

```
pod1-1:Lexpandu.edu ~ % more main1.c
// copy ascii file: file names given as command-line arguments
// similar to: cp <filename1> <filename2>

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

int main(int argc, char **argv)
{
FILE *fp1, *fp2;
char c;

// open file to be copied from
fp1 = fopen(argv[1], "r");
if(fp1 == NULL) {
    fprintf(stderr, "file %s does not exist\n", argv[1]);
    exit(1);
}

// create file to be copied to
fp2 = fopen(argv[2], "w");
if(fp2 == NULL) {
    fprintf(stderr, "file %s could not be created\n", argv[2]);
    exit(1);
}

while((c = fgetc(fp1)) != EOF)
    fputc(c, fp2);

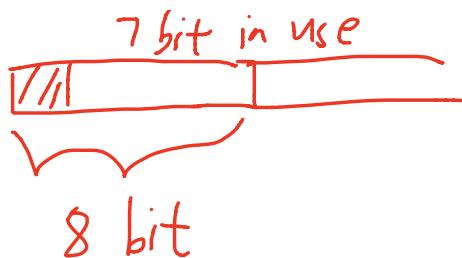
return 0;
}
pod1-1 40 %
```

Annotations in red:

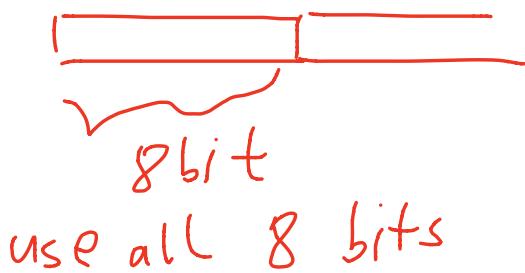
- Annotations for the first section:
 - Line 1: "read"
 - Line 6: "if(fp1 == NULL)" with a red bracket spanning the entire line.
 - Line 10: "write"
- Annotations for the second section:
 - Line 14: "read byte one by one"
 - Line 16: "fputc(c, fp2);"
 - Line 18: "write byte one by one"
 - Line 20: "'-' not ASCII character"

Can copy binary file, ASCII file;

ASCII file



binary file



use all 8 bits

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

int main(int argc, char **argv)
{
FILE *fp;
char c;
    fp = fopen(argv[1], "r");
    if(fp == NULL) {
        fprintf(stderr, "file %s does not exist\n", argv[1]);
        exit(1);
    }
    while((c = fgetc(fp)) != EOF)
        fprintf(stdout, "%c", c);

    return 0;
}
```

"main.c" 23L, 388C

→ mycat.c

[or] do binary file
too

A screenshot of a terminal window titled "pod1.Lex.purdue.edu - SecureCRT". The window displays the following C code:

```
// type casting
#include <stdio.h>
main() {
    float y;
    int x;
    y = 10.8;
    x = y;
    printf(stdout, "%d\n", x);
}
```

The terminal output shows the command "gcc main2a.c" being run, followed by the output "10".

explicit type casting
(int *) malloc()
char *

decimal part
not round it up

depends on compiler


```

// type casting
#include <stdio.h>
int main0 {
    unsigned int x;
    float y;
    x = 4000000000;
    y = x;
    printf(stdout,"%d\n",x);
    printf(stdout,"%f\n",y);
}

```

change to
%u

main2.c" 16L, 164C written
pod1-1 16 % !g
gcc main2.c
pod1-1 17 % a.out
-294967296
4000000000.000000
pod1-1 18 %
Ready

```

// type casting
#include <stdio.h>
int main0 {
    unsigned int x;
    float y;
    x = 4000000000;
    y = x;
    printf(stdout,"%u\n",x);
    printf(stdout,"%f\n",y);
}

```

unsigned
} 2³²-1
%u

main2.c" 16L, 164C written
pod1-1 19 % !g
gcc main2.c
pod1-1 20 % a.out
4000000000
4000000000.000000
pod1-1 21 %
Ready

Two screenshots of a terminal window showing the output of a C program. The program performs type casting from a float to a double and back.

Screenshot 1:

```
// type casting
#include <stdio.h>
int main() {
    float x;
    double y;
    x = -10.3;
    y = x;
    printf(stdout, "%f %lf\n", x, y);
}
```

The output shows two lines: -10.3 and -10.3 . Handwritten notes above the first line say -10.3 and -10.3 .

Screenshot 2:

```
float x;
double y;
x = -10.123456789;
y = x;
fprintf(stdout, "%f %lf\n", x, y);
```

The output shows: "main2b.c" 15L, 139C written
pod1-1 26 % !gc
gcc main2b.c
pod1-1 27 % a.out
-10.123457 -10.123457
pod1-1 28 %

A red arrow points from the handwritten note -10.3 to the output line -10.3 . Another red arrow points from the handwritten note -10.3 to the output line -10.123457 , with the handwritten note $lost .89$ written next to it.

Screenshot 3:

```
// type casting
#include <stdio.h>
int main() {
    float x;
    double y;
    x = -10.123456789;
    y = x;
    printf(stdout, "%9f %9lf\n", x, y);
}
```

The output shows: "main2b.c" 15L, 143C written
pod1-1 29 % !g
gcc main2b.c
pod1-1 30 % a.out
-10.123456955 -10.123456955
pod1-1 31 %

Handwritten annotations explain the behavior:
 - "double y is not overflow, but" (near the top of the output)
 - "y is assigned by x, x is overflowed" (near the bottom of the output)
 - "last three digits garbage" (near the bottom of the output)
 - "Both" (at the bottom right)

```
// type casting
#include <stdio.h>
int main() {
    float x;
    double y;
    x = -10.123456789;
    y = -10.123456789;
    //y = x;
    fprintf(stdout, "%f %.9lf\n", x, y);
}

```

y is ok,
Because
we directly

```
float x;
double y;
x = -10.123456789;
y = -10.123456789;
// y = x;
fprintf(stdout, "%f %.9lf\n", x, y);
}

main2b.c 16L, 167C written
pod1-1 32 % !g
gcc main2b.c
pod1-1 33 % a.out
-10.123456955 -10.123456789
pod1-1 34 %
```

assign this big value to y, which
is not overflow.

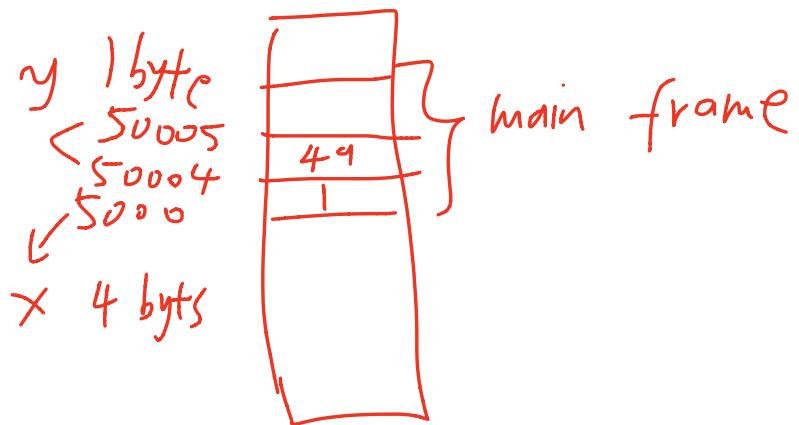
x is still overflow.

```
// type casting
#include <stdio.h>
void main() {
    int x;
    char y;
    x = 1;
    y = '1'; → 49
    x = x + y; → 50
    fprintf(stdout, "%d\n", x);
}

```

point

```
main2c.c 17L, 134C written
pod1-1 39 % gcc main2c
main2a.c main2b.c main2.c main2c.c main2d.c main2e.c
pod1-1 39 % gcc main2c.c
pod1-1 40 % a.out
50
pod1-1 41 %
```



after $x = x + y;$

y promote from char to int
1 byte 4 bytes

Compiler generate a new place
which is 4 bytes, substitute
the old address of y .

```
x = -1;
y = x;
fprintf(stdout,"%d %u\n", x, y);
}

~main2e.c" 15L, 129C written
pod1-1 52 % !g
gcc main2e.c
main2e.c:5:1: warning: return type defaults to 'int' [-wimplicit]
 main() {
 ^
pod1-1 53 % a.out
-1 4294967295
pod1-1 54 %
```

maximum number