



Introductory concepts for using C++

Problems and Opportunities

- **C++ is both powerful and difficult to learn and use**
- **Both are a result of its tremendous flexibility**
- **A very common source of problems is misusing the dynamic, or heap, memory**

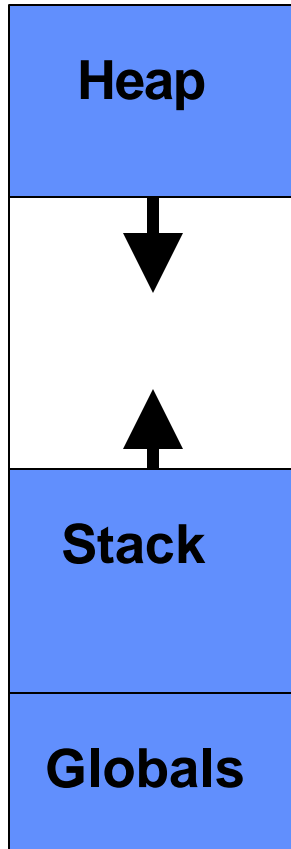
Why is C++ So Difficult to Use

- C++ does not impose a style of programming
- C++ does not do memory management (i.e. there is no garbage collector)
- Many features are specified by syntax rather than keywords
- Operator syntax can be given new user-defined meanings
- values, pointer and references are all used and easily confused
- Objects can exist both on the stack and on the heap
- It is very easy to create memory problems when programming –some suggest that most C++ programs have memory problems.
- Some C++ features (like the assignment statement) work for simple classes but not for more realistic complex classes.
- Some symbols and keywords are used multiple times to mean different things (e.g. the &)
- There are things that you CAN do in C++ that you probably SHOULD NOT do.

Memory Basics

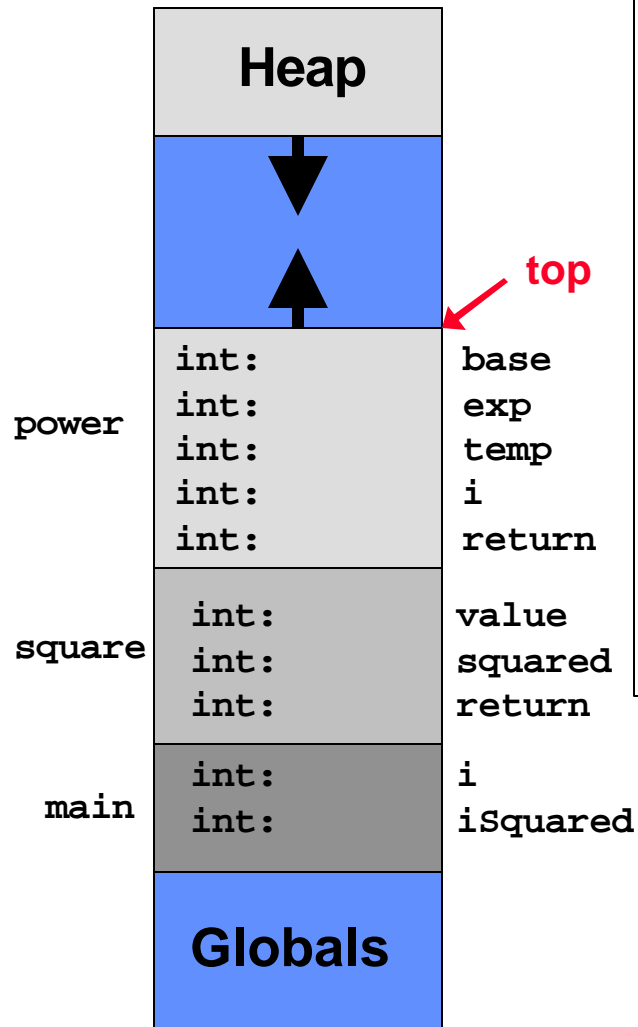
- When a program runs it uses three areas of memory: global memory, procedure stack, and dynamic store (or heap).
- **Global memory** is used to hold the values of constants and literal strings that will be accessed during the execution of the program
- The **procedure stack** provides memory for functions to execute.
- The **heap** provides memory for objects created with `new` and later removed with `delete`

...Memory Basics



- Global area is defined when the program is compiled and loaded with the program is run
- The stack grows and shrinks as procedures are called and then return
- The Heap grows when memory for objects are allocated with `new` and shrinks when objects are removed with `delete`


How The Stack Works



```
int power(int base, int exp){
    int temp = 1;
    for(int i=0; i< exp; i++) temp *= base;
    return temp;
}

int square(int value){
    int squared = power(value, 2);
    return squared;
}

void main()
{
    int i = 12;
    int iSquared = square(i);
    cout << "i= " << i << " i squared= " << iSquared;
}
```

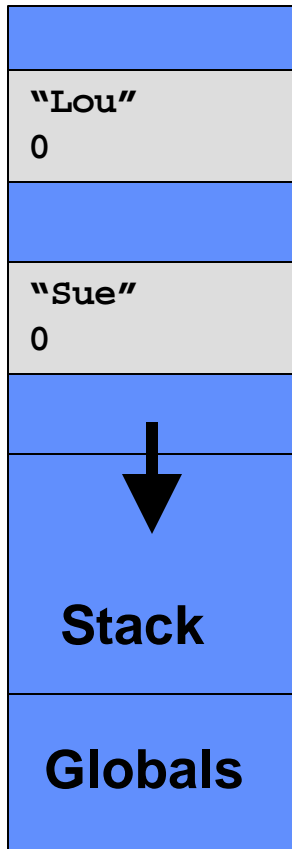


When a procedure is called its stack frame is **pushed** onto the procedure stack. When it returns, this stack frame is **popped**.

Stack Frames provide memory for: parameters, return values, temporary variables, return addresses etc. The required amount of memory is determined at compile time.

Stack memory grows and shrinks automatically with the calls and returns of procedures.

How The Heap Works



```
BankAccount * create(char * name){
    BankAccount * p = new BankAccount(name, 0);
    return p;
}

void closeAccount(BankAccount *bp){
    if(bp->getBalance() == 0)
        delete bp;
}

void main()
{
    BankAccount * account1 = create("Lou");
    BankAccount * account2 = create("Sue");

    //...
    closeAccount(account1);
}
```



Memory is allocated on the Heap by the keyword **new** and removed with **delete**.

Objects remain on the heap until they are deleted.

Notice that objects on the heap do not have a symbolic address like objects on the stack do.

Memory Problems: Your program will crash if:

- Heap collides with the stack.
- Objects are left abandoned on the heap (memory leak).
- Object are deleted off the heap more than once.
- An attempt is made to delete an Object from the stack with `delete`.
- Using a pointer or reference to a deleted object.
- Accidentally creating an alias for an object.
- Accidentally redirecting a pointer and abandoning a heap object.
- Using `delete` with the wrong pointer
- Array is indexed out of bounds.
- Single object is mistaken for an array of objects, or vice versa

Managing Memory in C++

“will the last person to leave please turn out the lights!”

In C++ the programmer must ensure that memory is properly allocated and released from the heap.

Unfortunately in C++:

- There is no easy test to determine if it's safe to delete an object.
- Objects can be placed on the Heap OR on the Stack.
- There is no easy test to determine if an object is on the stack or on the heap.
- There is no easy way to find all the pointers and references to objects you might want to delete.
- It is easy for a pointer to an object to be redirected to point to something else, thereby creating an abandoned object on the heap that cannot be found.