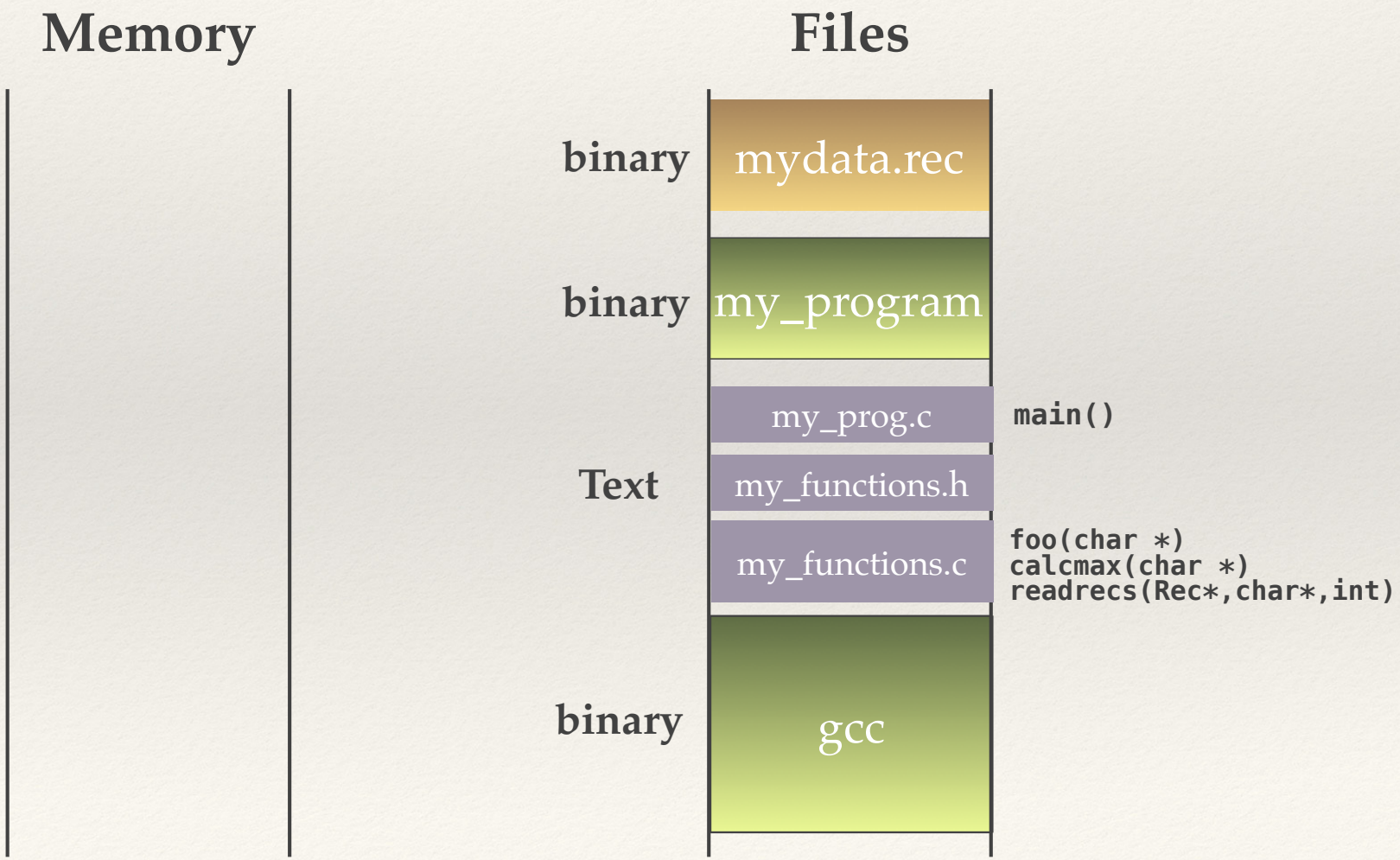
Yet more about pointers…

Interaction between Program, Functions and Memory

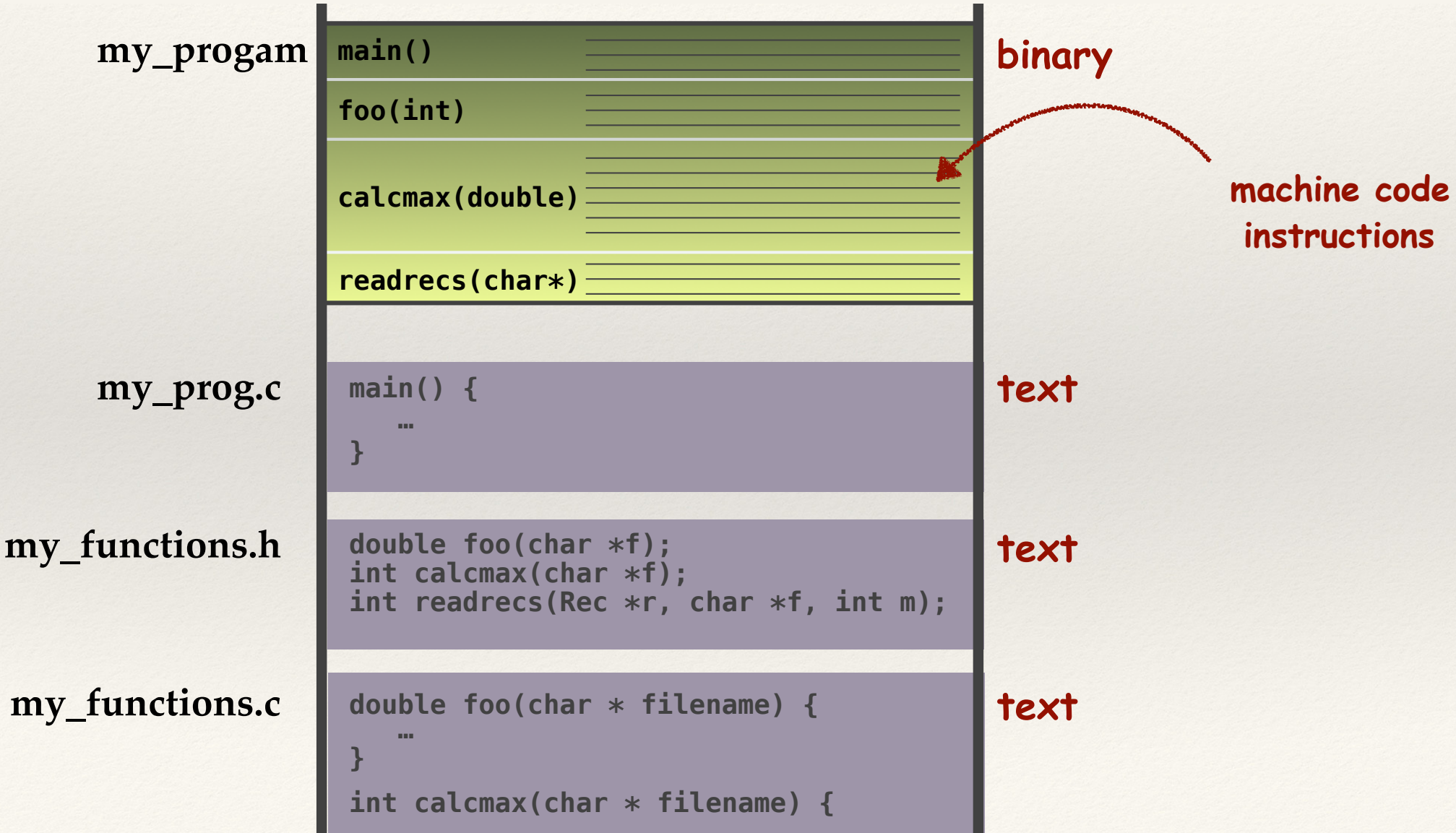
leading to function pointers

Back to the Compiler



Back to the Compiler

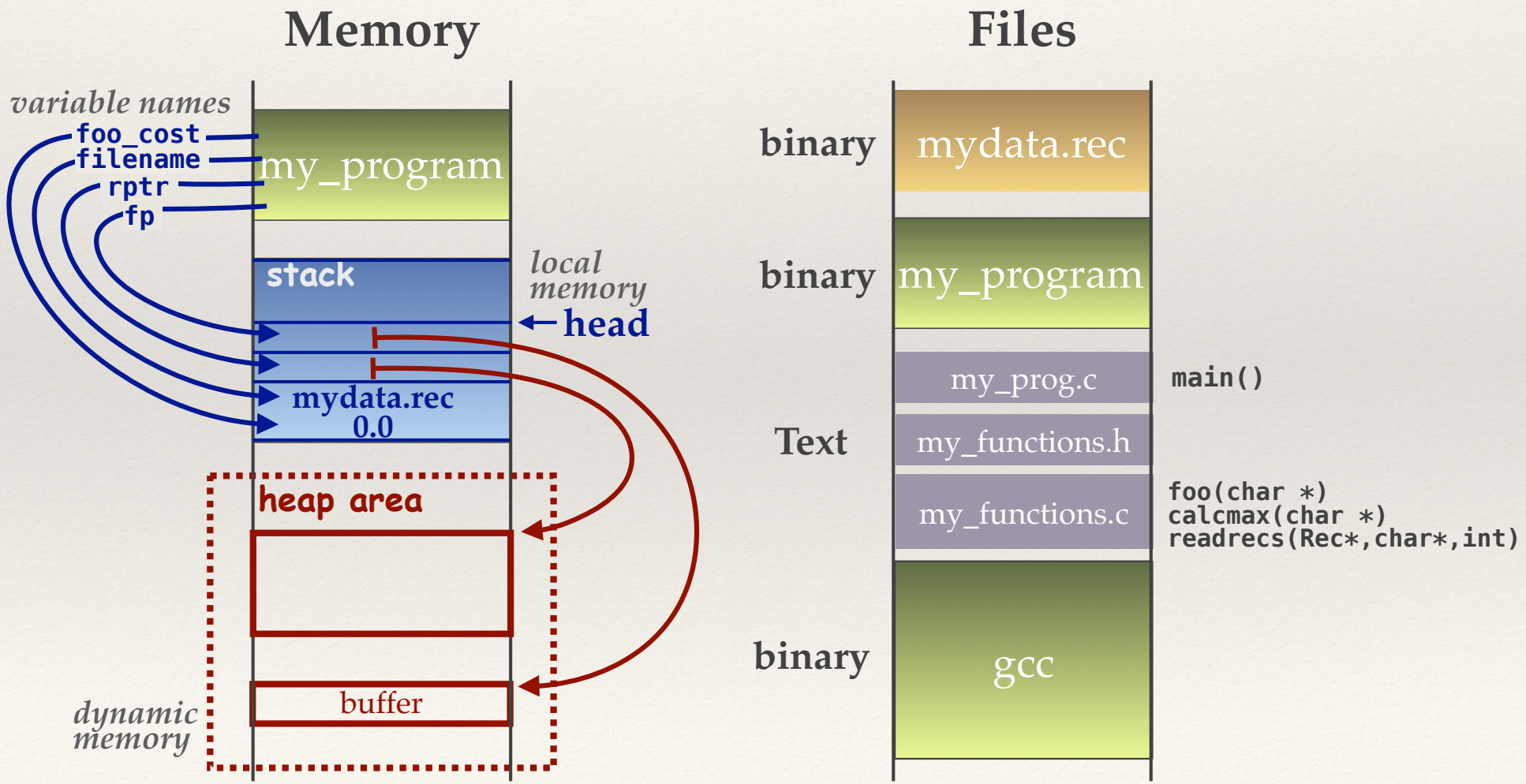
Back to the Compiler



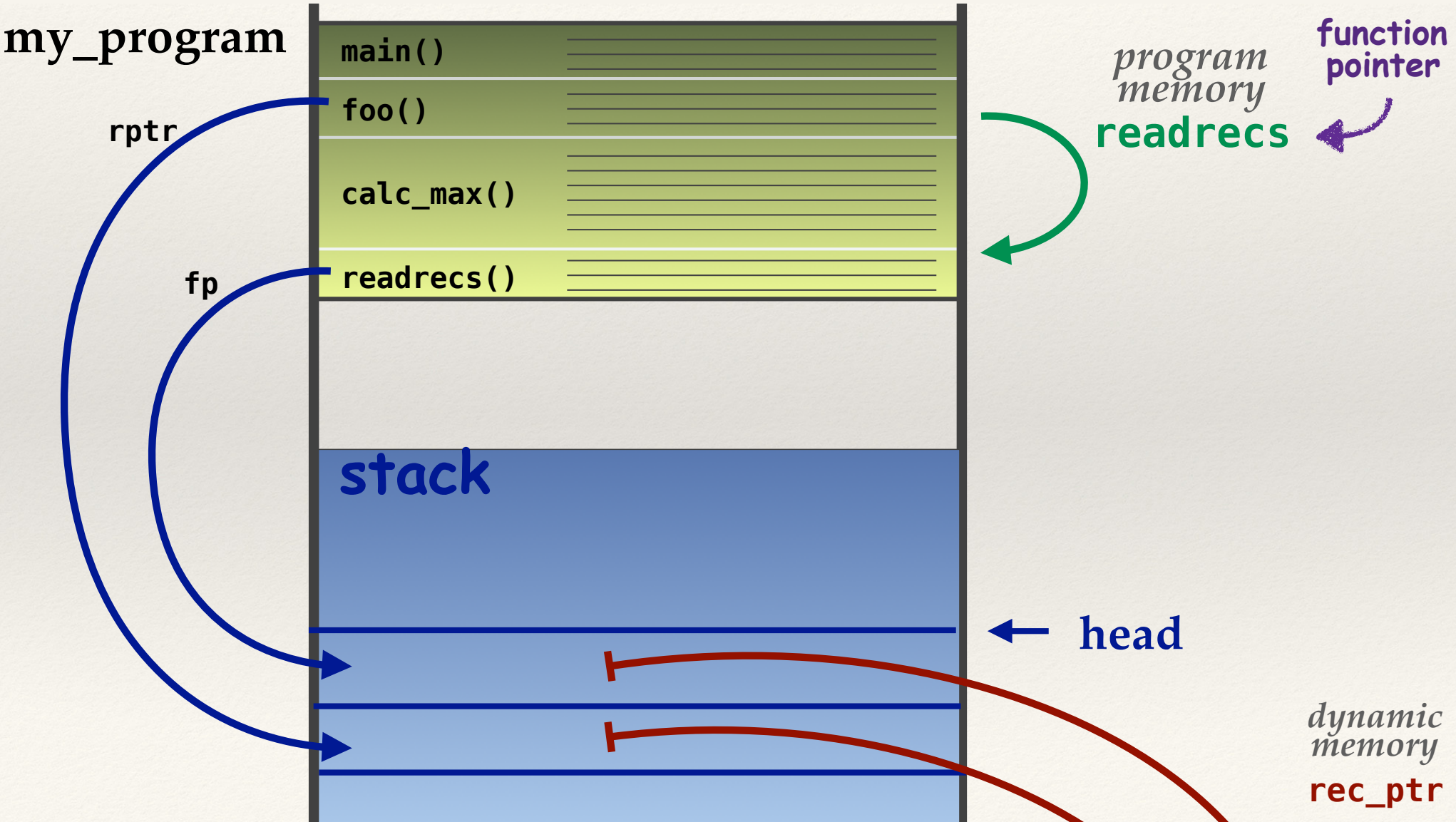
Running my_prog

```
int readrecs (Rec * rptr, char * fname, int rmax) {
    FILE * fp = fopen(fname, "rb");
    rcount = fread(rptr, sizeof(Rec), rmax, fp);
    fclose(fp)
    return rcount;
}
```

linux: ./my_program mydata



Running my_prog



Yet more about pointers…

Function Pointers

What is a Function Pointer?

- A pointer variable can point to almost anything - int's, strings, struct's and … functions!
- We declare it like any other pointer - we declare that it is a **pointer** and we give the **type** of *thing* that it is pointing to.
  - In this case we are pointing to code not data/variables.
  - Typically we are pointing to the start of the executable code.

Declaring a Function Pointer

Let's declare a fn pointer and set it to

void `print_me(int x)`

- declaring

```
void (*fn_ptr)(int);
```

We need the `()` around `fn_ptr` or it would be the declaration for a function that returned a `void` pointer.

- setting

```
fn_ptr = &print_me;
```

or

```
fn_ptr = print_me;
```

Function Pointer Example Code

print_me.c

```
#include <stdio.h>

void print_me ( int n )
{
   printf (“n = %d\n", n);
}

void print_2x ( int n )
{
   printf (“2n = %d\n", 2 * n);
}
```

fpointer1.c

```
#include <stdio.h>
void print_me(int);
void print_2x(int);


int main ()
{
   void (*fn_ptr)(int);

   fn_ptr = &print_me;
   (*fn_ptr) ( 10 );

   fn_ptr = &print_2x;
   (*fn_ptr) ( 10 );

   return 0;
}
```

Function Pointer Example Code

alternate function pointer usage

print_me.c

```
#include <stdio.h>

void print_me ( int n )
{
   printf (“n = %d\n", n);
}

void print_2x ( int n )
{
   printf (“2n = %d\n", 2 * n);
}
```

fpointer1.c

```
#include <stdio.h>
void print_me(int);
void print_2x(int);


int main ()
{
   void (*fn_ptr)(int);

   fn_ptr = print_me;
   fn_ptr ( 10 );

   fn_ptr = print_2x;
   fn_ptr ( 10 );

   return 0;
}
```

Function Pointer Example Code

alternate function pointer usage

print_me.c

```
#include <stdio.h>

void print_me ( int n )
{
   printf ("n = %d\n", n);
}

void pr
{
   prin
}
```

fpointer1.c

```
#include <stdio.h>
void print_me(int);
void print_2x(int);

nt);

e;

   fn_ptr = print_2x;
   fn_ptr ( 10 );

   return 0;
}
```

```
$ gcc -c print_me.c
$ gcc fpointer1.c print_me.o -o fpointer1
$ ./fpointer1
n = 10
2n = 20
```

More about Function Pointers

- We do not allocate/free memory for function pointers
  - function pointers point to executable code
  - i.e. code already exists in memory
- We can have an array of function pointers
  - just like we can have an array of pointers to data/memory
- A function pointer
  - can be passed as an argument
  - can be returned from a function

An Array of Function Pointers

- An array of function pointers can be used instead of a `switch` statement.

```
void ( *func_ptr_array[] ) (char *)
    = { func1, func2, func3 };
```

- This declares an array of function pointers to functions named `func1()`, `func2()`, and `func3()`.

```c
#include <stdio.h>
#include <string.h>
#define TRUE  1
#define FALSE 0

void func1 ( char * );
void func2 ( char * );
void func3 ( char * );

int main () {

   void ( *func_ptr_array[] ) ( char * ) = { func1, func2, func3 };

   char string[]="Number 1: this  is  a  test. And, this is another test!";
   int fn_num = 0, quit = FALSE;

   printf ("Before: %s (%lu)\n”, string, strlen(string));
   while ( !quit ) {
      printf ( "Enter the function number (1, 2, or 3): ");
      scanf ( "%d", &flag );
      if ( 0 < fn_num && fn_num < 4 ) {
         (*func_ptr_array[flag-1]) (string);
         printf ("After func%d: %s (%lu)\n”, flag,string, strlen(string));
      } else {
         quit = TRUE;
      }
   }
   return(0);
}
```

```
#include <string.h>

void func1 ( char *string )
{
   int size = strlen(string);
   int i,j;

   for ( i=1; i<size; i++ ) {
      if ( string[i-1] == ' ' && string[i] == ' ' ) {
         for ( j=i; j<size-1; j++ ) {
            string[j] = string[j+1];
         }
         string[j] = '\0';
      }
   }
}
```

removes extra spaces

```
#include <string.h>

void func2 ( char *string )
{
   int size = strlen(string);
   int i,j;

   for ( i=0; i<size; i++ ) {
      if ( string[i] == ',' || string[i] == ';' || string[i] == ‘:'){
         for ( j=i; j<size-1; j++ ) {
            string[j] = string[j+1];
         }
         string[j] = '\0';
      }
   }
}
```

removes some punctuation
(comma, semi-colon, colon)

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

void func3 ( char *string )
{
   int size = strlen(string);
   int i;

   for ( i=0; i<size; i++ ) {
      string[i] = tolower( string[i] );
   }
}
```

converts string to lower case

```
#include <stdio.h>
#include <string.h>
#define TRUE  1
#define FALSE 0

void func1 ( char * );
void func2 ( char * );
void func3 ( char * );

int main () {

   void ( *func_ptr_array[] ) ( char * ) = { func1, func2, func3 };

   char string[]="Number 1: this  is  a  test. And, this is another test!";
   int fn_num = 0, quit = FALSE;

   printf ("Before: %s (%lu)\n", string, strlen(string));
   while ( !quit ) {
      printf ( "Enter the function number (1, 2, or 3): ");
      scanf ( "%d", & fn_num );
      if ( 0 < fn_num && fn_num < 4 ) {
         fn_num--;
         (*func_ptr_array[fn_num]) (string);
         printf ("After func%d: %s (%lu)\n", flag, string, strlen(string));
      } else {
         quit = TRUE;
      }
   }
   return(0);
}
```

```
$ gcc -Wall -ansi -c func*.c
$ gcc -Wall -ansi func*.o fptrArray.c -o fptrArray
$ ./fptrArray
Before: Number 1: this  is  a  test. And, this is another test! (55)
Enter the function number (1, 2, or 3): 1
After func1: Number 1: this is a test. And, this is another test! (52)
Enter the function number (1, 2, or 3): 2
After func2: Number 1 this is a test. And this is another test! (50)
Enter the function number (1, 2, or 3): 3
After func3: number 1 this is a test. and this is another test! (50)
Enter the function number (1, 2, or 3): 0
```