Unix
Bash
C
GNU
Systems

Software Systems

Lectures Week 6

Introduction to C part 2

(STDIO.H, STRING.H, Pointers, Arrays)

Prof. Joseph Vybihal

Computer Science

McGill University

Week 6 Lecture 1

STDIO.H & STDLIB.H

STDIO.H

Defines three forms of I/O:

- Console
- Stream
- Files

Today's lecture will cover Console I/O and Stream I/O

Forms of I/O

Console I/O

- Input and output focused on the keyboard and screen.
  - Other forms: mouse, touch screen.
- Related to the computer the user is interacting with directly.

Stream I/O

- An abstraction.
- A logical or physical device that transmits/consumes n bytes of data, one byte at a time, in a continuous sequence over time.

File I/O

- Reading and writing to a file on disk.
  - This is a special case of stream I/O

STDIN, STDOUT, STDERR

Three standard streams exist in Unix and Linux:

- The input stream (stdin)
  - All C language commands can accept input from stdin.
  - By default stdin is attached to the keyboard.
  - The stdin can be redirected to other input sources.
- The output stream (stdout)
  - All C language commands can write to stdout.
  - By default stdout is attached to the screen.
  - The stdout can be redirected to other output sinks.
- The error stream (stderr)
  - All run-time errors and C error commands write to stderr.
  - By default stderr is attached to the screen.
  - The stderr can be redirected to other output sinks.

Example

We have seen:

c = getc(stdin);

One single character is extracted from the stdin stream, whatever is currently there, and returned.

Example

We have seen:

Bash-prompt $ ls > filename

There is one command: **ls**

Normally the output would show on screen.

The > symbol changes stdout attaching it temporarily to filename.

Example

We have seen:

Bash-prompt $ ls | more

There are two commands: **ls** and **more**.

The stdout for **ls** and the stdin for **more** are attached together.

Stdio Library

<stdio.h>:

constants

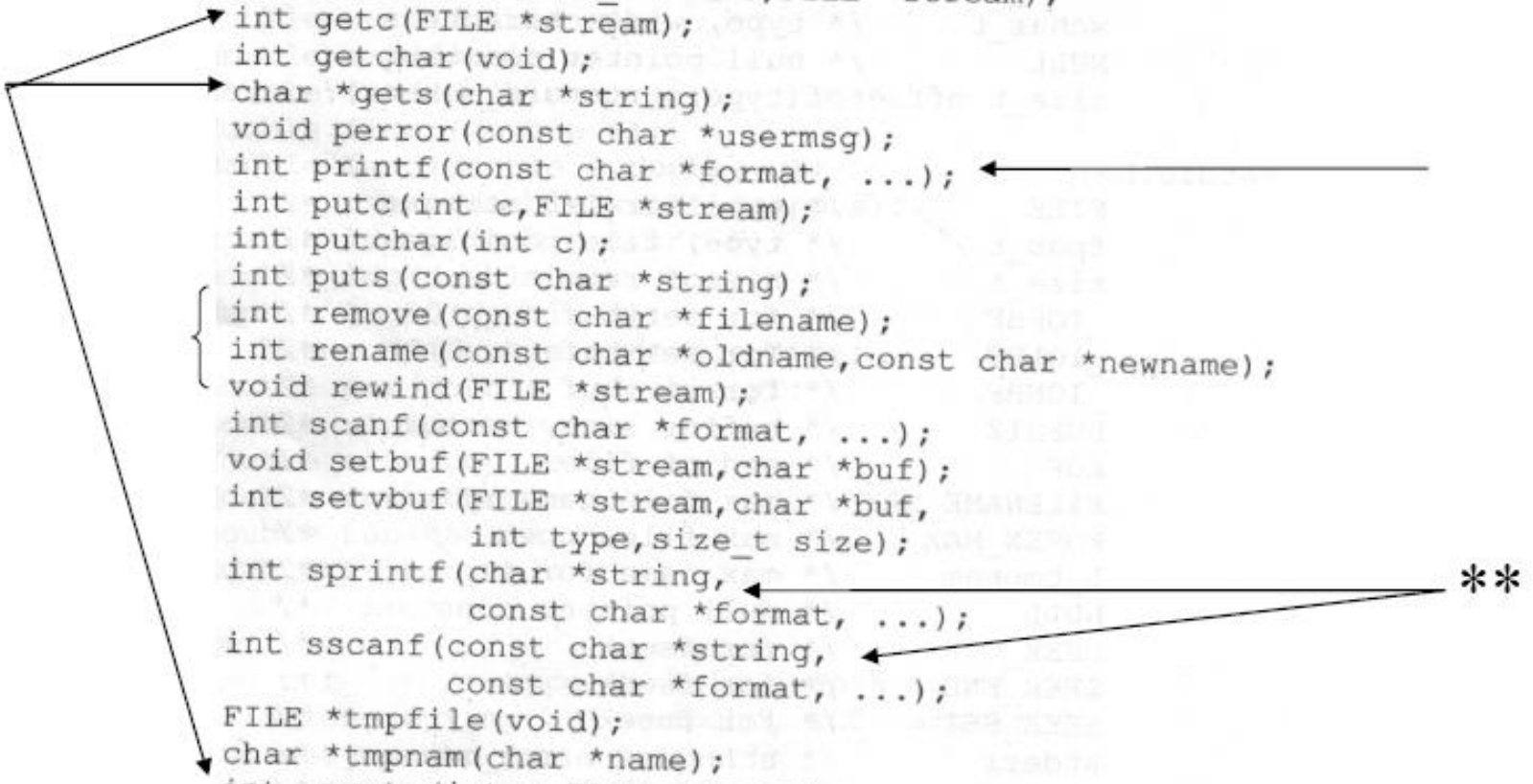
```
FILE            /* type for I/O streams    */
fpos_t          /* type, file position     */
size_t          /* sizeof result           */
_IOFBF          /* for setvbuf             */
_IOLBF          /* for setvbuf             */
_IONBF          /* for setvbuf             */
BUFSIZ          /* buffer size, setbuf     */
EOF             /* end of file             */
FILENAME_MAX    /* max file name size      */
FOPEN_MAX       /* max files open          */
L_tmpnam        /* max name for tmpnam     */
NULL            /* null pointer constant   */
SEEK_CUR        /* for fseek               */
SEEK_END        /* for fseek               */
SEEK_SET        /* for fseek               */
stderr          /* standard error stream   */
stdin           /* standard input stream   */
stdout          /* standard output stream  */
TMP_MAX         /* max tmpnam files        */
```

functions

```
void clearerr(FILE *stream);
int fclose(FILE *stream);
int feof(FILE *stream);
int ferror(FILE *stream);
int fflush(FILE *stream);
int fgetc(FILE *stream);
int fgetpos(FILE *stream,fpos_t *pos);
char *fgets(char *string,int n,FILE *stream);
FILE *fopen(const char *name,const char *options);
int fprintf(FILE *stream,const char *format, ...);
int fputc(int c,FILE *stream);
int fputs(const char *string,FILE *stream);
size_t fread(void *ptr,size_t size,
             size_t count,FILE *stream);
```

Stdio Library

```
FILE *freopen(const char *name,
              const char *options,
              FILE *stream);
int fscanf(FILE *stream,const char *format, ...);
int fseek(FILE *stream,long offset,int origin);
int fsetpos(FILE *stream,const fpos_t *pos);
long ftell(FILE *stream);
size_t fwrite(const void *ptr,size_t size,
              size_t count,FILE *stream);
int getc(FILE *stream);
int getchar(void);
char *gets(char *string);
void perror(const char *usermsg);
int printf(const char *format, ...);
int putc(int c,FILE *stream);
int putchar(int c);
int puts(const char *string);
int remove(const char *filename);
int rename(const char *oldname,const char *newname);
void rewind(FILE *stream);
int scanf(const char *format, ...);
void setbuf(FILE *stream,char *buf);
int setvbuf(FILE *stream,char *buf,
            int type,size_t size);
int sprintf(char *string,
            const char *format, ...);
int sscanf(const char *string,
           const char *format, ...);
FILE *tmpfile(void);
char *tmpnam(char *name);
int ungetc(int c,FILE *stream);
int vfprintf(FILE *stream,const char *format,
             va_list ap);
int vprintf(const char *format,va_list ap);
```



Important STDIO.H functions

- getc, putc, and puts
- getchar, putchar
- fgets
- printf
- scanf
- sprintf
- sscanf

- The file functions are for another lecture…

The getc and puts functions

We have seen these before:

- int getc(STREAM);
  - Where STREAM is stdin, from a file, or other input source
  - It returns the ASCII integer code for the character inputted
    - int asci = getc(stdin);

- int puts(STRING)
  - Where STRING is a series of characters
  - The string is printed to the screen (console)
  - It returns an error code
    - puts(“Hello”);
    - int c = puts(“Hello”);
    - if (puts(“Hello”) == 0) ….

The putc function

Single character stream output:

- int putc(CHARACTER, STREAM);
  - Where STREAM is stdout, to a file, or other output sink
  - Where CHARACTER is a char or int ASCII value
  - It returns error code
    - int errorcode = putc('a', stdout);
    - putc(x, stdout);

The getchar function

This is a console command:

- int getchar(void);
  - The void indicates that there are no arguments.
  - It returns the ASCII of the character read, or error code.
  - It stores the user's entire input into a buffer and then returns a single character from that buffer each time getchar is used.
  - Once the buffer is empty, all the characters have been returned/removed, the function once again stops to read characters into its buffer.
  - When the user presses enter input to the buffer ends.
- Example:
  - int c = getchar(); // user enters: My name is Bob<CR>. Int c = 'M'.
    - Each subsequent call to getchar returns the next char: 'y', then ' ', 'n',…
    - Until there are no more characters
    - When buffer is empty it reads from the console.

Key & Screen Example

```
#include <stdio.h>

int main(void)
{
   char c = '0';

   puts("Input characters until x: ");

   while(c != 'x' && c != 'X')
   {
      c = getchar();
      if (c >= '0' && c <= '9') continue;

      putchar(c); // echo what was read in
   }

   return 0;  // no errors
}
```

What does this do?

The putchar function

This is a console command:

- int putchar(CHARACTER);
  - It returns error code.
  - Outputs the CHARACTER to the screen.
- Example:
  - int errorcode = putchar('A');
  - putchar('B');

The fgets function

This is a stream command:

- POINTER fgets(ARRAY, LIMIT, STREAM);
  - Returns
    - On success a POINTER to the ARRAY.
    - On failure a POINTER to NULL.
    - If no data, or at the end of data, returns POINTER to NULL.
  - Reads at most LIMIT characters from STREAM and store in ARRAY as ASCII codes.
  - The finction fgets inserts a \0 at the end of the stream to indicate the last character. The '\0' character is called the null character.
- Example:
  - char array[30];
  - fgets(array, 29, stdin);
  - char *x = fgets(array, 29, stdin);
  - if (x == NULL) …

I/O Example

```
#include <stdio.h>

int main(void)
{
   char name[30];
   char gender;

   puts("Input your name: ");
   fgets(name, 29, stdin);

   puts("Gender: ");
   gender = getchar();

   puts("Welcome, ");
   puts(name);
   puts(".");

   return 0;
}
```

What does this do?

The printf function

Important console function:

- int printf(STRING, OPTIONAL_ARGUMENTS);
  - Outputs and formats all types of data
  - Returns
    - On success the number of arguments printed to screen.
    - On failure a zero.
  - STRING is the text displayed to the screen.
    - STRING contains escape-character symbols \ and %
    - Example STRING: “I am 12 years old” // simple string, no new line
    - Example STRING: “I am 12 years old \n” // string with new line
    - Example STRING: “I am %d years old \n” // integer inserted into string
- Example:
  - printf(“I am 12 years old.\n”);
  - printf(“I am %d years old.\n”, 12);
  - printf(“I am %d years old.\n”, age);

Escape characters

Escape characters are used for formatting:

- The backslash (\) character
  - \n new line
  - \t tab
  - \a bell
  - \b backspace with no erase
  - \r carriage return
  - \\ backslash
- The percentage (%) character formats variables
  - Format: % SIGN SIZE TYPE
    - %: required
    - SIGN: + -, optional
      - + = normal justification, - = reverse justification
    - SIZE: integer, optional
    - TYPE: d,c,f,s, required
      - d = integer, c = character, f = float, s = string

Examples

```
int age = 12;
printf("I am %d years old.\n", age);
printf("I am %10d years old.\n", age);
printf("I am %-10d years old.\n", age);
```

Numbers are right justified.

Characters and strings are left justified.

Examples

```
float age = 12.5;
printf("I am %f years old.\n", age);
printf("I am %5.1f years old.\n", age);
```

%5.1 → _ _ _ . _

The scanf function

Important console function:

- int scanf(STRING, &VARIABLES);
  - Reads all types of data
  - Returns
    - On success the number of arguments read from keyboard.
    - On failure a zero.
  - STRING is the text format expected from the keyboard.
    - Follows all the same rules we saw with printf
  - VARIABLES
    - At least one variables must be present
    - Leading &
      - Required for regular variables
      - Not used for pointers
- Example:
  - scanf("%d", &age); // reading an integer number into variable age

Example

```
#include<stdio.h>
int main(void) {
  char name[30]; int age;

  printf("Enter your name: ");
  scanf("%s", name);

  printf("Enter your age: ");
  scanf("%d", &age);

  printf("Welcome %s, you are %d years old.\n", name, age);

  return 0;
}
```

Example

```
#include<stdio.h>
int main(void) {
  int a, b, c;

  printf("Enter three numbers with spaces: ");
  scanf("%d %d %d", &a, &b, &c);

  printf("You entered %d, %d, and %d.\n", a, b, c);

  return 0;
}
```

Important

- The function scanf does not check for the size of the array.
  - If the user enters more characters than the size of the array, C does not crash, it will accept even the extra characters without changing the size of the array, resulting in interesting side effects.
  - This is a system feature allowing programmers to have freedom in accessing and manipulating memory will fewer imposed language rules.

Important

- All input: gets, getc, scanf, etc. do not handle mixed input correctly due to the carriage return issue.
  - %c and %s (or characters and strings) accept the enter key (or carriage return) as a valid input character, and so will read it into the variable.
  - %d and %f (or numbers) only accept numerical values ignoring all other characters.
    - %d will not accept the letter ‘a’ as a valid integer (same for %f)
    - %d and %f will not accept the enter key as a valid number
  - This leads to an interesting usage problem…

Usage Problem

```
#include<stdio.h>
int main(void) {
  int a;
  char array[30];

  // This works
  scanf("%s", array);
  scanf("%d", &a);

  return 0;
}
```

The string entered is saved in array and the integer entered is saved in the variable a.

```
#include<stdio.h>
int main(void) {
  int a;
  char array[30];

  // This fails
  scanf("%d", &a);
  scanf("%s", array);

  return 0;
}
```

The integer entered is stored in the variable a, but the carriage return is still in the buffer. The array reads the carriage return.

Usage Problem Solution

Use a temporary variable to store the carriage return.

- The garbage array captures the carriage return
- The scanf with the array will now wait for the user to input their information

```
#include<stdio.h>
int main(void) {
  int a;
  char array[30];
  char garbage[10];

  // This fails
  scanf("%d", &a);
  scanf("%s", garbage);
  scanf("%s", array);

  return 0;
}
```

The sscanf and sprintf Functions

These two functions are identical to scanf and printf except they do not print to the screen or read from the keyboard:

- sprint(CHAR_ARRAY, STRING, VARIABLES);
- sscanf(CHAR_ARRAY, STRING, VARIABLES);

For sprintf the output goes to CHAR_ARRAY

For sscanf the input comes from CHAR_ARRAY

The sscanf and sprintf Functions

```
char array[100];
int a, b, c;

// Developers assume that users do not follow instructions
printf("Please enter three numbers and press enter at the end: ");
scanf("%s", array);

// If the user input something incorrectly program won't crash, instead zeros
// will be assigned to the offending variable(s)
sscanf(array, "%d %d %d", &a, &b, &c);
```

The sscanf and sprintf Functions

```
char array[100];
char name[30];
int age;
float salary;

// Build a formatted string in memory before you output it
sprintf(array, “Employee: %s, Salary= %6.2f, Age= %3d”, name, salary, age);
```


```
#include <stdlib.h>
```

Important Elements

- NULL = 0
- EXIT_FAILURE = 1
- EXIT_SUCCESS = 0
- int x = rand(void); // 0 to RAND_MAX

32767

- int system(string)
- float x = atof(string)
- int y = atoi(string)
- int z = abs(int)
- void exit(int)

Example

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

int main(void)
{
   int randomValue, result, factor = 10;

   randomValue = rand();

   result = factor * randomValue;

   return EXIT_SUCCESS;
}
```

Example

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

int main()
{
    char beingCareful[300];
    int age;
    float salary;

    printf("What is your age?: ");
    gets(beingCarefule);

    age = atoi(beingCareful);

    printf("What is your salary?: ");
    gets(beingCareful);

    salary = atof(beingCareful);

    return EXIT_SUCCESS;
}
```

Example

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

int main(void)
{
   char string[200];

   puts("Please input a command: ");
   gets(string);

   system(string);
}
```

system("ls");
system("./program");
system("cd docs;cp a b; ls");


```
int errorcode = system("./a.out path filename");

// Usage 1 - error / status messages

if (x != EXIT_SUCCESSS)

// Usage 2 - passing messages back

switch(x)
{
case 0: // message 1
case 1: // message 2
case 2: // message 3
}
```

The actual value return by system depends on your OS, but it is based on the shell's error codes.

functions

```
void abort(void);
int abs(int i);
int atexit(void (*wrapfunc)(void));
double atof(const char *s);
int atoi(const char *s);
long atol(const char *s);
void * bsearch(const void *key,
               const void *table,
               size_t N, size_t keysize,
               int (*compar)(const void *,
                             const void *));
void *calloc(size_t N,size_t size);
div_t div(int top,int bottom);
void exit(int status);
void free(void *ptr);
char *getenv(const char *name);
long labs(long n);
ldiv_t ldiv(long top,long bottom);
void *malloc(size_t size);
int mblen(const char *mb,size_t N);
size_t mbstowcs(wchar_t *wcstring,
                const char *mbstring,
                size_t N);
int mbtowc(wchar_t *wc,const char *mb,size_t N);
void qsort(void *table,size_t N,size_t size,
           int (*compar)(const void *,
                         const void *));
int rand(void);
void *realloc(void *oldp,size_t size);
void srand(unsigned seed);
double strtod(const char *s,char **ptr);
long strtol(const char *s,char **ptr,int base);
unsigned long strtoul(const char *s,
                      char **ptr,int base);
int system(const char *command);
size_t wcstombs(char *mbstring,
                const wchar_t *wcstring,
                size_t N);
int wctomb(char *mb, wchar_t wc);
```

**

**

later

**

Week 6 Lecture 2

Pointers, Strings, and STRING.H

Pointers, Strings, and string.h

Pointers are variables that can directly reference other computer structures through the structure's address.

Strings, in C, are implemented with pointers.

The library string.h has many string manipulation functions.

Pointers

By example:

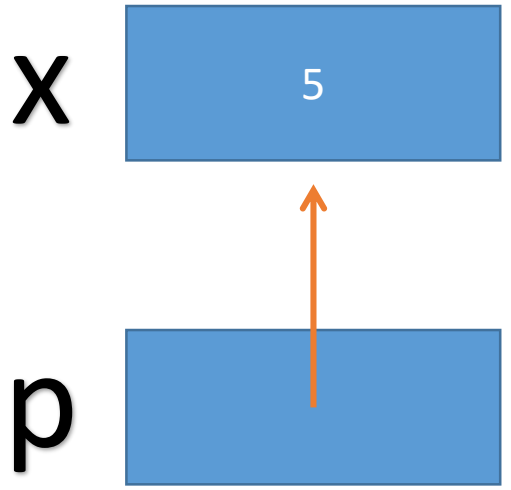
- int x = 5; // is a simple structure that stores integer numbers
- int *p; // create a variable that can store the address of another structure
- p = &x; // p has been assigned the address of x
  p is said to be “pointing to” x
  p does not have the value of x, p just “knows” where x is located in the computer’s memory
- printf(“%d”, x); // prints 5 to the screen
- printf(“%d”, *p); // prints 5 to the screen

& → return the address of a structure (referencing)
* → using the address, get the value from the location in the computer’s memory (derefrencing)

Pointers Visually

By example:

- int x = 5;
- int *p;
- p = &x;
- printf("%d", x);
- printf("%d", *p);



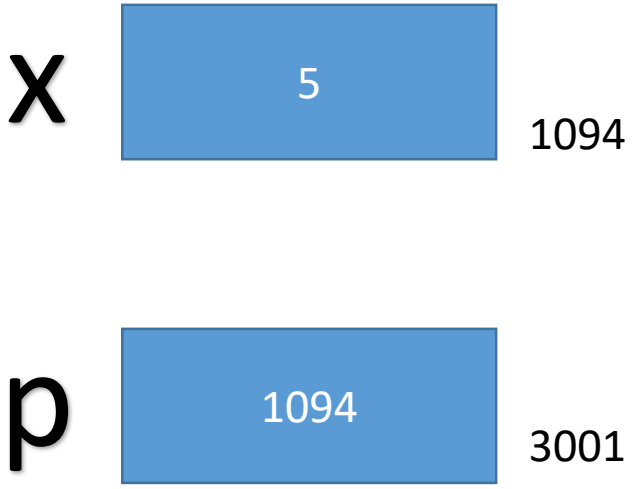
&	creates the arrow
*	follows the arrow

Pointers

Pointers, in C, are stored as integer numbers.

This means we can do math with them.

```
x = 5;
p = &x;

printf("%d", *p);
printf("%d", p);

p = p + 1;
printf("%d", p);
```



Strings

Definition:

- In C, a string is defined as a constant, a series of contiguous characters ending with a special end-of-string character called the null character, represented by '\0'.

Syntax:

- char *p = "my name is bob";
- The p is a variable pointer
  - The p points to the first character (in this case 'm')
- The "my name is bob" is a static constant value
  - It cannot be edited
  - It cannot be written over

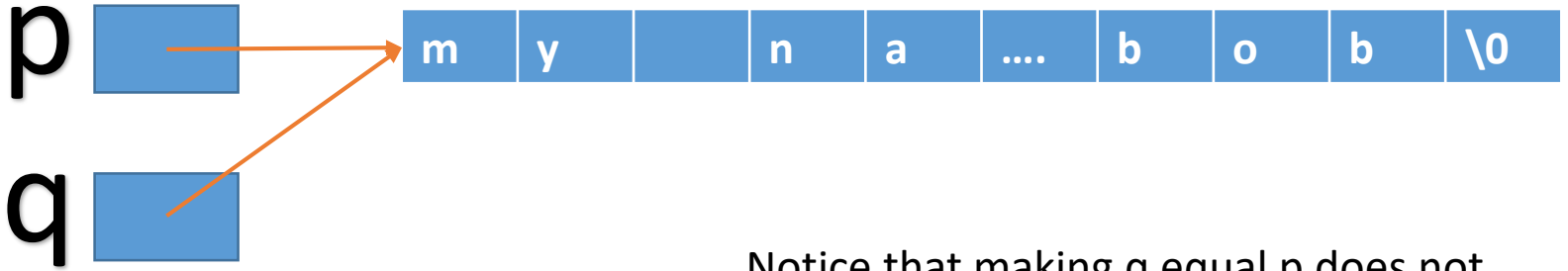
String Manipulation

```
char *p = “my name is bob”;
char *q;

printf(“%s”, p);    // outputs: my name is bob

printf(“%s”, (p+1)); // outputs: y name is bob

q = p + 3;
printf(“%s”, q); // outputs: name is bob
```

Note: The %s in printf will print character by character from the string until it comes to the \0 character. If the \0 character was missing then printif would not stop printing until it came to a \0 or crashed somewhere in memory.

String Manipulation

```
char *p = “my name is bob”;
char *q;

q = p;
```



Notice that making q equal p does not create a new copy of the string.

String Manipulation

```
char *p = “my name is bob”;
scanf(“%s”, p); // this will crash or behave strangely
```

Since p is a variable this compiles.

But p points to a constant string.

At run-time scanf will attempt to write over the constant, which is illegal. The solution is to use array. Arrays are not constant.

String Manipulation

```
char *a=”bob”;
char *b=”bob”;

if (a == b) // false
```

Note: a and b contain the same info
But this information is in a different location.

String.h

Important Functions

- int strcmp(char *s1, char*s2)
- int strncmp(char *s1, char*s2, int len)
- int strlen(char *string)
- char *strcpy(char *dest, char*src) … strncpy
- char *strcat(char *dest, char*src) … strncat
- void *memset(char* string, char character, int len)

The strcmp function

Calculates the difference between two strings:

- Returns 0 when equal
- Returns >0 when first argument is larger than second
- Returns <0 when first argument is smaller than second

```
char *a=”bob”;
char *b=”bob”;

if (strcmp(a,b)==0) // then it is the same
if (strcmp(a,b)!=0) // then it is not the same

if (!strcmp(a,b)) // then this is the SAME, don’t do this…
```


```
int x = strcmp("bob", "bob");      //     0
int x = strcmp("bob","mary");      //    <0
int x = strcmp("mary","bob");      //    >0

int x = strncmp("mark","mary",3); //  0

int x = strlen("mary");                 // x = 4

char array[100];
strcpy(array, "first words");
strcat(array, "second words");
printf("%s", array);                    // outputs: first words second words

char array2[100];
memset(array2, '*', 50);                // first 50 cells are *
```

#include <string.h>

```
void    *memccpy(void *restrict, const void *restrict, int, size_t);

void    *memchr(const void *, int, size_t);
int     memcmp(const void *, const void *, size_t);
void    *memcpy(void *restrict, const void *restrict, size_t);
void    *memmove(void *, const void *, size_t);
void    *memset(void *, int, size_t);
char    *strcat(char *restrict, const char *restrict);
char    *strchr(const char *, int);  <----------------------------------
int     strcmp(const char *, const char *);
int     strcoll(const char *, const char *);
char    *strcpy(char *restrict, const char *restrict);
size_t  strcspn(const char *, const char *);
char    *strdup(const char *); <-----------------------------------------
char    *strerror(int);
int    *strerror_r(int, char *, size_t);
size_t  strlen(const char *);
char    *strncat(char *restrict, const char *restrict, size_t);
int     strncmp(const char *, const char *, size_t);
char    *strncpy(char *restrict, const char *restrict, size_t);
char    *strpbrk(const char *, const char *);
char    *strrchr(const char *, int);
size_t  strspn(const char *, const char *);
char    *strstr(const char *, const char *); <---------------------------
char    *strtok(char *restrict, const char *restrict);
char    *strtok_r(char *, const char *, char **);
size_t  strxfrm(char *restrict, const char *restrict, size_t);
```

Implementing string.h functions using pointers.

```
int strlen(char *str)
{
        int i;

        for(i=0; *str != '\0'; i++, str++);

        return i;
}
```

```
int strcmp(char *s1, char *s2)
{
        for(; *s1!='\0' && *s2!='\0' && *s1==*s2; s1++,s2++);

        return *s1-*s2;
}
```

Week 6 Lecture 3

Arrays, Strings, math.h, ctype.h

Arrays

Syntax:

- TYPE NAME [SIZE];
- TYPE NAME [COLS][ROWS];
- TYPE NAME [COLS][ROWS][LAYERS];
- TYPE NAME [COLS][ROWS][LAYERS][CUBES];
- Etc.

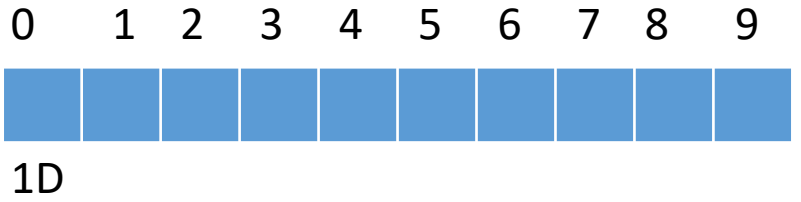
Multidimensional arrays are easy to create an manipulate in C.

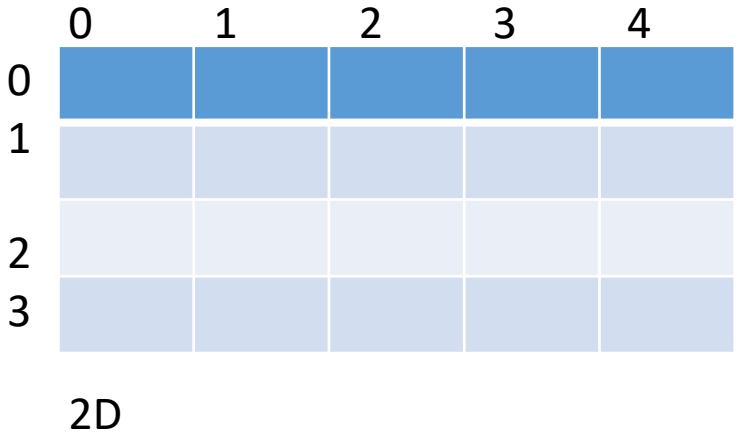
C arrays are variables, which means we can write to arrays and read from arrays.

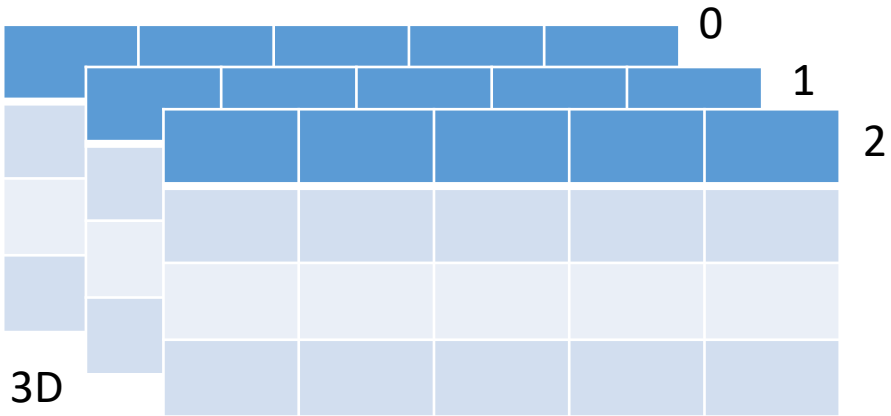
Arrays

Syntax:

- TYPE NAME [SIZE];
  - int data[100];
  - char name[30];
- TYPE NAME [COLS][ROWS];
  - int picture[100][200];
- TYPE NAME [COLS][ROWS][LAYERS];
  - char world[100][100][50];







Find a value

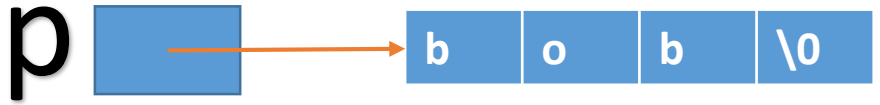
```
#include <stdio.h>
int main(void) {
    int numbers[5] = {5, 10, 15, 20, 25};
    int n, x;

    scanf("%d", n);

    for(x=0; x<5; x++) {
      if (numbers[x] == n) { puts("Found\n"); break; }
    }

    if (x == 5) puts("Not found\n");
    return 0;
}
```

Multiply

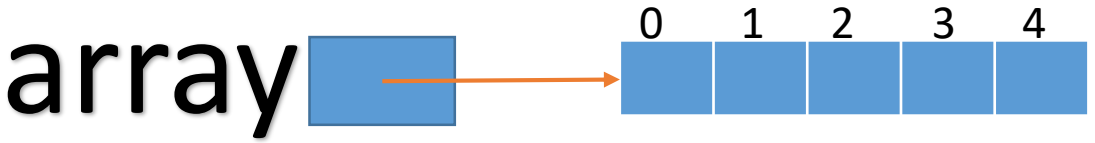
```
#include <stdio.h>
int main(void) {
    int vector[5] = {5, 10, 15, 20, 25}, result[5];
    int matrix[5][2] = { {1,2,3,4,5}, {6,7,8,9,0} };
    int a, b, multsum;

    for (a=0; a<5; a++) {
      multsum = 0;
      for (b=0; b<2; b++) {
            multsum += (vector[a] * matrix[a][b]);
      }
      result[a] = multsum;
    }
    return 0;
}
```

Arrays, Strings, and pointers

char *p = “bob”;



char array[5];



- Notice that they are structurally similar.
- This means they are interchangeable in many contexts within C.
- TYPE* and TYPE[] are interchangeable.

Example

```
char array[100];

scanf("%s", array);

strcat(array, " extra stuff");

printf("%s", array); // What does it print out ?
```

Example

```
char array[100];
char *p, *q;

p = array;
printf("%s", p); // what is printed out?
printf("%s", (p+2)); // what is printed out?


q = &array[5]; // since we are not pointing to the first cell
printf("%s", q); // what is printed out?
```

#include <math.h>

Important Elements

- Standard math:
  - double y = sqrt(double);
  - double y = pow(base,exponent);
  - int x = abs(int);
  - double y = fabs(double);
  - double x = floor(double);
  - double x = ceil(double);
- Trigonometry:
  - sin, cos, tan, asin, acos, atan

```
X = sqrt(25);

X = pos(10,2);



10.9 -> 10.0

10.2 -> 11
```


```
<math.h>:
        HUGE_VAL    /* large double value */
        double acos(double x);
        double asin(double x);
        double atan(double x);
        double atan2(double y,double x);
        double ceil(double x);
        double cos(double x);
        double cosh(double x);
        double exp(double x);
        double fabs(double x);
        double floor(double x);
        double fmod(double x,double y);
        double frexp(double x,int *expptr);
        double ldexp(double x,int N);
        double log(double x);
        double log10(double x);
        double modf(double x,double *yp);
        double pow(double x,double y);
        double sin(double x);
        double sinh(double x);
        double sqrt(double x);
        double tan(double x);
        double tanh(double x);
```

#include <ctype.h>

Important Elements

- Case manipulation:
  - int c = toupper(int);
  - int c = tolower(int);
- Character testing:
  - int x = isalpha(int);
  - int x = isalphanum(int);
  - int x = isdigit(int);

```
if(toupper(c)=='X')
```

```
if(isalpha(c))
```

Note: char is in int.

<ctype.h>:

```
int isalnum(int c);
int isalpha(int c);
int iscntrl(int c);
int isdigit(int c);
int isgraph(int c);
int islower(int c);
int isprint(int c);
int ispunct(int c);
int isspace(int c);
int isupper(int c);
int isxdigit(int c);
int tolower(int c);
int toupper(int c);
```

Counting Characters

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
    char *message="Hi there 123";
    int digits=0, letters=0, other=0, i;
    char c;

    for(i=0; i<strlen(message); i++)
    {
        c = *(message+i);

        if (isalpha(c)) letters++;
        else if (isdigit(c)) digits++;
        else other++;
    }
}
```