SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220214<A|D>

Replace <A|D> with this section's letter

Pointers II

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Conditional expressions
- Pointers and `const`
- Dynamic memory
- In-class problem

Conditional expressions

Conditional expressions

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& rhs) {
        os << "Name: " << rhs.name << ", ";

        if (rhs.spouse == nullptr) {
            os << "Single";
        } else {
            os << "Married~~ to ~~";
            ~~os << rhs.spouse->name;~~
        }

        return os;
    }

    ...
};
```

Conditional expressions

```cpp
class Person {
    friend ostream& operator<<(ostream& os, const Person& rhs) {
        os << "Name: " << rhs.name << ", ";

        if (rhs.spouse == nullptr) {
            os << "Single";
        } else {
            os << "Married";
        }

        return os;
    }

    ...
};
```

```cpp
os << (rhs.spouse == nullptr ? "Single" : "Married");
```

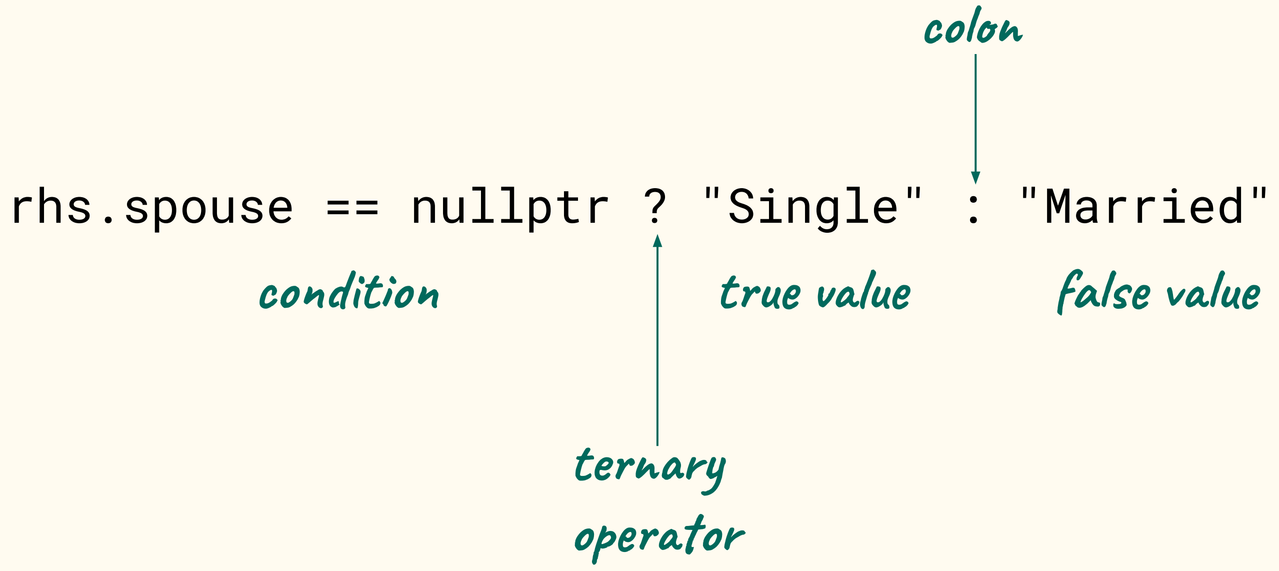
```
% g++ -std=c++11 person.cpp -o person.o
% ./person.o
Name: John, Married
Name: Mary, Married

Name: John, Married
Name: Bill, Single
Name: Mary, Married
```

Conditional expressions

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& rhs) {
        os << "Name: " << rhs.name << ", ";

        os << (rhs.spouse == nullptr ? "Single" : "Married");

        return os;
    }

    ...
};
```

```
% g++ -std=c++11 person.cpp -o person.o
% ./person.o
Name: John, Married
Name: Mary, Married

Name: John, Married
Name: Bill, Single
Name: Mary, Married
```

Conditional expressions



```
rhs.spouse == nullptr ? "Single" : "Married"
```

Pointers and `const`

The `const` keyword

```
const int SPECIAL_VAL = 3167;
```

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220214<A|D>

Replace <A|D> with this section's letter

What is implied by the statement below? What do we know about `SPECIAL_VAL`;

```
const int SPECIAL_VAL = 3167;
```

The `const` keyword and Pointers

```
int x_coord = 17, y_coord = 6;

int* const ptr = &x_coord;  ptr can only point to x_coord

*ptr = 42;  👍

ptr = &y_coord;  compilation error
```

`const` keyword after `*`

The `const` keyword and Pointers

```
int x_coord = 17, y_coord = 6;

const int* ptr = &x_coord;
```

ptr will treat x_coord as const

```
*ptr = 42;
```

compilation error

```
ptr = &y_coord;
```

👍

`const` keyword before `*`

The `const` keyword and Pointers

```
int x_coord = 17, y_coord = 6;

const int* const ptr = &x_coord;

*ptr = 42;
```

compilation error

```
ptr = &y_coord;
```

compilation error

ptr will treat x_coord as const AND
ptr can only point to x_coord

Dynamic memory

when the real fun begins

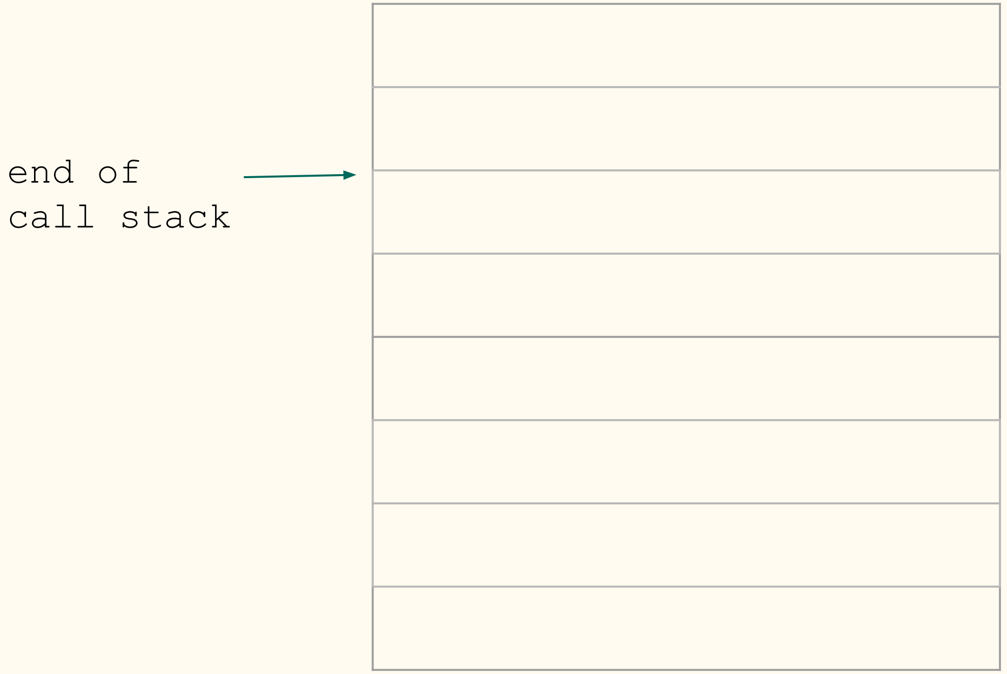
More control

Python

C/C++

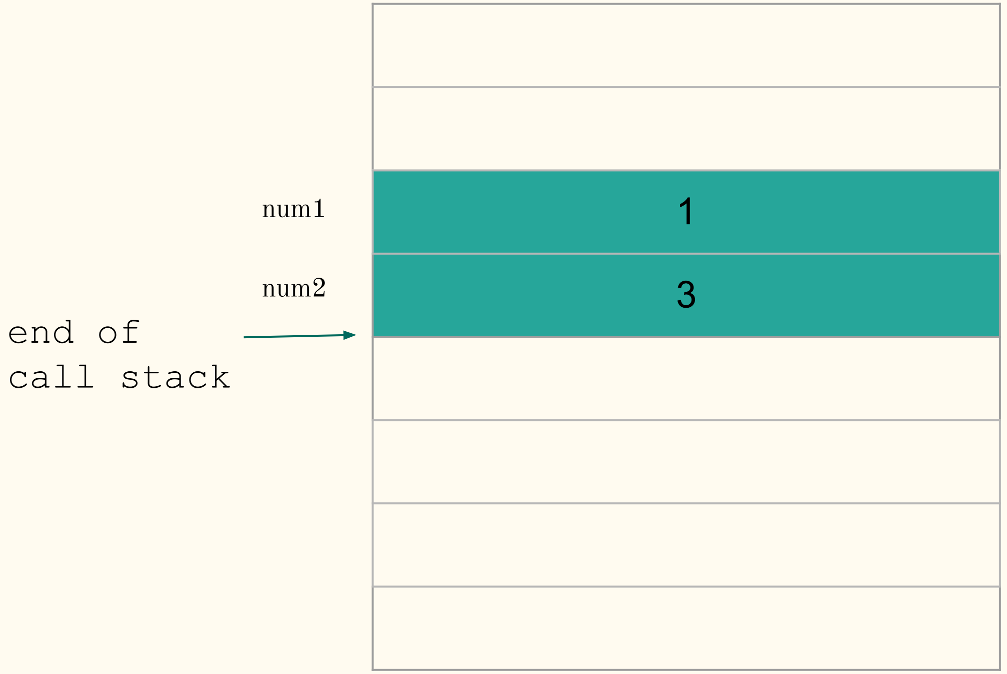
Dynamic memory - heap allocation

```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```



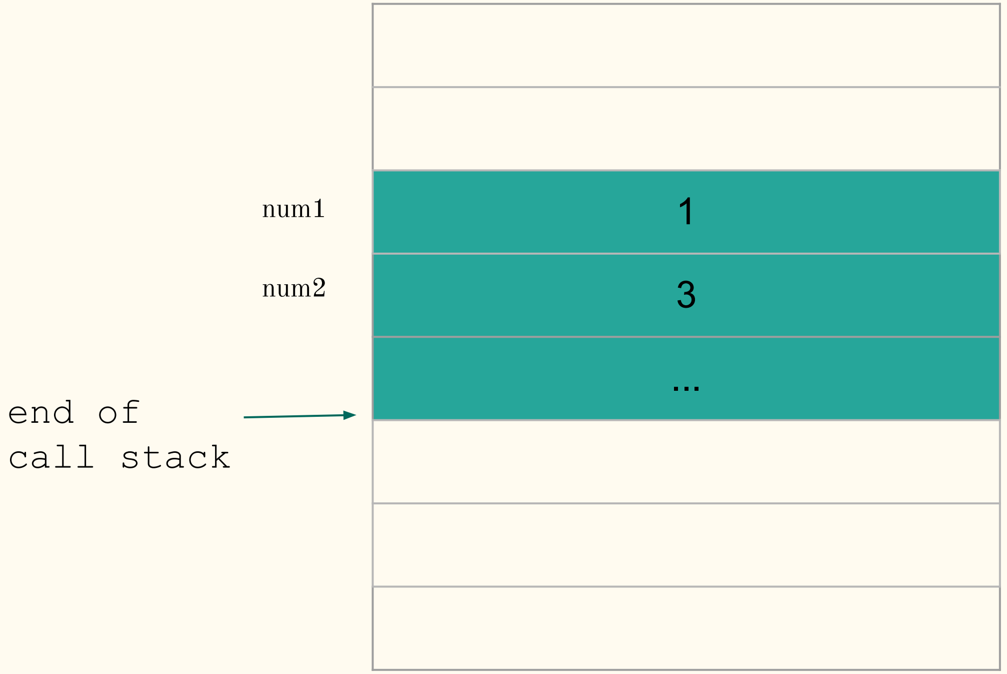
Dynamic memory

```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```



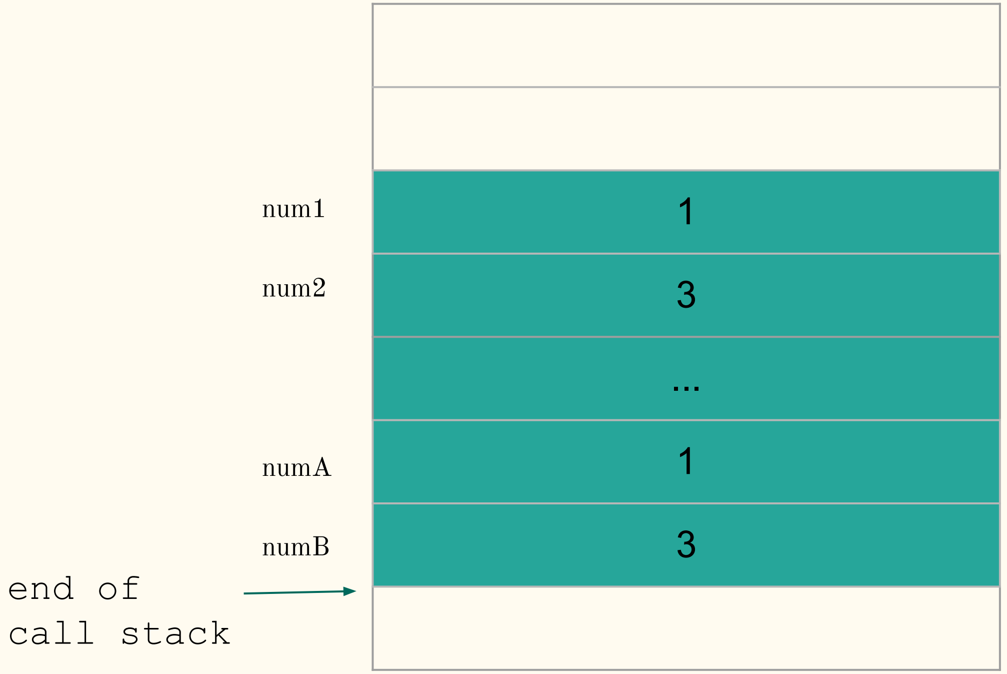
Dynamic memory

```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```



Dynamic memory

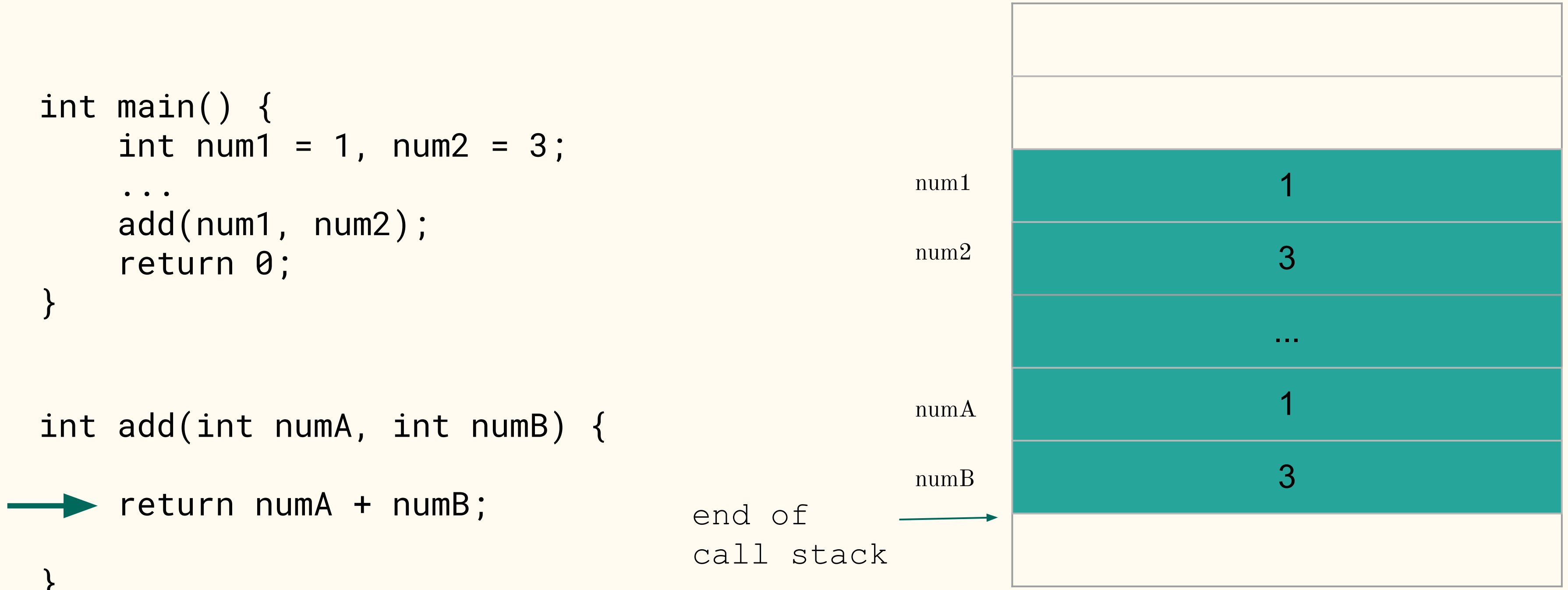
```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```



stack size limited

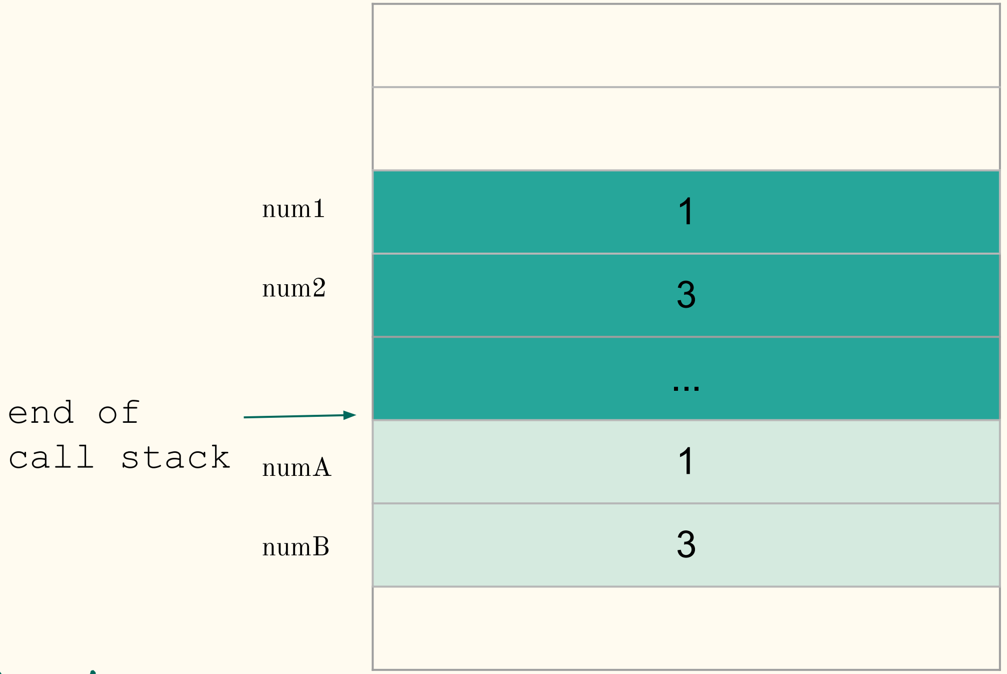
Dynamic memory

```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```



Dynamic memory

```
int main() {
    int num1 = 1, num2 = 3;
    ...
    add(num1, num2);
    return 0;
}


int add(int numA, int numB) {

    return numA + numB;

}
```

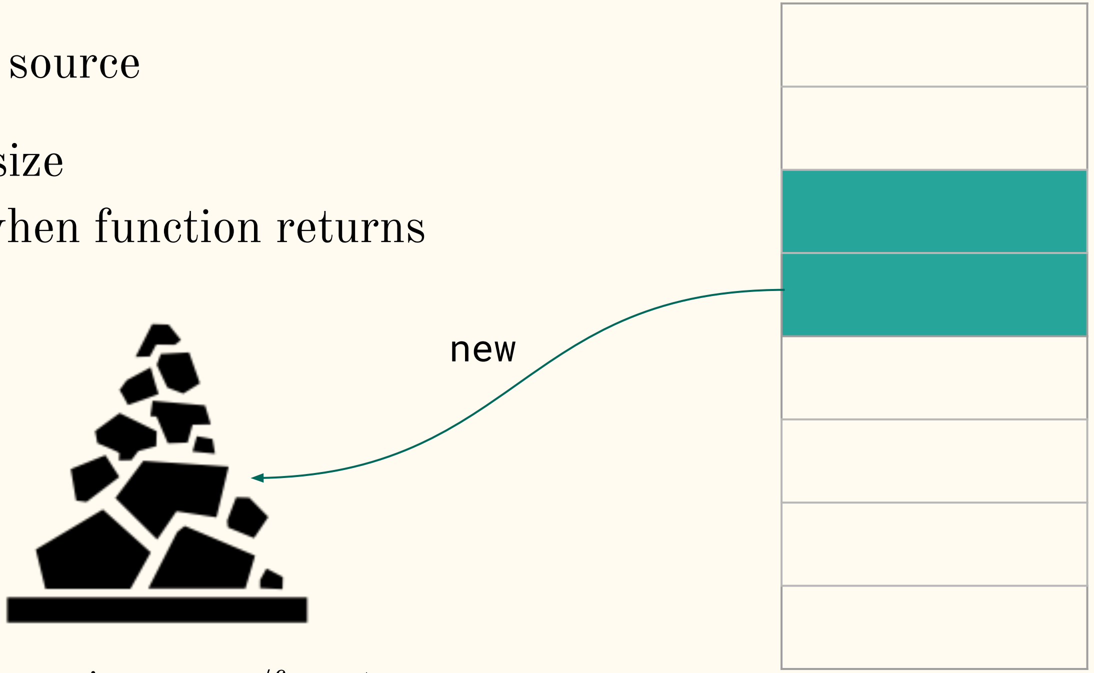


numA/numB no longer accessible to program

Dynamic memory

Need for a large memory source

1) call stack limited in size
2) local variables lost when function returns



heap/dynamic memory/free store
(a large memory source)

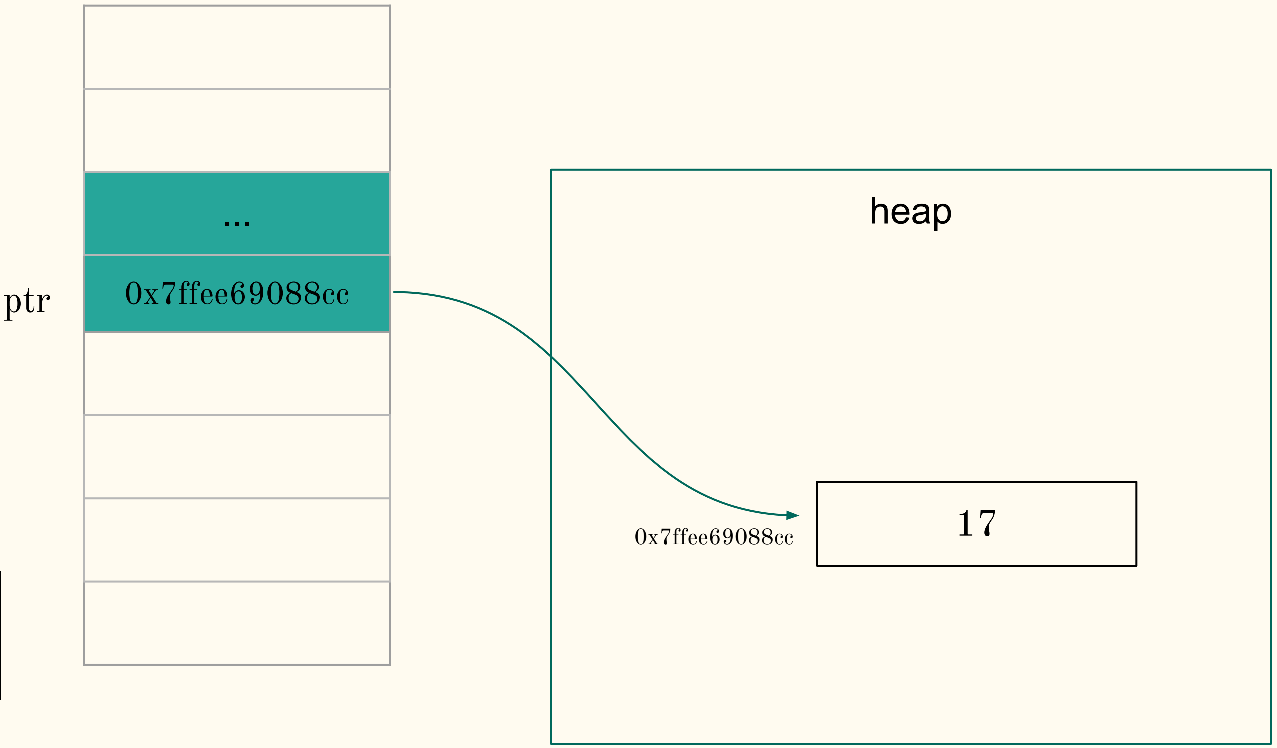
Dynamic memory - heap allocation

```
int* ptr = new int(17);
```

$\underbrace{\texttt{new int(17);}}_{\text{0x7ffee69088cc}}$

```
cout << *ptr << endl;
cout << ptr << endl;
```

```
17
0x7ffee69088cc
```

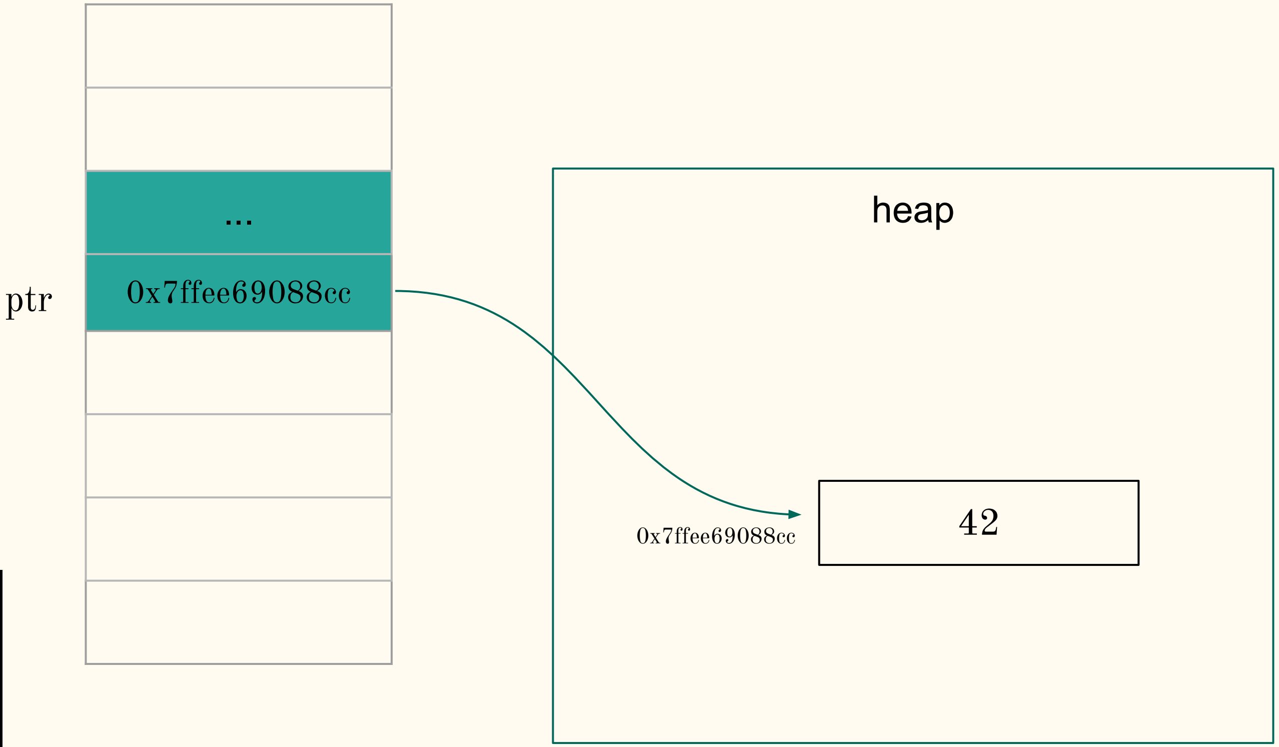


Dynamic memory - heap allocation

$\texttt{int* ptr = } \underbrace{\texttt{new int(17);}}_{\text{0x7ffee69088cc}}$

```
cout << *ptr << endl;
cout << ptr << endl;
*ptr = 42;
cout << *ptr << endl;
cout << ptr << endl;
```

```
17
0x7ffee69088cc
42
0x7ffee69088cc
```

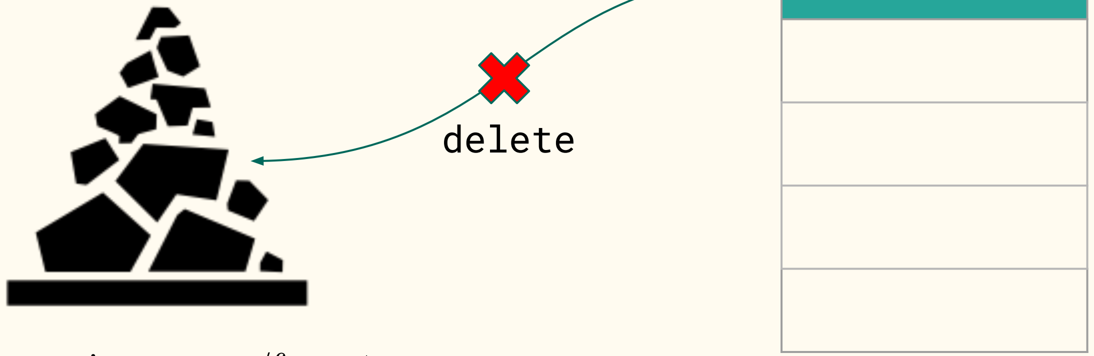


Dynamic memory - freeing memory

Need for a large memory source

1) call stack limited in size
2) local variables lost when function returns

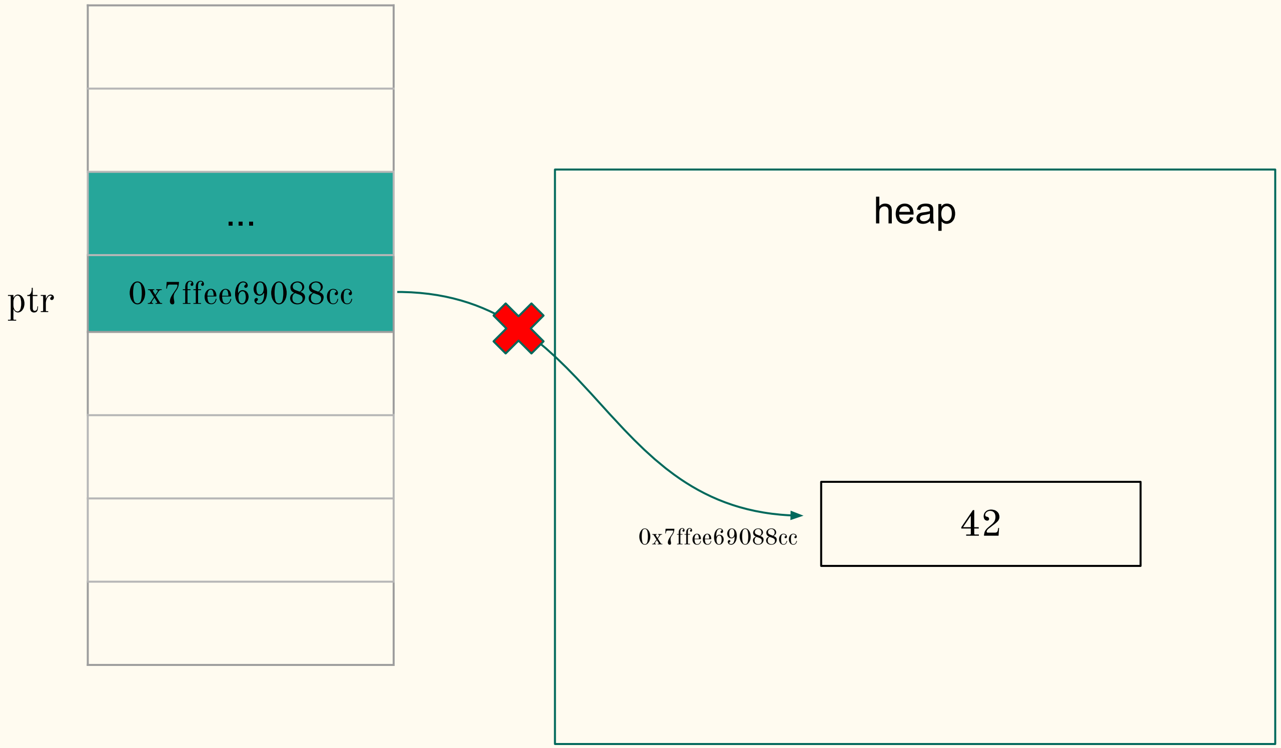
ability to indicate that memory no longer needed



heap/dynamic memory/free store
(a large memory source)

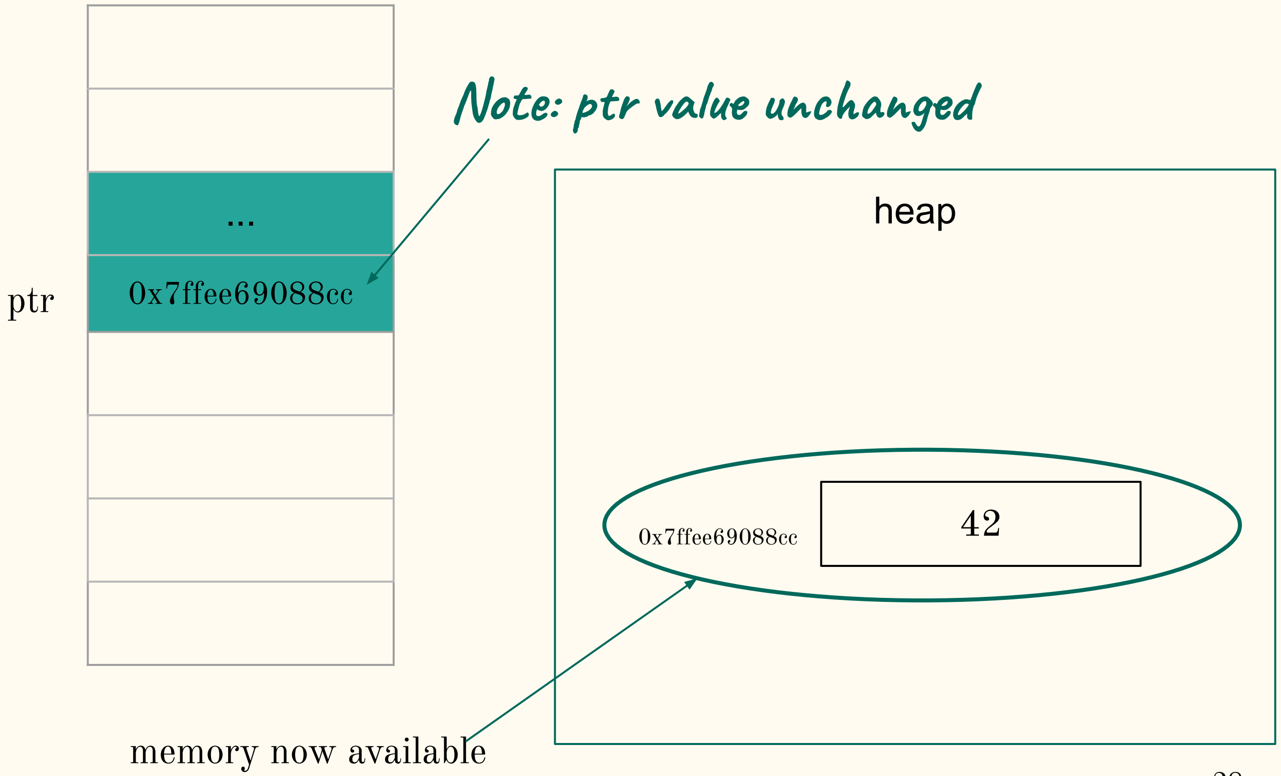
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
```



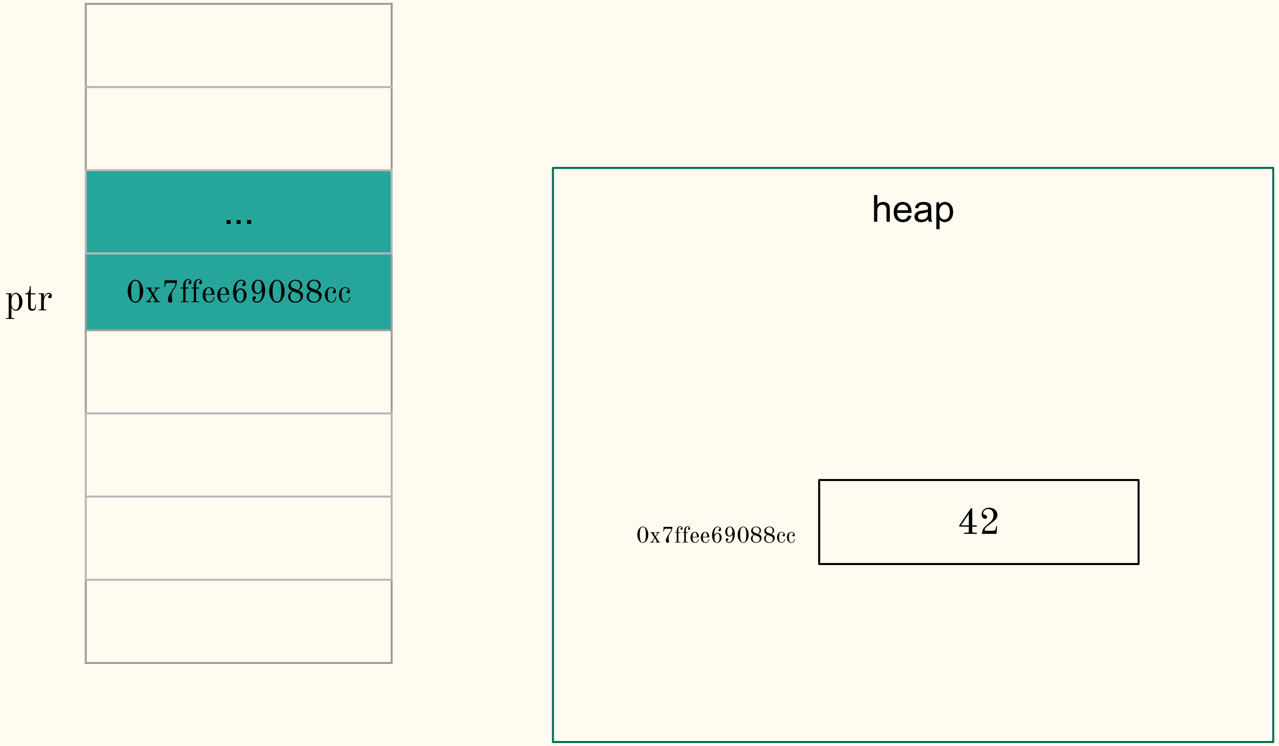
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
```



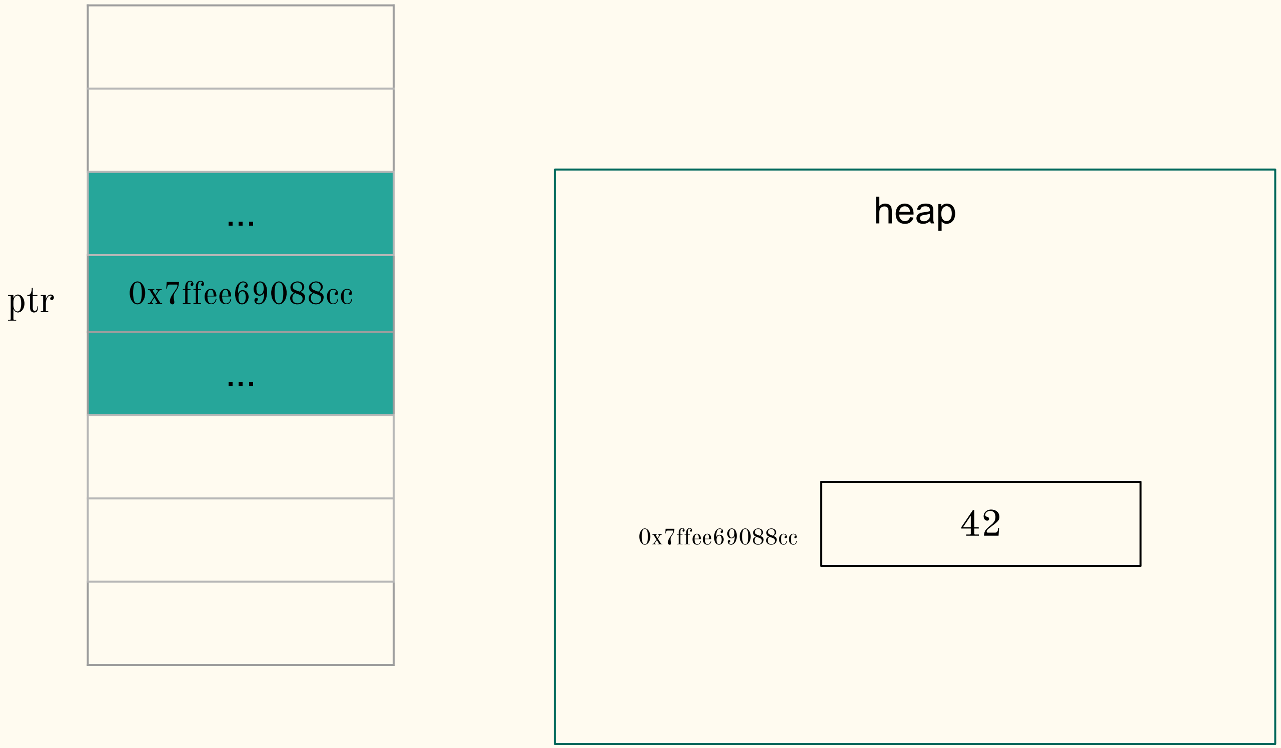
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
```



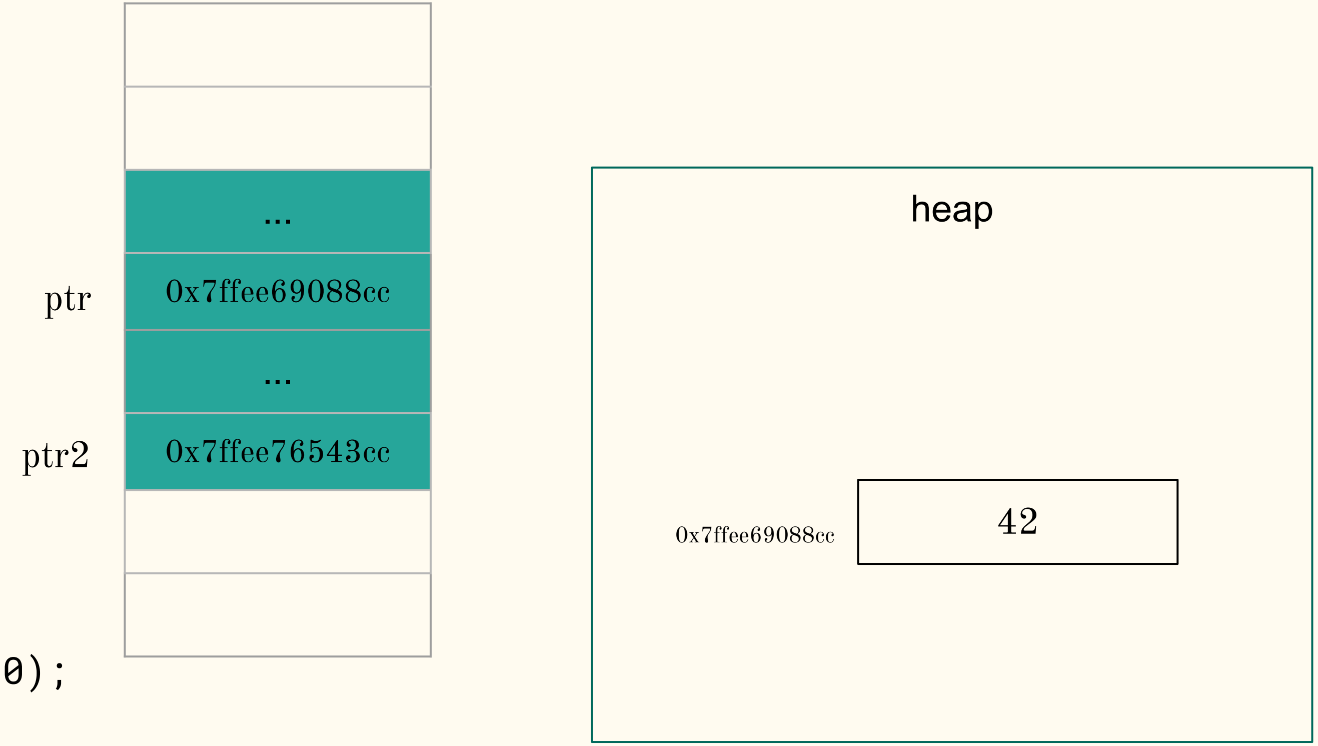
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
```



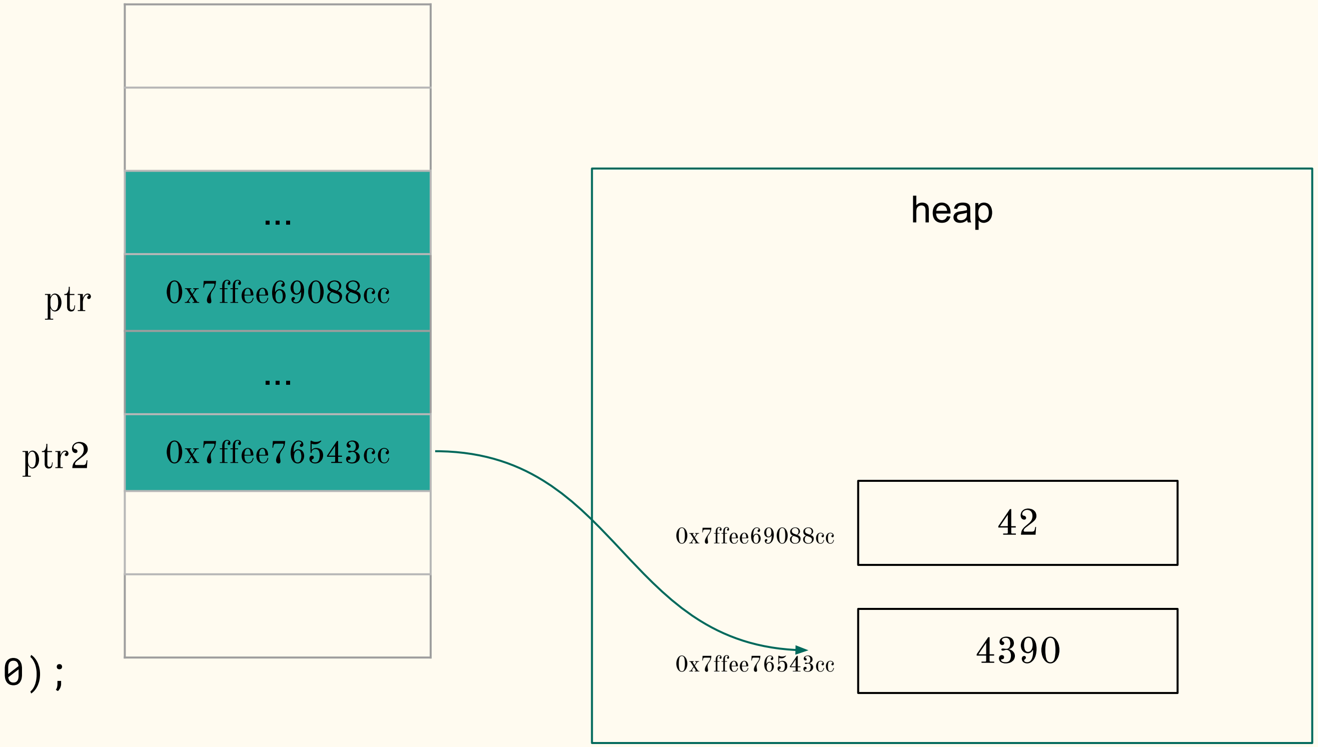
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
int* ptr2 = new int(4390);
```



Dynamic memory - freeing memory

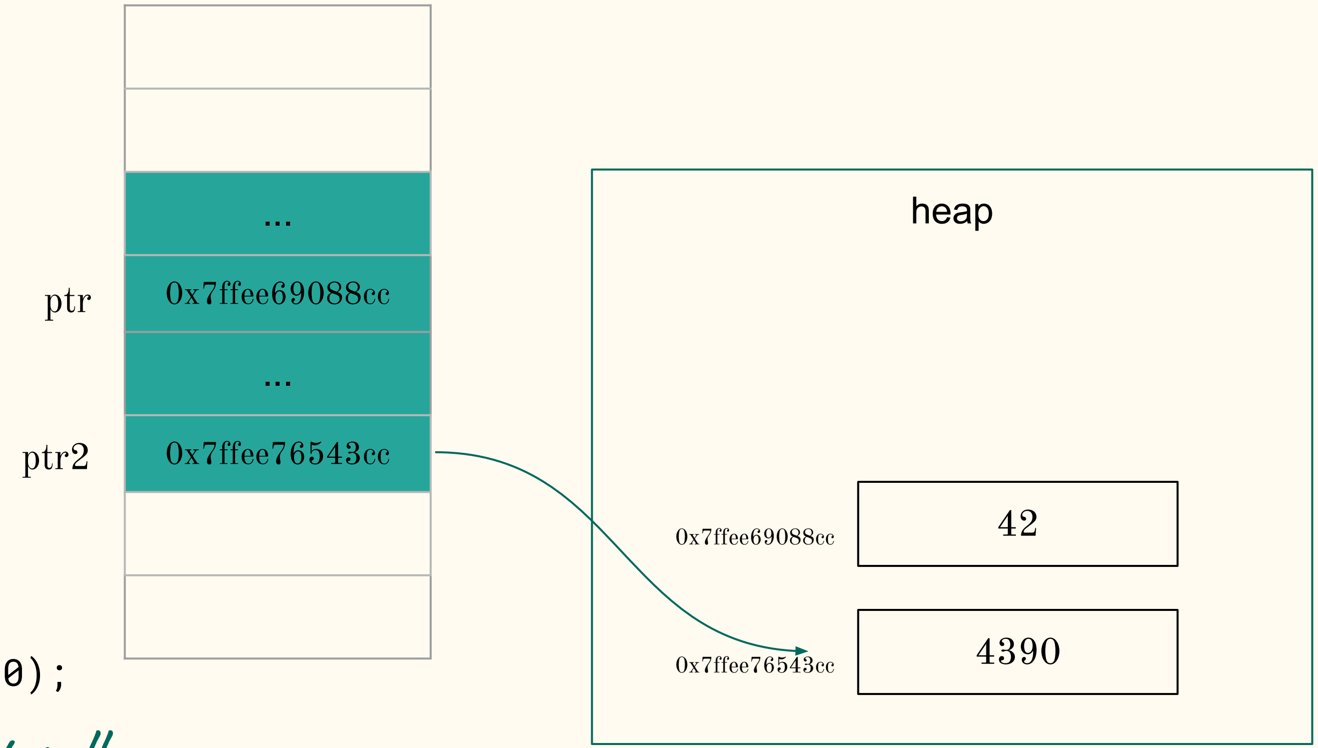
```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
int* ptr2 = new int(4390);
```



Dynamic memory - freeing memory

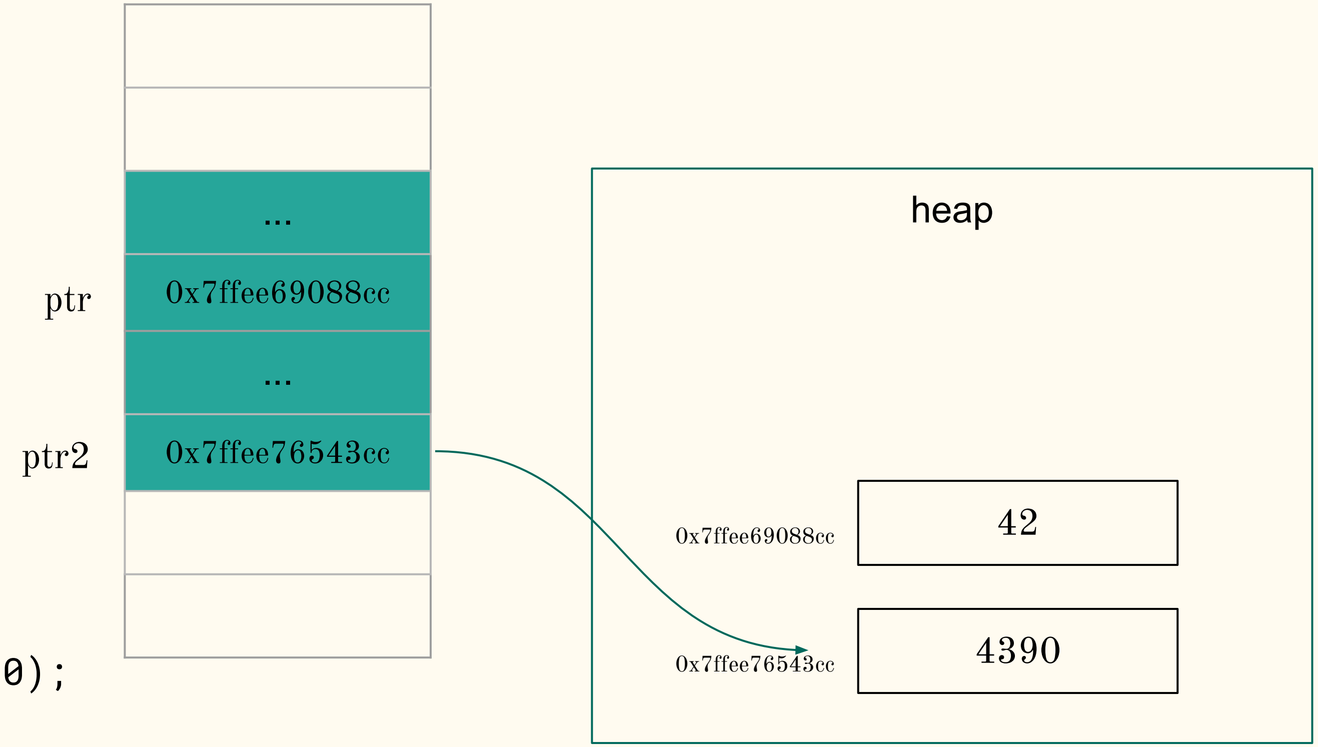
```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
int* ptr2 = new int(4390);
...
int res = 5 * (*ptr);
```

typo!!



What value is going to be assigned to `res`?

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
int* ptr2 = new int(4390);
...
int res = 5 * (*ptr);
```



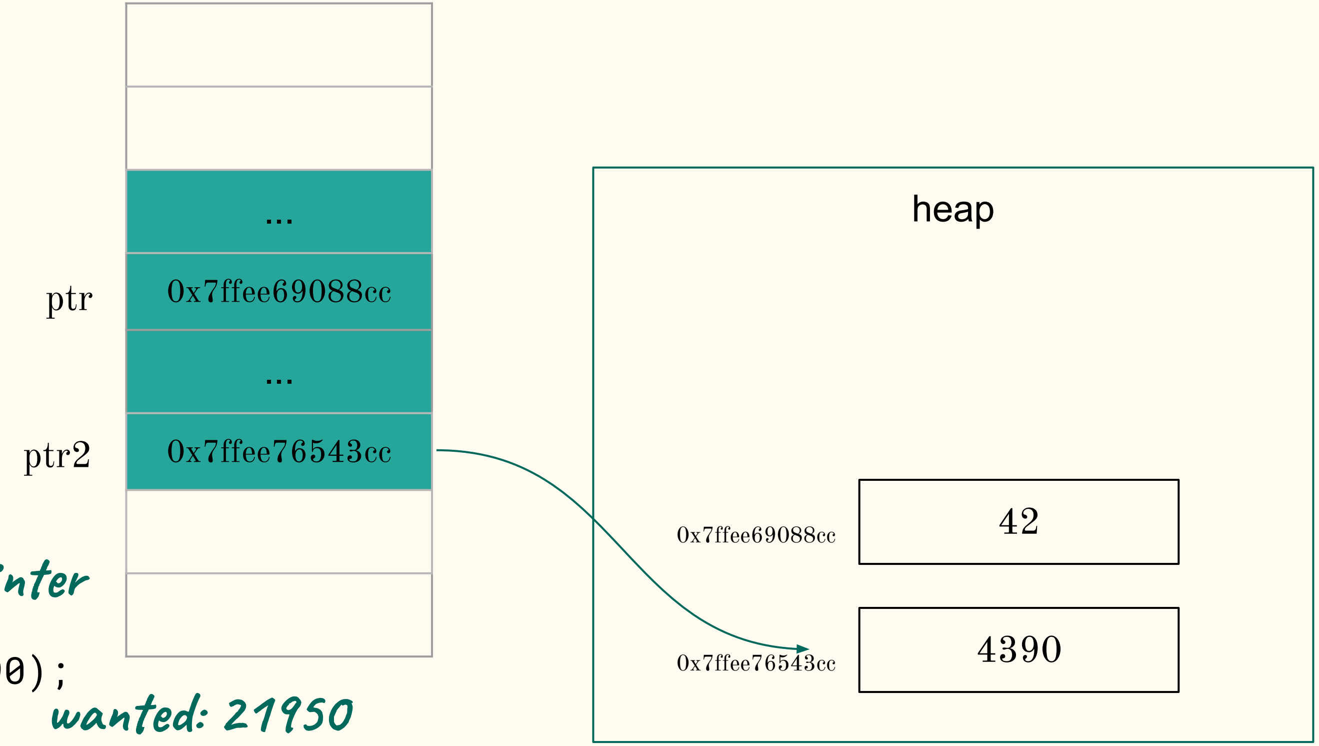
Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;
...
int* ptr2 = new int(4390);
...
int res = 5 * (*ptr);
```

dangling pointer

wanted: 21950

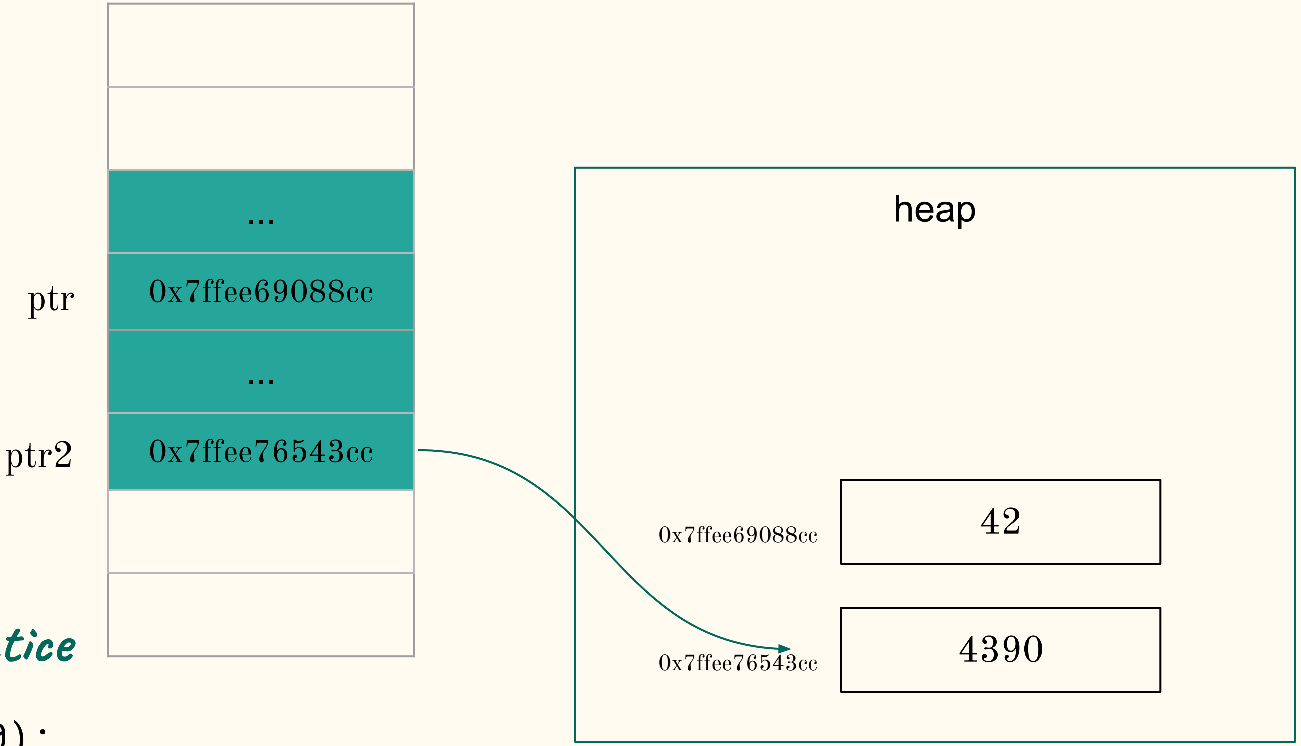
got: 210



Dynamic memory - freeing memory

```
int* ptr = new int(17);

cout << *ptr << endl;
cout << ptr << endl;

*ptr = 42;

cout << *ptr << endl;
cout << ptr << endl;

delete ptr;

ptr = nullptr;
...
int* ptr2 = new int(4390);
int res = 5 * (*ptr);
```

best practice



program crashes

Vectors of pointers

```
void fill_person_vector() {

}
```

Vectors of pointers

```
void fill_person_vector(ifstream& ifs, vector<Person*>& vec) {

    string name;    // Used to read in the name

    while (ifs >> name) {
        Person a_person(name);  // Person object defined inside loop
        vec.push_back(&a_person);  // local variable address added to vector
    }

}
```

Two problems:

1) stack variable no good after function returns
2) pushing *same* address to vector

Vectors of pointers

```
void fill_person_vector(ifstream& ifs, vector<Person*>& vec) {

    string name;    // used to read in the name

    while (ifs >> name) {
        ~~Person a_person(name);~~  // Person object defined inside loop
        vec.push_back~~(&a_person)~~;  // local variable address added to vector
    }
}
```

Vectors of pointers

```
void fill_person_vector(ifstream& ifs, vector<Person*>& vec) {

    string name;    // used to read in the name

    while (ifs >> name) {
        Person* ptr = new Person(name);  // new Person object defined on each iteration
        vec.push_back~~(&a_person)~~;  // local variable address added to vector
    }
}
```

Vectors of pointers

```
void fill_person_vector(ifstream& ifs, vector<Person*>& vec) {

    string name;    // used to read in the name

    while (ifs >> name) {
        Person* ptr = new Person(name);  // new Person object defined on each iteration
        vec.push_back(ptr);  // address from heap added to vector
    }
}
```

access to ptr variable lost when function returns

¯_(ツ)_/¯

Vector of pointers

```
void display_person_pointer_vector(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
    }

}
```

Vector of pointers

```
void display_person_pointer_vector(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
        ___
    }

}
```

Vector of pointers

```
void display_person_pointer_vector(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
        _1_
    }

}
```

Which statement replaces blank #1 to free the Person object's heap memory?

```
void display_person_pointer_vector(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {

    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
        _1_
    }

}
```

Vector of pointers

```
void display_person_pointer_vector(const vector<Person*>& vpp) {
    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {
    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
        delete vpp[i];
    }
}
```

Vector of pointers

```
void display_person_pointer_vector(const vector<Person*>& vpp) {
    for (size_t i = 0; i < vpp.size(); ++i) {
        vpp[i]->display();
    }
}
```

arrow operator (->) needed

```
void free_person_pointer_vector_memory(const vector<Person*>& vpp) {
    for (size_t i = 0; i < vpp.size(); ++i) {
        // free Person object's heap memory
        delete vpp[i];
        vpp[i] = nullptr;
    }
}
```

Tips for using pointers

1. Always *initialize* before use
2. After deleting a pointer, assign `nullptr` to deleted pointer
3. Don't delete a pointer more than once
4. *Never* return the address of a local variable
5. Take *extreme* caution with two pointers pointing to same address

In-class problem

Person objects (yes...again)

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
};
```

track age of a Person

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    // variable for age
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    // variable for age
    ___
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    // variable for age
    _1_
};
```

Which declaration replaces blank #1 to add a member variable, `age`, to the `Person` class?

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    // variable for age
    _1_
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    // variable for age
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, ___) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, _2_) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Which parameter declaration replaces blank #2 to make constructor require the `Person`'s `age` on creation?

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, _2_) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, int age) : name(name) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), ___ {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), _3_ {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Which expression replaces blank #3 for initializing the `Person` object's `age`?

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), _3_ {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name; // display age also
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << ___;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << _4_;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Which expression replaces blank #4 to insert the `Person` object's `age` into `ostream& os`?

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << _4_;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects with ages

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << someone.age;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects and pointers

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << someone.age;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

Person objects and pointers

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& someone) {
        os << "Person: " << someone.name << ", " << someone.age;
        return os;
    }

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

private:
    string name;
    int age;
};
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    // open file for reading
    ___ ___(___);

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    // open file for reading
    _5_ ___(___);

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Which type replaces blank #5 to create an input file stream object?

```
int main() {
    // open file for reading
    _5_ ___(___);

}
```

Person objects and pointers

```
int main() {
    // open file for reading
    ifstream ___(___);

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    // open file for reading
    ifstream ifs(_6_);

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Which argument to the constructor replaces blank #6 when declaring the input file stream object?

```
int main() {
    // open file for reading
    ifstream ifs(_6_);

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    // open file for reading
    ifstream ifs("stooges.txt");

}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    // declare a vector of Person pointers


}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    // declare a vector of Person pointers
    ___ stooges;


}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    // declare a vector of Person pointers
    _7_ stooges;


}
```

Which type declaration replaces blank #7 to create a `vector` of `Person` pointers?

```
int main() {
    ifstream ifs("stooges.txt");

    // declare a vector of Person pointers
    _7_ stooges;


}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    // declare a vector of Person pointers
    vector<Person*> stooges;


}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;


}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        ___ ptr = ___ ___;
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        _8_ ptr = ___ ___;
    }
}
```

Which type replaces blank #8 to assign the `Person` object's address from the heap to the variable `ptr`?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        _8_ ptr = ___ ___;
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = ___ ___;
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = ___ _9_;
    }
}
```

Which constructor invocation replaces blank #9 to create a `Person` object using `name` and `age` read from the input file?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = ___ _9_;
    }
}
```

```
class Person {
    ...

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

...
};
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = ___ Person(name, age);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = _10_ Person(name, age);
    }
}
```

Which operator replaces blank #10 providing the `Person` object's memory address from the heap?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = _10_ Person(name, age);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = new Person(name, age);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        // create Person object from the heap
        Person* ptr = new Person(name, age);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);


    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);

        // add Person objects's address to vector
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);

        // add Person object's address to vector
        stooges.push_back(___);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);

        // add Person object's address to vector
        stooges.push_back(_11_);
    }
}
```

What replaces blank #11 to add the `Person` object's address to the `stooges` vector?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);

        // add Person object's address to vector
        stooges.push_back(_11_);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {
        Person* ptr = new Person(name, age);

        // add Person object's address to vector
        stooges.push_back(ptr);
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {

        Person* ptr = new Person(name, age);
        stooges.push_back(ptr);
    }

}
```

$\}$ *can do this in one line*

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges;

    string name;
    int age;

    while (ifs >> name >> age) {

        stooges.push_back(new Person(name, age));

    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    // close the input file
    ___

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    // close the input file
    _12_

}
```

Which method invocation replaces blank #12 to close the input file stream `ifs`?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    // close the input file
    _12_

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    // close the input file
    ifs.close();

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << ___ << '\t' << ___ << endl;
    }

}
```

```
Moe      0x7f83f1d04630
Larry    0x7f83f1d04660
Curly    0x7f83f1d04690
...
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << _13_ << '\t' << ___ << endl;

    }

}
```

Which method invocation replaces blank #13 to output the `name` of the `Person` object pointed to by `ptr`?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << _13_ << '\t' << ___ << endl;
    }

}
```

```
class Person {
...

public:
    Person(const string& name, int age) : name(name), age(age) {}
    const string& get_name() const { return name; }

…

};
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << ptr->get_name() << '\t' << ___ << endl;

    }

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << ptr->get_name() << '\t' << _14_ << endl;

    }

}
```

What replaces blank #14 to output the *address* from the heap where the `Person` object is located?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
       // output Person object name and heap address
       cout << ptr->get_name() << '\t' << _14_ << endl;
    }

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      // output Person object name and heap address
      cout << ptr->get_name() << '\t' << ptr << endl;

    }

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
      cout << ptr->get_name() << '\t' << ptr << endl;
    }

}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
       // free allocated memory
       ---
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
       // free allocated memory
       _15_
    }
}
```

Which statement replaces blank #15 to free the heap memory allocated to the `Person` object pointed to by `ptr`?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
       // free allocated memory
       _15_
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
       // free allocated memory
       delete ptr;
    }
}
```

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        // eliminate dangling pointer
        ___
    }
}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        // eliminate dangling pointer
        _16_
    }
}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Which statement replaces blank #16 to ensure no dangling pointers remain after the for loop completes?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        // eliminate dangling pointer
        _16_
    }
}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        // eliminate dangling pointer
        ptr = nullptr;
    }
}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        ptr = nullptr;
    }
}
```

copy of vector item

dangling pointers remain

variable storing copy modified to store nullptr

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        ptr = nullptr;
    }
}
```

copy of vector item

dangling pointers remain

variable storing copy modified to store nullptr

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

How can we modify the `for` loop to ensure that the dangling pointers are eliminated?

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person* ptr : stooges) {
        delete ptr;
        ptr = nullptr;
    }
}
```

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

Person objects and pointers

```
int main() {
    ifstream ifs("stooges.txt");

    vector<Person*> stooges; // vector of pointers!

    string name;
    int age;

    while (ifs >> name >> age) {
        stooges.push_back(new Person(name, age));
    }

    ifs.close();

    for (Person* ptr : stooges) {
        cout << p->get_name() << '\t' << ptr << endl;
    }

    for (Person*& ptr : stooges) {
        delete ptr;
        ptr = nullptr;
    }
}
```

Person reference -- allows modification of vector items*

```
Moe    77
Larry  72
Curly  48
...
```

stooges.txt

```
% g++ -std=c++11 load_stooges.cpp -o load_stooges.o
% ./load_stooges.o
Moe      0x7f83f1d04630
Larry    0x7f83f1d04660
Curly    0x7f83f1d04690
...
```