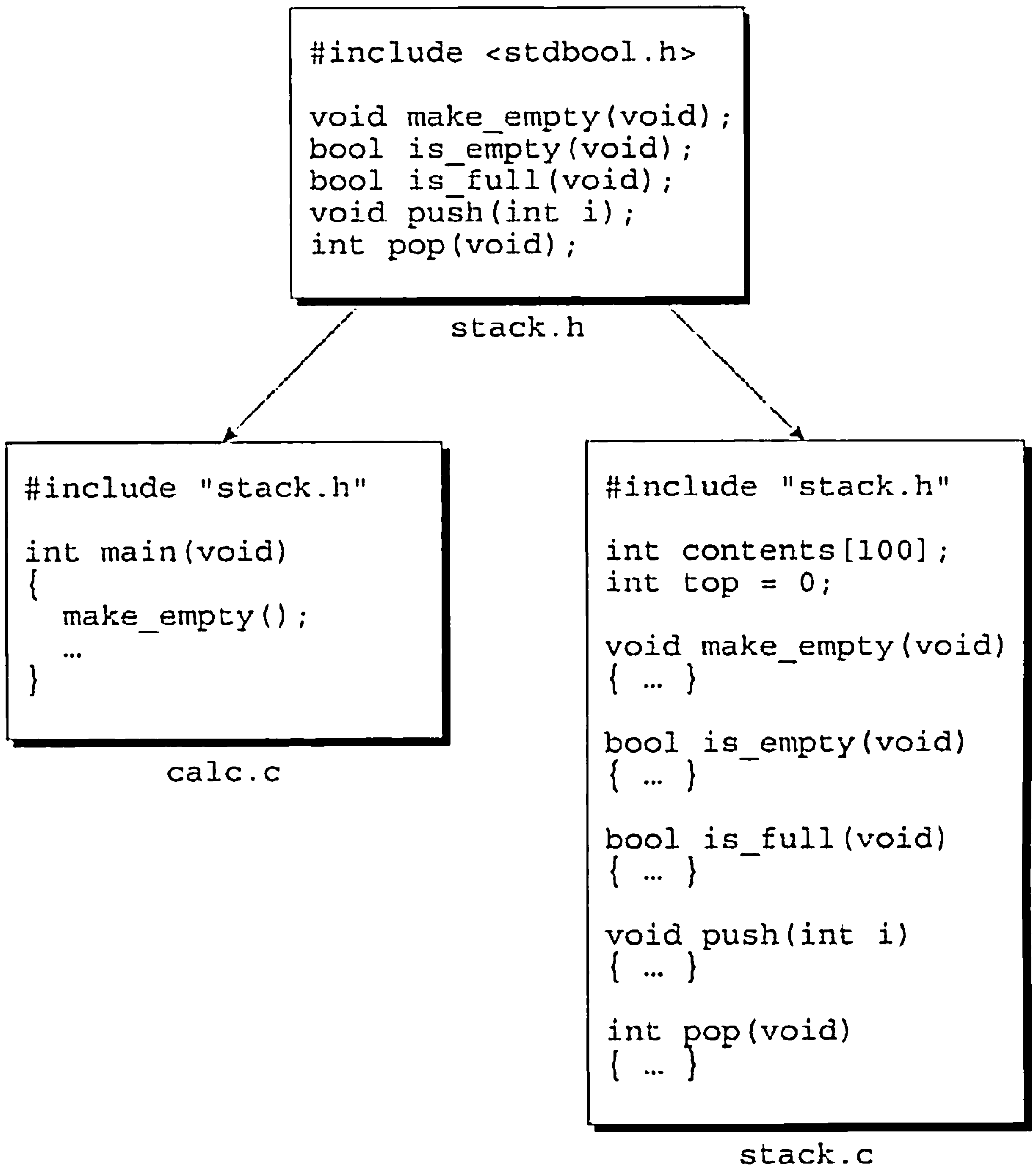

```
#include <stdbool.h>

void make_empty(void);
bool is_empty(void);
bool is_full(void);
void push(int i);
int pop(void);
```

```
#include "stack.h"

int main(void)
{
  make_empty();
  ...
}
```

```
#include "stack.h"

int contents[100];
int top = 0;

void make_empty(void)
{ ... }

bool is_empty(void)
{ ... }

bool is_full(void)
{ ... }

void push(int i)
{ ... }

int pop(void)
{ ... }
```

ular person. Team members can then work largely independently of one another.

- *Reusability.* Any module that provides services is potentially reusable in other programs. Our stack module, for example, is reusable. Since it's often hard to anticipate the future uses of a module, it's a good idea to design modules for reusability.
- *Maintainability.* A small bug will usually affect only a single module implementation, making the bug easier to locate and fix. Once the bug has been fixed, rebuilding the program requires only a recompilation of the module implementation (followed by linking the entire program). On a larger scale, we could replace an entire module implementation, perhaps to improve performance or when transporting the program to a different platform.

Although all these advantages are important, maintainability is the most critical. Most real-world programs are in service over a period of years, during which bugs are discovered, enhancements are made, and modifications are made to meet changing requirements. Designing a program in a modular fashion makes maintenance much easier. Maintaining a program should be like maintaining a car—fixing a flat tire shouldn't require overhauling the engine.