Stdio.h

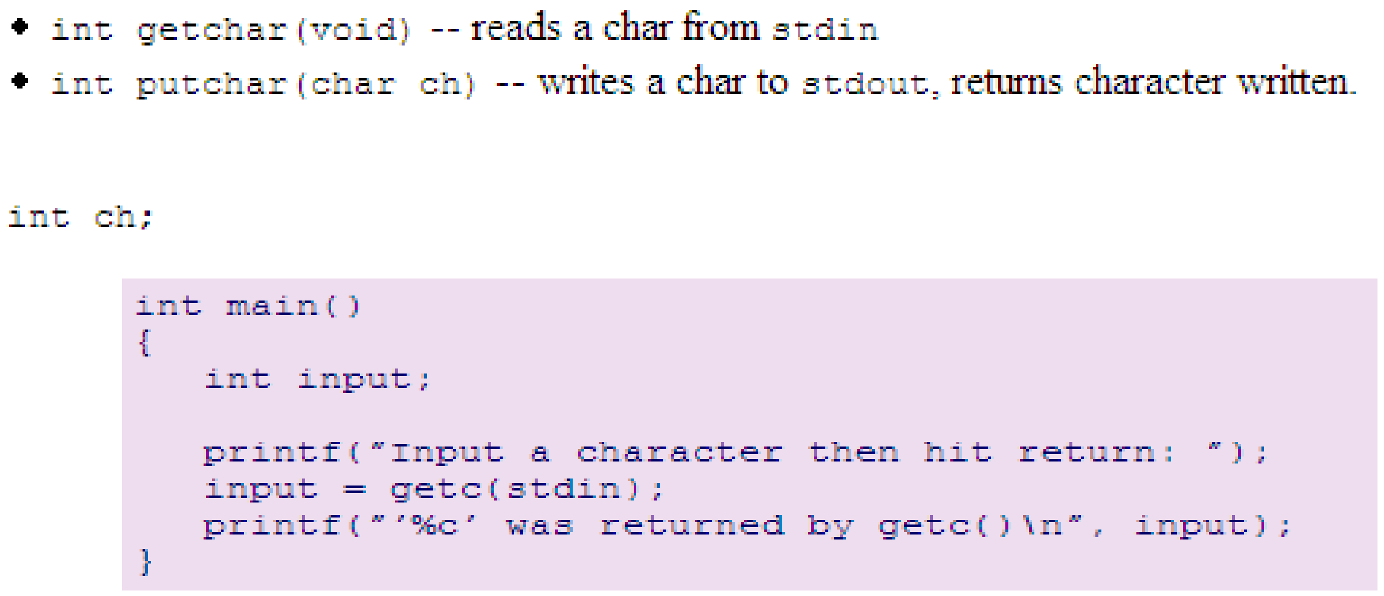
http://www.cplusplus.com/reference/clibrary/cstdio/

Stdio.h

- Stdio.h - #include <stdio.h>
  - stands for "standard input-output header,"
    - http://www.opengroup.org/onlinepubs/007908799/xsh/stdio.h.html
- Functions declared in stdio.h are extremely **popular**, since as a part of the C standard library, they are guaranteed to work on any platform which supports C
  - gets, puts, getchar, putchar
  - fopen, fclose, fscanf, fprintf, scanf/printf

Basic I/O
getchar / putchar

- There are a couple of function that provide basic I/O facilities.
- getchar() and putchar(). They are defined and used as follows:



- `int getchar(void)` -- reads a char from `stdin`
- `int putchar(char ch)` -- writes a char to `stdout`, returns character written.

```
int ch;
```

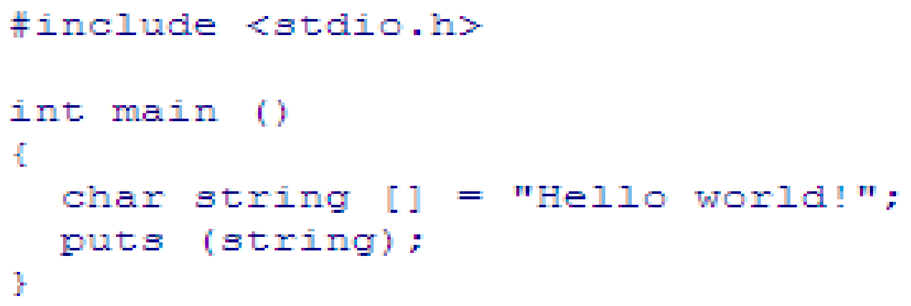
```
int main()
{
    int input;

    printf("Input a character then hit return: ");
    input = getc(stdin);
    printf("'%c' was returned by getc()\n", input);
}
```

Basic I/O
puts / fputs

- The function fputs writes the string pointed to by str to the stream pointed to by stream.

  The function puts writes the string str, and a terminating newline character, to the stream stdout.
- **#include <stdio.h>**
  - int **fputs** (const char *str, FILE *stream)
  - int **puts** (const char *str)

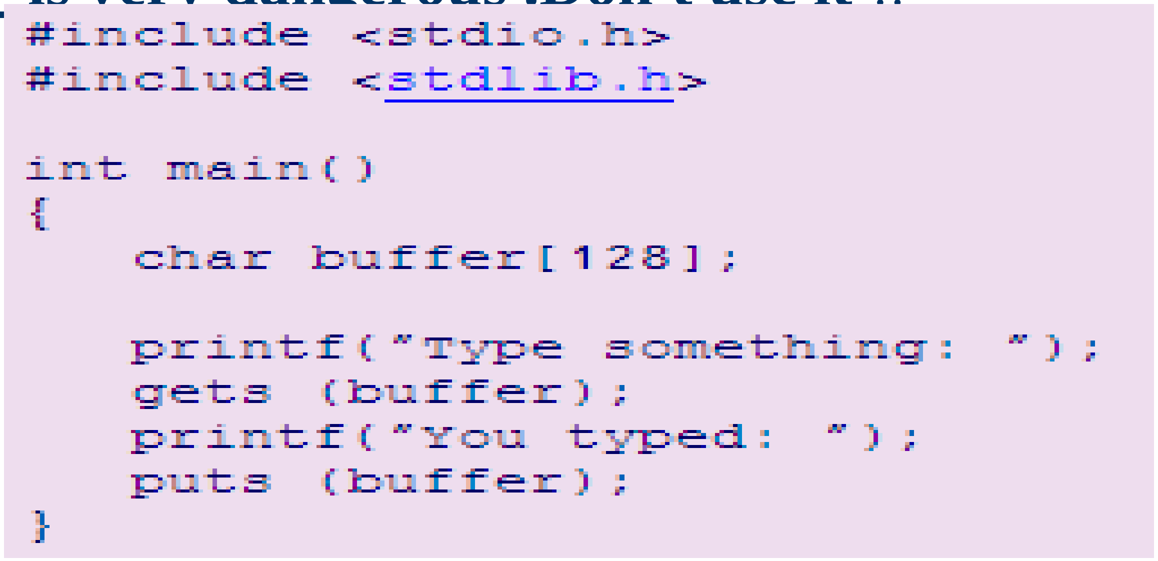


```
#include <stdio.h>

int main ()
{
  char string [] = "Hello world!";
  puts (string);
}
```

Basic I/O
gets/puts

- Read characters from stdin into the string str until a newline is read or an end-of-file is encountered.
- Newlines are not written to the string. The string is terminated with a NULL character.
- str must be large enough to hold the resulting string.
  - **gets – is very dangerous .Don't use it !!**



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

int main()
{
    char buffer[128];

    printf("Type something: ");
    gets (buffer);
    printf("You typed: ");
    puts (buffer);
}
```

Basic I/O
fgets

- Reads characters from stream fp into the string pointed to by str.
- **The integer argument:** n indicates the maximum number of characters that the buffer str can store.
- **Reading stops when:** a newline is read, end-of-file is encountered, a read error occurs, or n-1 characters were read.
- Newlines are included in the string. The string read is terminated with a 0

```
int main()
{
    char buffer[255];
    int buflen;
    char *result;

    buflen = 255;
    printf("Enter line of data:\n");
    result = fgets (buffer, buflen, stdin);
    if (result == NULL)
          printf("\nEOF or Error\n");
    else
          printf("The input was :\n%s\n", buffer);
}
```

Basic I/O
fprintf

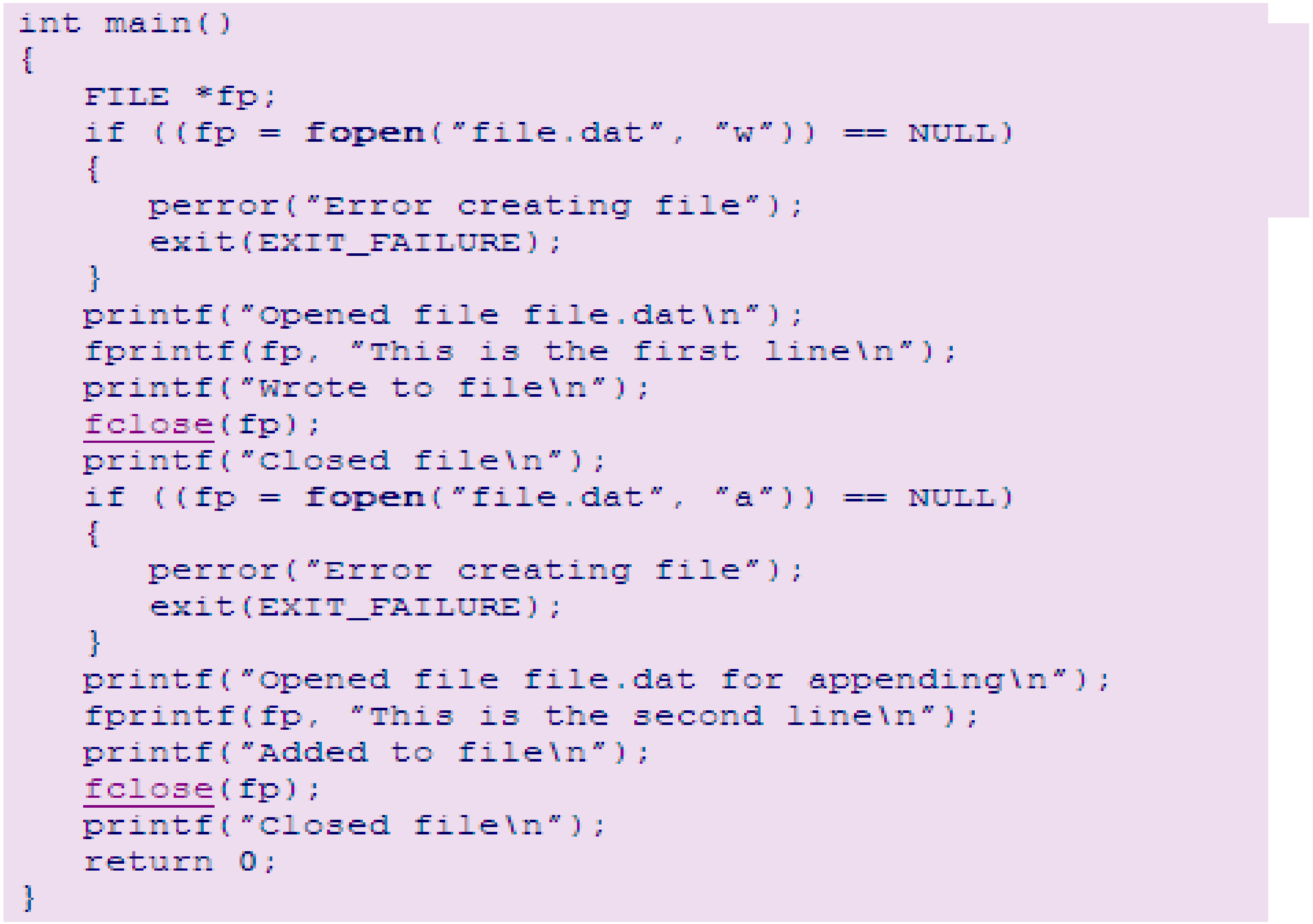
- **Prototype**
  - int **fprintf**(FILE *fp, const char *format, arg0... argn);
    int **printf**(const char *format, arg0... argn);
- **Description**
  - **fprintf** writes formatted data to the file stream *fp*.
  - **printf** writes formatted data to **stdout**.
- Arguments are interpreted according to the null-terminated *format* string.
- The ***format*** string is a sequence of characters with embedded conversion commands.
- Characters that are not part of the conversion command are output

Basic I/O
fscanf

- **Prototype**
  - int **fscanf**(FILE **fp*, const char **format*, ...);
  - int **scanf**(char **format*, ...);
- **Description**
  - **fscanf** reads characters from the input stream *fp*.
  - **scanf** reads characters from the input stream stdin.
- Characters read are converted according to the *format* string and the values created are stored through the argument pointers.
- Note that the arguments are pointers to where values will be stored.

```
printf("Enter your first and last name in
                 the form \" first last\": ");
res = fscanf(stdin, "%s %s", first, last);
```

Basic I/O Example



```
int main()
{
    FILE *fp;
    if ((fp = fopen(”file.dat”, ”w”)) == NULL)
    {
        perror(”Error creating file”);
        exit(EXIT_FAILURE);
    }
    printf(”Opened file file.dat\n”);
    fprintf(fp, ”This is the first line\n”);
    printf(”Wrote to file\n”);
    fclose(fp);
    printf(”Closed file\n”);
    if ((fp = fopen(”file.dat”, ”a”)) == NULL)
    {
        perror(”Error creating file”);
        exit(EXIT_FAILURE);
    }
    printf(”Opened file file.dat for appending\n”);
    fprintf(fp, ”This is the second line\n”);
    printf(”Added to file\n”);
    fclose(fp);
    printf(”Closed file\n”);
    return 0;
}
```

Stdlib.h

- To use all functions in this library you must:
- #include <stdlib.h>
- There are three basic categories of functions:
  - Arithmetic
  - Random Numbers
  - String Conversion
- The use of all the functions is relatively straightforward.

Stdlib.h
Arithmetic Functions

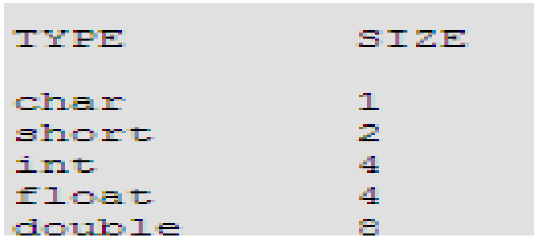
- To use all functions in this library you must:
- #include <stdlib.h>
- There are three basic categories of functions:
    - Arithmetic (abs , div …)
    - Random Numbers
    - String Conversion
- The use of all the functions is relatively straightforward.

Malloc, Sizeof, and Free

- In C++: **long *pL = new long[128];**
- malloc is most commonly used to attempt to ``grab'' a continuous portion of memory.
    - void *malloc(size_t number_of_bytes)
- **Void *** is returned the C standard states that this pointer can be converted to any type.
- **The size_t** argument type is defined in stdlib.h and is an *unsigned type*.

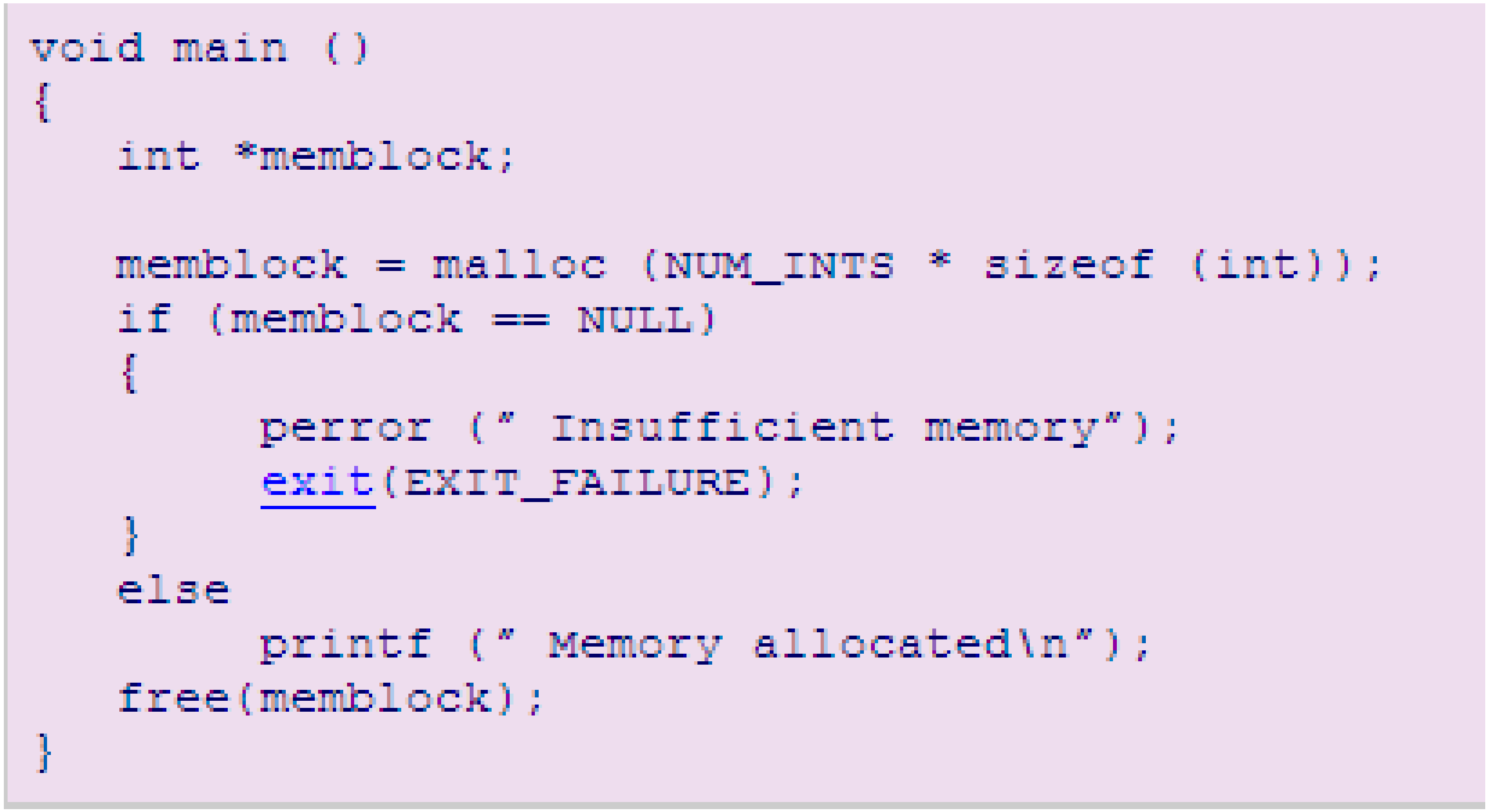
Malloc, Sizeof, and Free

- **sizeof** will return the number of bytes reserved for a variable or data type.
- int *ip;
  ip = (int *) malloc(100*sizeof(int));
- Heap (part of the adress space 4.14)



TYPE	SIZE
char	1
short	2
int	4
float	4
double	8

Malloc, Sizeof, and Free



```
void main ()
{
    int *memblock;

    memblock = malloc (NUM_INTS * sizeof (int));
    if (memblock == NULL)
    {
         perror (" Insufficient memory");
         exit(EXIT_FAILURE);
    }
    else
         printf (" Memory allocated\n");
    free(memblock);
}
```