Functions & Pointers

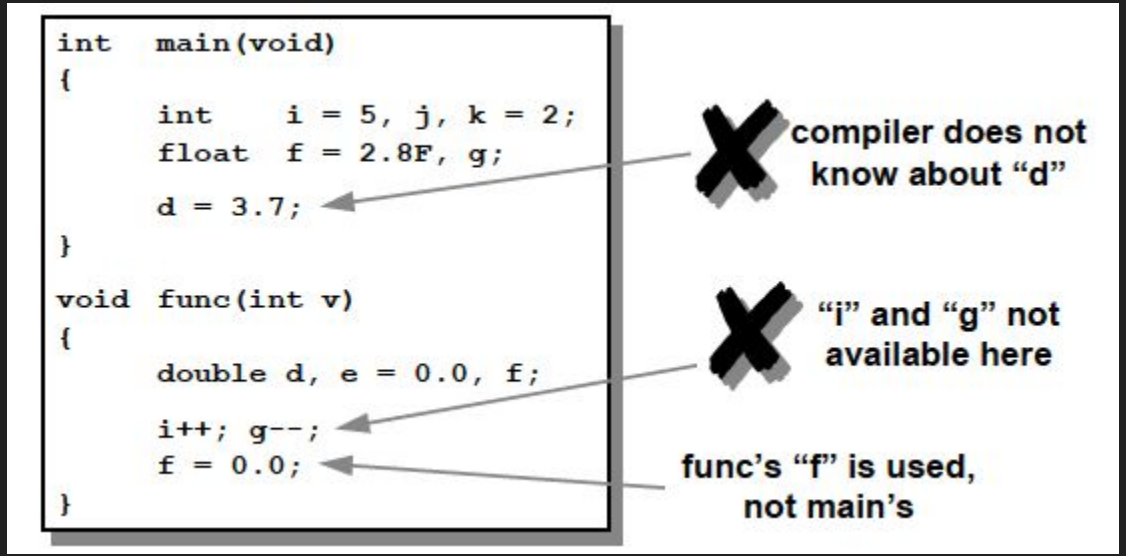
Derrick

Prototypes

- `int putchar(int);` is known as a prototype.
- The prototype provides the compiler with important information about the return type and parameters.
- When calling a `Standard Library` function, `#include` the file specified in the man page(s) - this file will contain the prototype
- When calling one of your own functions, write a prototype by hand.
- The function prototype may optionally include variable names. (which are ignored) - `int putchar(int c);`
- Notice that `void` must be used to indicate the absence of a type.

Scope

- C is a block structured language, variables may only be used in functions declaring them.



Pointers

What?

“A Variable that holds a reference”

(memory address)

- *Already been using pointers (in Java)*

Why?

- *More **efficient** than copying everything.*

How?

- Pointers are declared by using “ `*` ”.
- Declare an integer: `int i;`
- Declare a pointer to an integer: `int *p;`

Example

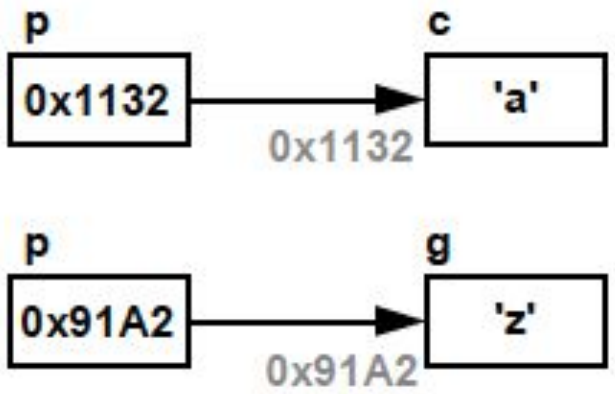
```
int         *pi;            /* pi is a pointer to an int */

long int    *p;             /* p is a pointer to a long int */

float*      pf;             /* pf is a pointer to a float */

char        c, d, *pc;      /* c and d are a char
                               pc is a pointer to char */

double*     pd, e, f;       /* pd is pointer to a double
                               e and f are double */

char*       start;          /* start is a pointer to a char */

char*       end;            /* end is a pointer to a char */
```

Address? - The “ `&` ” Operator

- The “ & ” operator (“`address of`” operator) , generates the address of a variable.
- All variables have addresses except register variables(CPU).

Example

```
char  g = 'z';

int   main(void)
{
      char    c = 'a';
      char    *p;

      p = &c;
      p = &g;

      return 0;
}
```



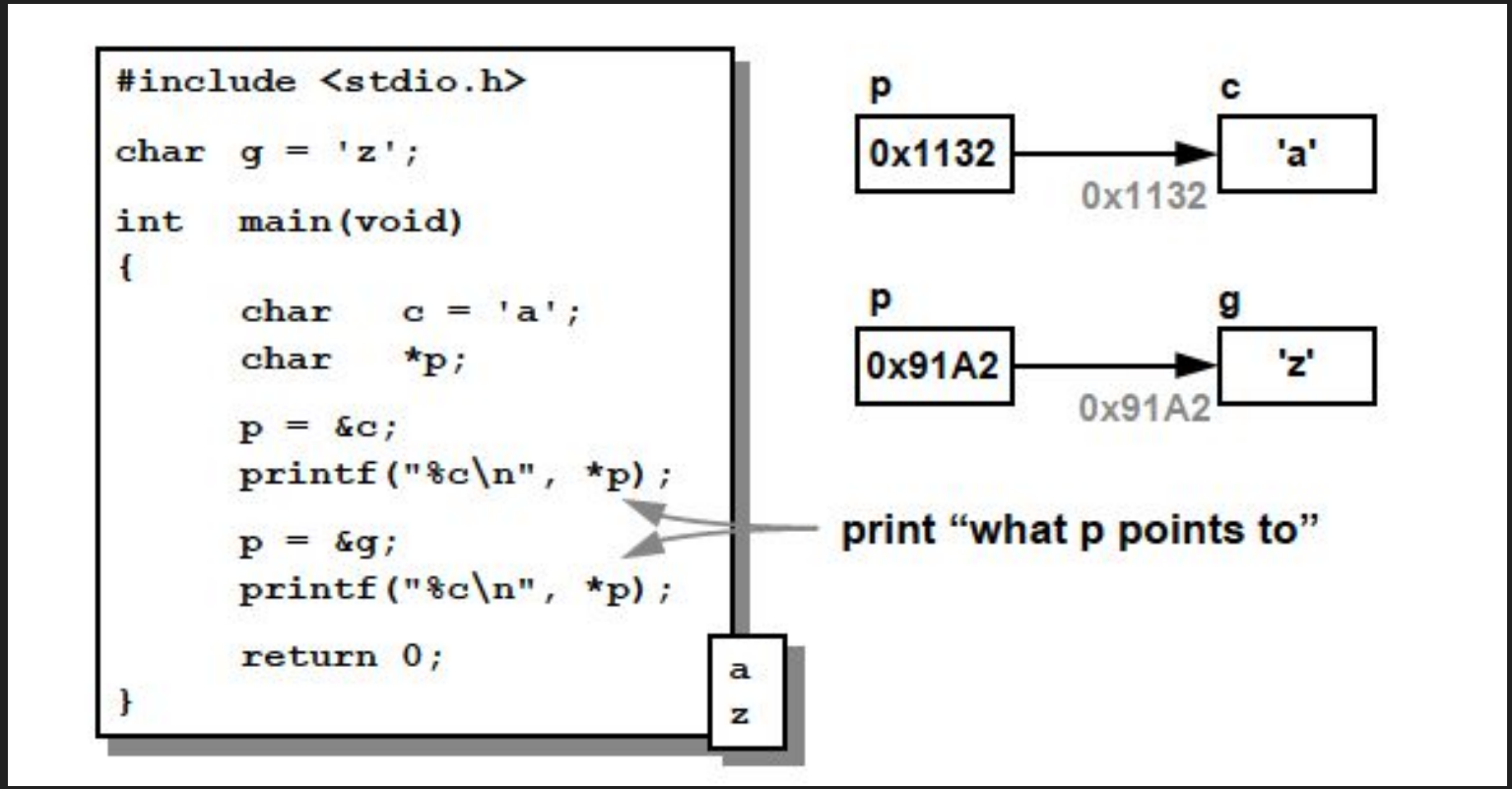
Rules

- Pointers only point to variables of the same type as the pointer has been declared to point to.
  - A pointer to an `int` only point to an `int`.

The “ * ” Operator - Dereference

- The “ * ”, “*dereferencing operator*”, accesses the value that the pointer is pointing to.
  - `int *p;`
  - `p vs *p;`
- The value of a pointer may be seen by calling `printf` with `%p` format specifier.
- `printf(“The address: %p\n”, p);`

Example


#include <stdio.h>
char g = 'z';
int main(void)
{
char c = 'a';
char *p;
p = &c;
printf("%c\n", *p);
p = &g;
printf("%c\n", *p);
return 0;
}
a
z
p
c
0x1132
'a'
0x1132
p
g
0x91A2
'z'
0x91A2
print “what p points to”

NULL

- A special invalid pointer value exists `#defined` in various header files.
  - `stdio.h , and a few other of Standard headers just in case.`
  - `#define NULL 0`

- Mostly defined as zero, but you should never assume this. On some machines zero is a legal pointer and so `NULL` will be defined as something else.

Example

There is a great deal of difference between:

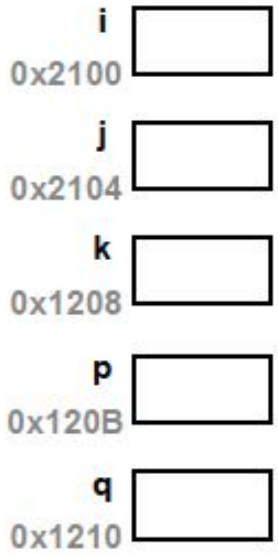
```
int   i = 10, j = 14;
int   *p = &i;
int   *q = &j;

*p = *q;
```

and:

```
int   i = 10, j = 14;
int   *p = &i;
int   *q = &j;

p = q;
```

Fill in the Gaps

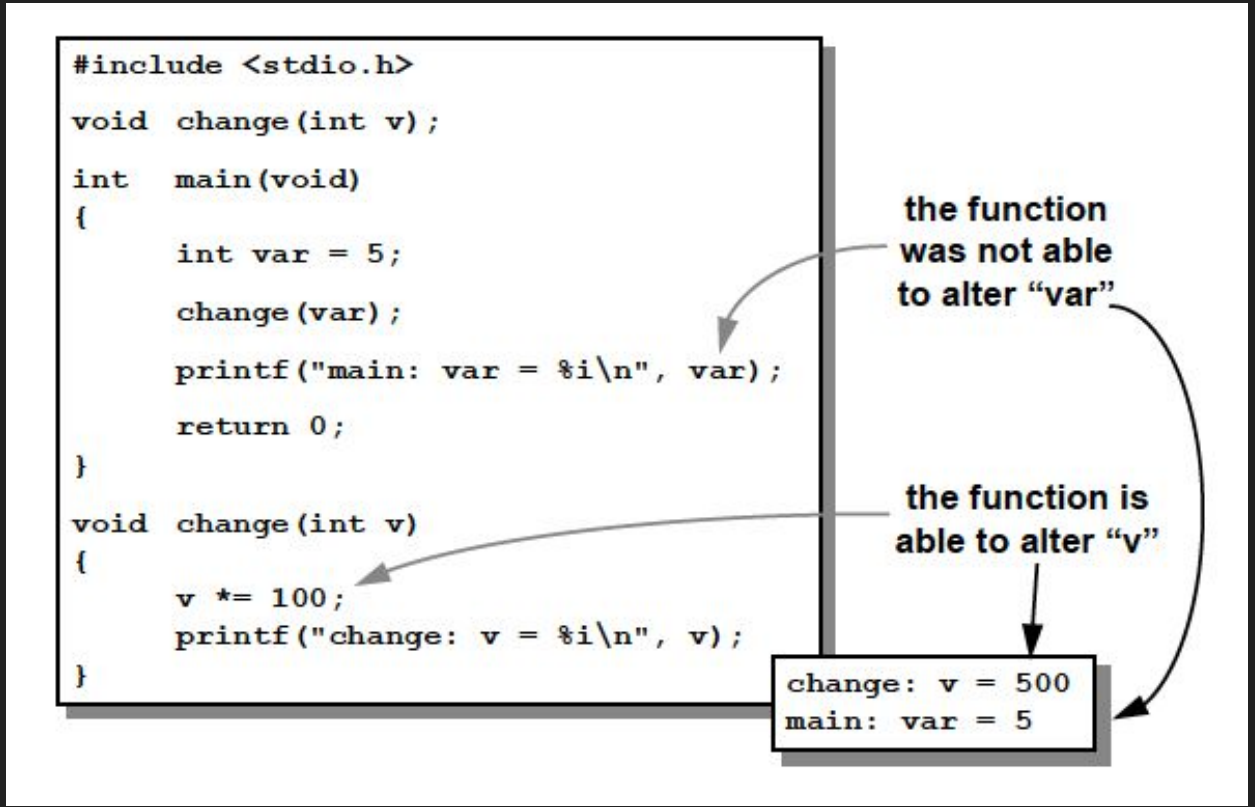
```
int   main(void)
{
      int  i = 10, j = 14, k;
      int  *p = &i;
      int  *q = &j;

      *p += 1;

      p = &k;

      *p = *q;

      p = q;

      *p = *q;

      return 0;
}
```



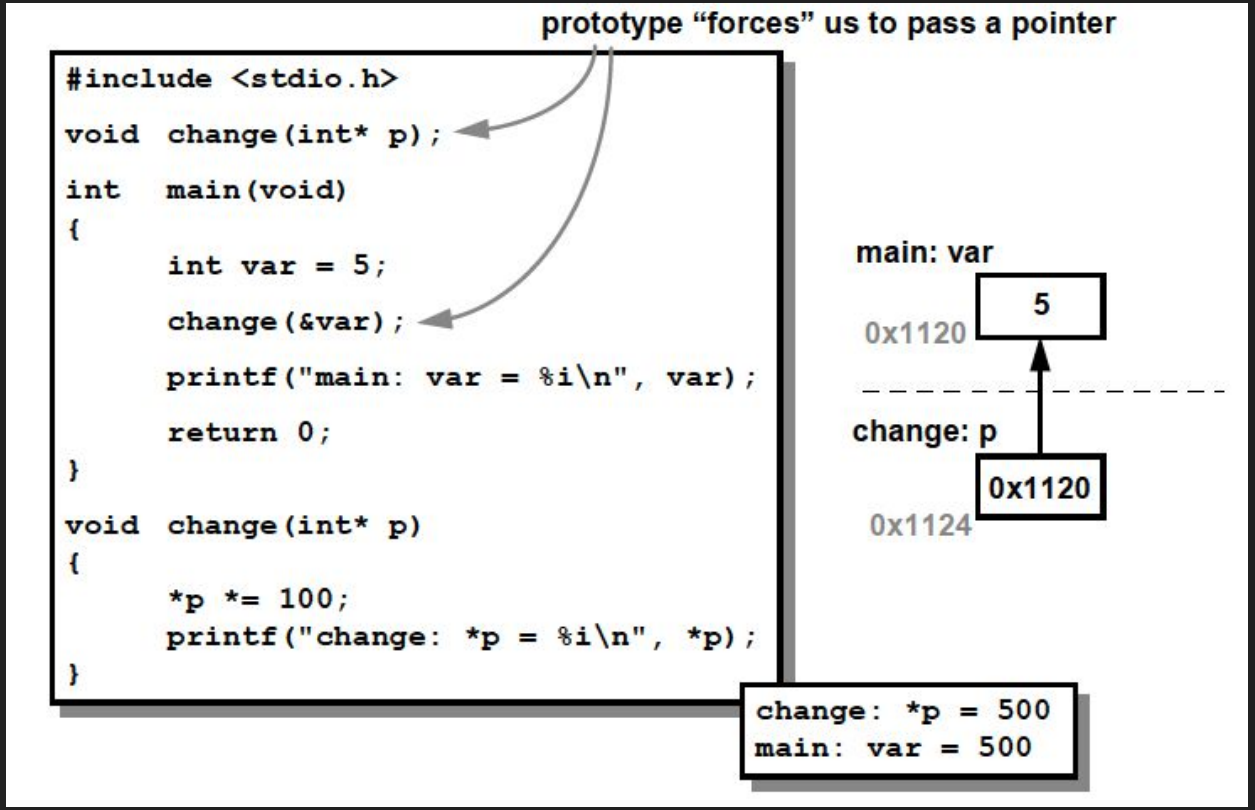
Reminder

- *Call by Value*
- *Call by Reference*

Example (call by value)

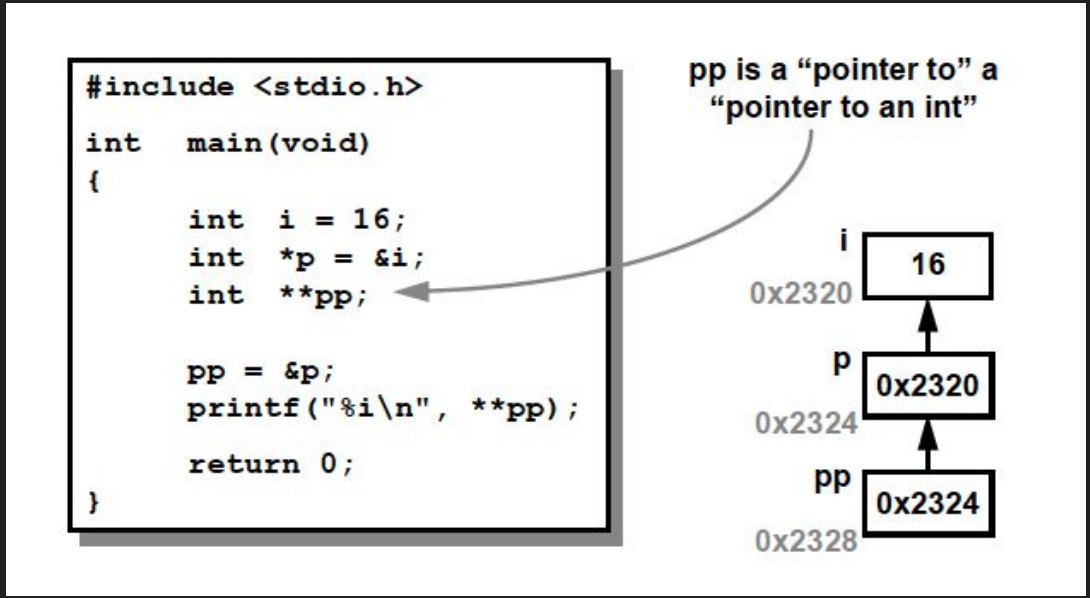


Example (call by reference)



Pointers to Pointers

- C allows pointers to any type.
- It is possible to declare a pointer to a pointer



Review

```
int   main(void)
{
      int  i = 10, j = 7, k;
      int  *p = &i;
      int  *q = &j;
      int  *pp = &p;

      **pp += 1;

      *pp = &k;

      **pp = *q;

      i = *q***pp;

      i = *q/**pp;    /* headache? */;

      return 0;
}
```