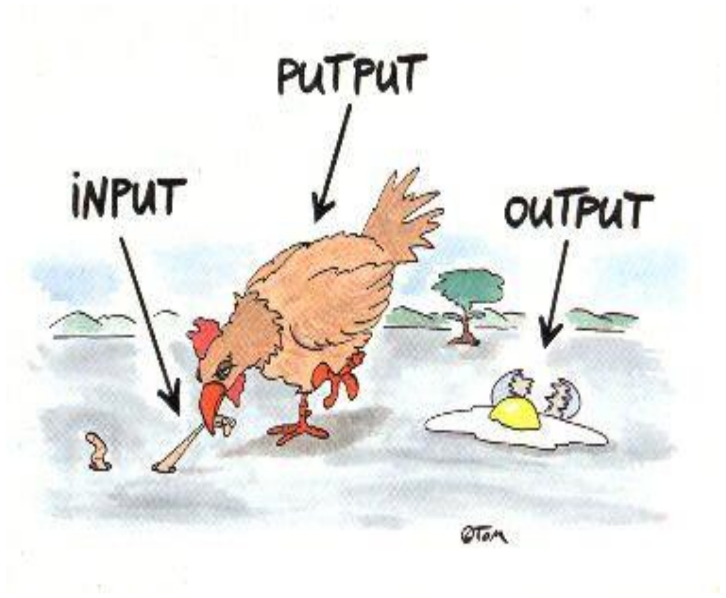
Bring ideas to life
VIA University College

Standard Input/Output

ESW1



Standard Input/Output

```
int getchar (void)
int putchar (int)
int printf (char* format, arg1, arg2, …)
int sprintf (char* string, char* format, arg1, arg2, …)

int scanf (char* format, arg1, arg2, …)
int sscanf (char* string, char* format, arg1, arg2, …)
```

File Input/Output

```
FILE *fp
FILE *fopen (char* name, char* mode)
int feof(FILE *fp)
int fclose(FILE *fp)

int getc (FILE *fp)
int putc (int c, FILE *fp)
int fprintf (FILE *fp, char* format, arg1, arg2, …)
int fscanf (FILE *fp, char* format, arg1, arg2, …)
```

File Open Modes

```
FILE *fopen (char* name, char* mode)
```

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

File Input

```
#include <stdio.h>
main() {

   FILE *fp;
   char buff[255];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);
}
```

File Output

```
#include <stdio.h>

main() {
   FILE *fp;

   fp = fopen("/tmp/test.txt", "w+");
   fprintf(fp, "This is testing for fprintf...\n");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

Return Codes

Remember you **MUST** check return codes from functions E.g.

```
FILE *fopen(const char *filename, const char *mode)
```

This function returns a FILE pointer. Otherwise, NULL is returned and the global variable `errno` is set to indicate the error

```
int fscanf(FILE *stream, const char *format, ...)
```

This function returns the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure

Remember C has no Exception handler like Java's `try {…} catch (….){…}`

Return Codes

Remember you **MUST** check return codes from functions. E.g.

```
…
fp = fopen("/tmp/test.txt", "r");

if (fp == NULL) {
   // File is not opened or not found!!!
   exit(EXIT_FAILURE);
}

// File is opened correct - ready for use!
if (fscanf(fp, "%s", buff) == 1) {
   // You can use buf!
```

Avoid Security Warnings From Visual C++ Compiler

The safe way: Use Security-Enhanced versions

https://docs.microsoft.com/en-us/cpp/c-runtime-library/security-enhanced-versions-of-crt-functions?view=vs-2017

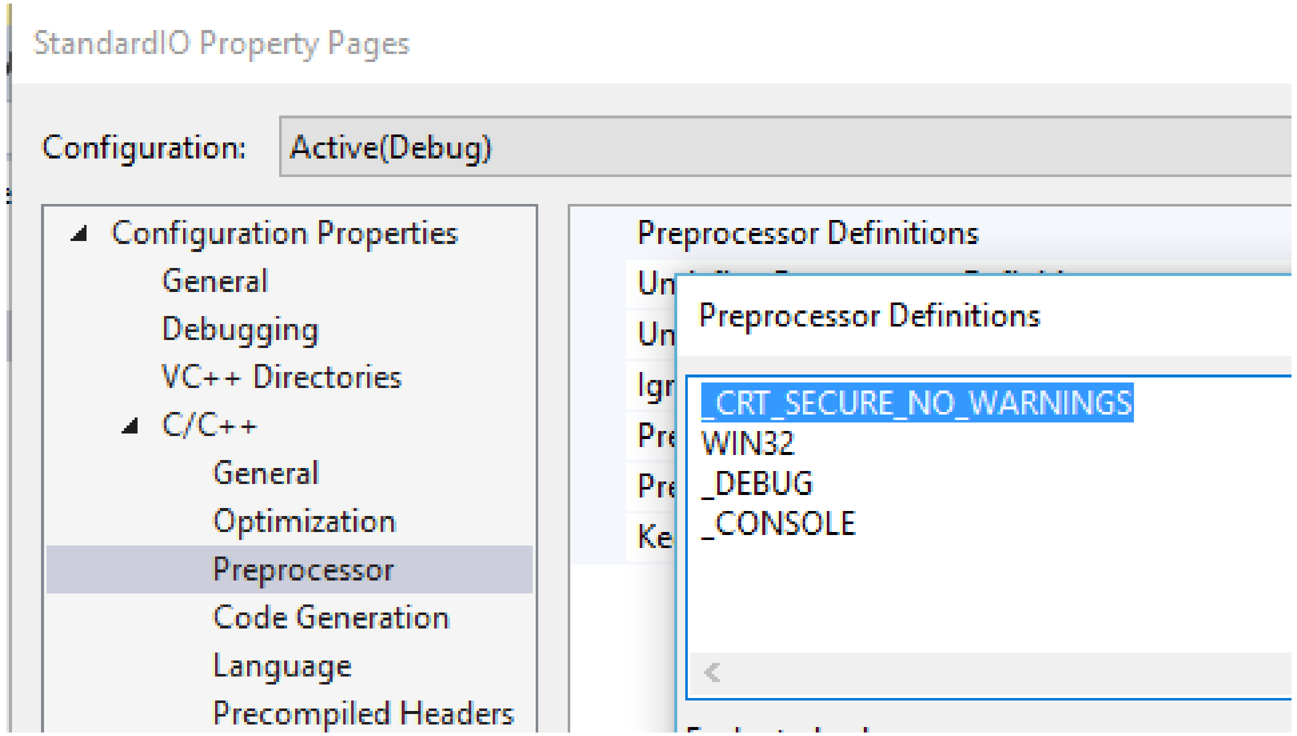
```
int printf_s(const char *restrict format, ...);
int sprintf_s(char *restrict buffer, rsize_t bufsz, const char *restrict format, ...);
int scanf_s(const char *restrict format, ...);
int sscanf_s(const char *restrict buffer, const char *restrict format, ...);

errno_t fopen_s(FILE *restrict *restrict streamptr, const char *restrict filename,
                const char *restrict mode);
int fprintf_s(FILE *restrict stream, const char *restrict format, ...);
int fscanf_s(FILE *restrict stream, const char *restrict format, ...);
```

Avoid Security Warnings From Visual C++ Compiler

Alternative solution:

Add *_CRT_SECURE_NO_WARNINGS* to your projects *Preprocessor Definitions*



Configuration: Active(Debug)

Preprocessor Definitions

```
_CRT_SECURE_NO_WARNINGS
WIN32
_DEBUG
_CONSOLE
```