Functions and the stack

Michael Nowak

Texas A&M University

Acknowledgement: Lecture slides based on those created by Bjarne Stroustrup for use with his textbook

Overview

Overview

Anatomy of a program in memory

Code / Static Data	Where the code to be executed and other static data (think global variables, things explicitly tagged with the static keyword, etc.) are stored; lifetime of static data objects: throughout program execution
Heap / Free Store	The dynamic memory area, where dynamic objects created are stored; lifetime of heap objects: until explicitly deleted or when the program terminates
	In classical architectures, the stack and heap grow towards one another
Stack	Stores local variables, manages function calls; extensively involved in performing computations; lifetime of 'automatic' objects: persistent until the end of the block that declared them

*** Note: This is a simplified model*

Overview

The stack

- During the execution of your programs, the stack manages function calls that are made

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack
- The `activation record` is responsible for storing:

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack
- The `activation record` is responsible for storing:
  - Any necessary house-keeping information (such as return location)

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack
- The `activation record` is responsible for storing:
  - Any necessary house-keeping information (such as return location)
  - The actual arguments passed to the function

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack
- The `activation record` is responsible for storing:
  - Any necessary house-keeping information (such as return location)
  - The actual arguments passed to the function
  - The local variables defined in that function

The stack

- During the execution of your programs, the stack manages function calls that are made
- Each time a function is called, an `activation record` for that function is `pushed` (added) to the stack
- The `activation record` is responsible for storing:
  - Any necessary house-keeping information (such as return location)
  - The actual arguments passed to the function
  - The local variables defined in that function
- When the function returns to its callee, its `activation record` is `popped` (removed) from the stack

The stack

- When you compile your code, the compiler examines the `variable`s that will reside on the stack when a respective function is called

The stack

- When you compile your code, the compiler examines the `variable`s that will reside on the stack when a respective function is called
- The amount of space that is required for each activation record is known up front

The stack

- When you compile your code, the compiler examines the `variable`s that will reside on the stack when a respective function is called
- The amount of space that is required for each activation record is known up front
- When a respective function is called, that amount of memory will be "allocated" on the stack

The stack

- When you compile your code, the compiler examines the `variable`s that will reside on the stack when a respective function is called
- The amount of space that is required for each activation record is known up front
- When a respective function is called, that amount of memory will be "allocated" on the stack
- When that same function has finished executing, the memory associated with its activation record will be "deallocated"

The stack

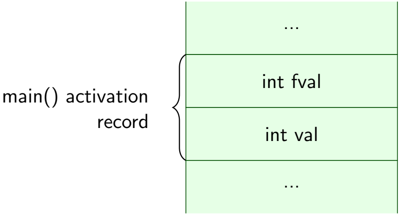
- When you compile your code, the compiler examines the `variables` that will reside on the stack when a respective function is called
- The amount of space that is required for each activation record is known up front
- When a respective function is called, that amount of memory will be "allocated" on the stack
- When that same function has finished executing, the memory associated with its activation record will be "deallocated"
- This is why local variables are known as `automatic variables`: their memory is managed automatically by the function call mechanism

Overview

Simplified example

- When you run your program, an `activation record` for `main` is pushed for the stack

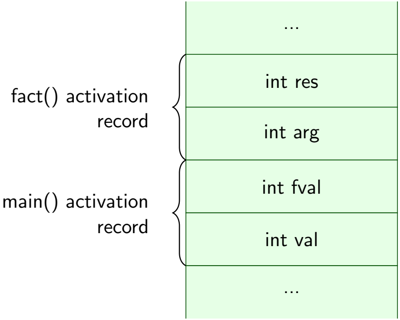
```
int main()
{
    cout << "Number:  ";
    int val;
    cin >> val;
    int fval = fact(val);

    return 0;
}
```



Simplified example

- When our factorial function `fact` (`int fact(int val)`) is called, an activation record for it is pushed (added) to the stack

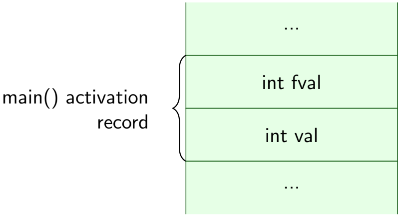
```
int fact(int val)
{
    int res = 1;
    while(val > 1) {
        res *= val;
        val -= 1;
    }
    return res;
}
```



Simplified example

- When our factorial function (`int fact(int val)`) has finished executing, its activation record is popped (removed) from the stack

```
int main()
{
    cout << "Number:  ";
    int val;
    cin >> val;
    int fval = fact(val);

    return 0;
}
```



Overview

Where's my program code stored?

- The code defining each function (including main) is stored in the `code/static` region of your program's address space

Where's my program code stored?

- The code defining each function (including main) is stored in the `code/static` region of your program's address space
- Calling a respective function retrieves the instructions from this region of memory for execution

Where's my program code stored?

- The code defining each function (including main) is stored in the `code/static` region of your program's address space
- Calling a respective function retrieves the instructions from this region of memory for execution
- Meanwhile, the stack will maintain the following in its activation record:
  - Any necessary house-keeping information (such as return location)
  - Any arguments passed to the function
  - Any local variables variables that are declared in the function body

Overview

References

- Lippman, B., Lajoie, Josee, & Moo, B. E. (2016). *C++ primer* (5th ed.). Addison-Wesley.
- Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.