(s); // reads a line from stdin**

**fp = fopen( ... );**

**fgets(s, 100, fp); // reads a line from fp**

***fgets* in detail**

* Prototype:   
  **char \*fgets(char \*s, int n, FILE \*fp)**
  + ***s*** is a pointer to the beginning of a character array
  + ***n*** is a count
  + ***fp*** is an input stream
* Characters are read from the input stream ***fp*** into ***s*** until
  + a newline character is seen,
  + end-of-file is reached, or
  + *n* – 1 characters have been read without encountering newline character or end-of-file
* If the input was terminated because of a newline character, the newline character will be stored in the array before the terminating null character
* If end-of-file is encountered before any characters have been read from the stream,
  + **fgets** returns a null pointer
  + The contents of the array ***s*** are unchanged
* If a read error is encountered,
  + **fgets** returns a null pointer
  + The contents of the array ***s*** are indeterminate
* Whenever **NULL** is returned, **feof** or **ferror** should be used to determine the status

**Detecting EOF/Errors:**

* Each stream is associated with two indicators: error indicator & end-of-file (EOF) indicator
* Both indicators are cleared when the stream is opened
* Encountering end-of-file sets end-of-file indicator
* Encountering read/write error sets error indicator
* An indicator once set remains set until it is explicitly cleared by calling **clearerr** or some other library function
* <stdio.h> library functions:
  + **feof** returns a non-zero (true) value if the end-of-file indicator is set; otherwise returns 0 (false)
  + **ferror** returns a non-zero value if the error indicator is set; otherwise returns 0