COPY ELISION & MOVE SEMANTICS

Plan for Today

- □ Copy Elision: RVO, NRVO, URVO
- Move Semantics (Motivation)

RAII Vector Class

```
class Vec {
 size t len{};
  int *ptr{nullptr};
public:
 Vec() = default;
 ~Vec() { delete [] ptr; }
 Vec(Vec const& rhs)
  : len{rhs.len}, ptr{new int [len]} {
    std::copy(rhs.ptr, rhs.ptr+len, ptr);
 Vec& operator=(Vec const& rhs) {
   Vec copy{rhs};
    copy.swap(*this);
    return *this;
```

RAII Classes: Rule of Three

- If your class manages a resource, you'll need to write three special member functions:
 - Destructor to release the resource
 - Copy constructor to clone the resource
 - Copy assignment operator to release current resource and clone resource of assigned object

C++'s Copy Problem

- Perception that C++ is overly fond of copying
 - Pass-by-value means invoking copy constructor
 - Return-by-value means invoking copy constructor
 - Assignment means invoking copy assignment operator
 - STL containers employ value semantics

C++'s Copy Problem

 Based on our understanding of stack-based function semantics in C/C++, one would categorically assert that every invocation of following functions requires invocation of copy ctor

```
void foo(X xx) {
    // use xx
}

int main() {
    X x;
    // use x
    foo(x);
    // use x
}
```

```
X bar() {
    X xx;
    // process xx
    return xx;
}

int main() {
    X x = bar();
    // use x
}
```

C++'s Copy Problem

- Based on our understanding of stack-based function semantics in C/C++, one would categorically assert that every invocation of following functions requires invocation of copy ctor
- Pass-by-reference becomes default mode of transferring resources to functions

Return Value Optimization [RVO]

- Most high-quality C++ implementations allow <u>copy</u> <u>elision</u> [that is, omit copying] even in cases where copy ctors and dtors may have side effects
 - □ Copy elision now part of C++17
- This compiler optimization is more commonly called Return Value Optimization
 - Avoids copying object that function returns as its value
 - Avoids creation of temporary object
 - Permits function to efficiently return large objects
 - Simplifies function's interface
 - Eliminates scope for issues such as resource leaks

Functions: Pass-by-Value Convention (1/20)

this variable is called formal parameter or just parameter

```
int myabs(int number) {
  return number < 0 ? -number : number;
}</pre>
```

client calls function myabs using function call operator ()

```
int num = 10; this expression is called function argument num = myabs(-num)
```

- 1) At runtime, expression (or argument) num is evaluated
- 2) Result of evaluation is used to initialize parameter number
- 3) Changes made to parameter number are localized to function myabs
- 4) Function myabs terminates by returning value of type int
- 5) When function myabs terminates, variable number ceases to exist

Functions: Pass-by-Value Convention (2/20)

Example

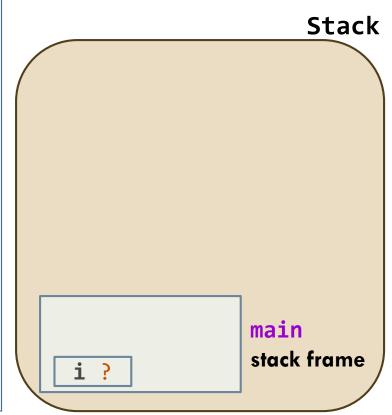
```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Output

Before call: i is 5 In foo, x is 5 In foo, x is now 10 After call: i is 5

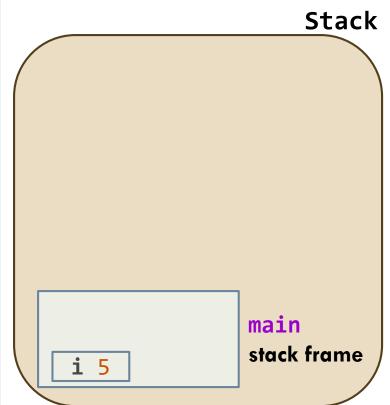
Functions: Pass-by-Value Convention (3/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
→ int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```



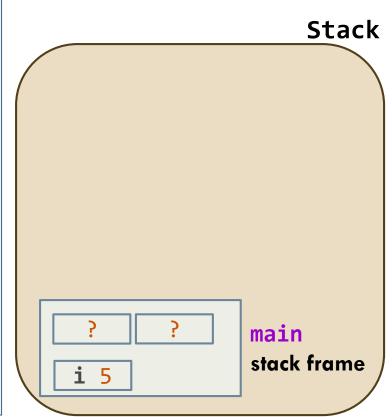
Functions: Pass-by-Value Convention (4/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
\rightarrow i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```



Functions: Pass-by-Value Convention (5/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
→ printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```



Functions: Pass-by-Value Convention (6/20)

```
#include <stdio.h>
void foo(int x) {
                                                                 Stack
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
→ printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
                                              str
  printf("After call: i is %d\n", i);
                                                             main
  return 0;
                                                             stack frame
                                              i 5
```

Functions: Pass-by-Value Convention (7/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
→ printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5

Stack local variables in function printf printf var1 var2 stack frame p str str main stack frame **i** 5

Functions: Pass-by-Value Convention (8/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
→ foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5

Stack main stack frame

Functions: Pass-by-Value Convention (9/20)

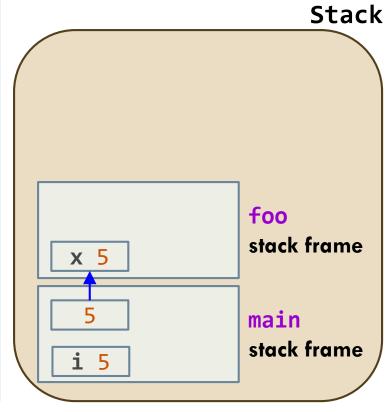
Before call: i is 5

```
#include <stdio.h>
void foo(int x) {
                                                                  Stack
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
→ foo(i); // call to function foo
                                                             main
  printf("After call: i is %d\n", i);
  return 0;
                                                             stack frame
                                               i 5
```

Functions: Pass-by-Value Convention (10/20)

```
#include <stdio.h>
void foo(int x) {
   printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
   printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
   printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5



Functions: Pass-by-Value Convention (11/20)

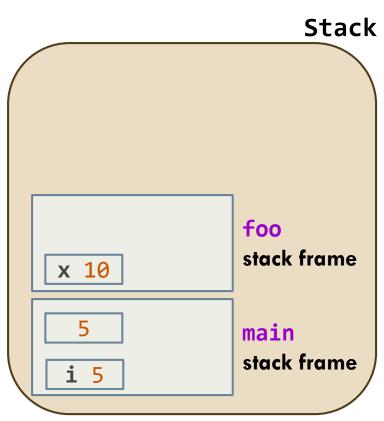
```
Before call: i is 5
                                                  In foo, x is 5
#include <stdio.h>
void foo(int x) {
                                                                       Stack
→ printf("In foo, x is %d\n", x);
  x = 10;
  x = 10;

printf('In foo, x is now %d\n''\ x);
                                                                 printf
                                                         var2
                                                 var1
                                                                 stack frame
                                                 p str
int main(void) {
  int i;
                                                 str
                                                                 foo
  i = 5;
                                                                 stack frame
                                                  x 5
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
                                                                 main
  return 0;
                                                                 stack frame
                                                  i 5
```

Functions: Pass-by-Value Convention (12/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
\rightarrow x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

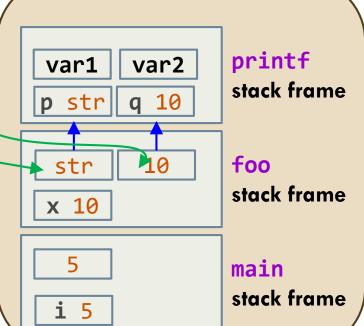
Before call: i is 5 In foo, x is 5



Functions: Pass-by-Value Convention (13/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
 x = 10;
▶ printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10



Functions: Pass-by-Value Convention (14/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10

Stack foo stack frame x 10 main stack frame **i** 5

Functions: Pass-by-Value Convention (15/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
▶ printf("After call: i is %d\n", i);
  return 0;
```

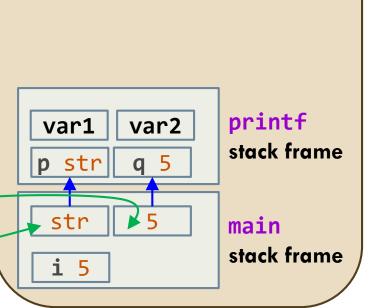
Before call: i is 5 In foo, x is 5 In foo, x is now 10

Stack main stack frame **i** 5

Functions: Pass-by-Value Convention (16/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
▶ printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10 After call: i is 5



Functions: Pass-by-Value Convention (17/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
→ return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10 After call: i is 5



Functions: Pass-by-Value Convention (18/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10 After call: i is 5

Functions: Pass-by-Value Convention (19/20)

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
  x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

Before call: i is 5 In foo, x is 5 In foo, x is now 10 After call: i is 5

Main takeaway:

Inter-function communication uses pass-by-value semantics. Using the stack, copy of argument i is passed to function foo to initialize parameter X.

Changes made to parameter X do not affect argument i!!!

Functions: Pass-by-Value Convention (20/20)

□ <u>Visualization</u> of program

```
#include <stdio.h>
void foo(int x) {
  printf("In foo, x is %d\n", x);
 x = 10;
  printf("In foo, x is now %d\n", x);
int main(void) {
  int i;
  i = 5;
  printf("Before call: i is %d\n", i);
  foo(i); // call to function foo
  printf("After call: i is %d\n", i);
  return 0;
```

RVO, NRVO, URVO

- URVO: copy elision of unnamed objects
- NRVO: copy elision of named objects
- RVO: copy elision of named and unnamed objects

Copy Elision in C++17

- Compilers are required to provide copy elision when function returns unnamed [temporary] object
- Not required to provide copy elision when function returns named object
- Whether copy elision helpful or not depends on how function's return value is consumed

URVO

```
// without URVO
Str urvo(char const *prc) {
  return Str{prc}; // 1) ctor
} // 2) copy ctor for unnamed copy
// 3) dtor of temporary

int main() {
  // 4) copy ctor for s
  Str s = urvo("s");
  // 5) dtor for unnamed copy 2)
} // 6) dtor for s
```

```
// with URVO
Str urvo(char const *prc) {
    // 1) ctor for s in calling
    // environment
    return Str{prc};
}

int main() {
    Str s = urvo("s");
} // 2) dtor for s
```

NRVO

```
// without NRVO
Str nrvo(char const *prc) {
   Str x{prc}; // 1) ctor for x
   // process x
   return x; // 2) copy ctor
} // 3) dtor for x

int main() {
   // s constructed by step 2
   Str s = nrvo("s"); s
} // 4) dtor for s
```

```
// with NRVO
Str nrvo(char const *prc) {
  Str x{prc}; // 1) ctor for s
  // process x
  return x;
}

int main() {
  Str s = nrvo("s");
} // 2) dtor for s
```

stack

```
33
```

```
std::vector<Str> f98() {
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
 w.push_back(s+s);
 w.push_back(s);
  return w;
std::vector<Str> v = f98();
```

```
heap
```

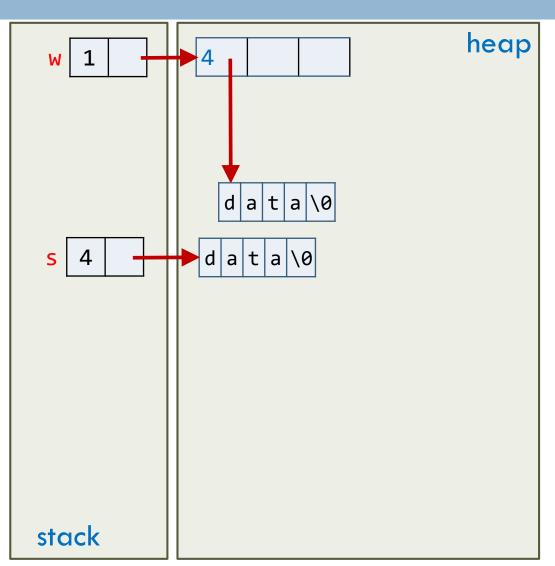
Motivation for RVO

```
34
std::vector<Str> f98() {
                                                                    heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
  w.push_back(s+s);
  w.push_back(s);
                                               d a t a \0
  return w;
std::vector<Str> v = f98();
                                  stack
```

Motivation for RVO

```
std::vector<Str> f98() {
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
 w.push_back(s+s);
 w.push_back(s);
  return w;
std::vector<Str> v = f98();
```

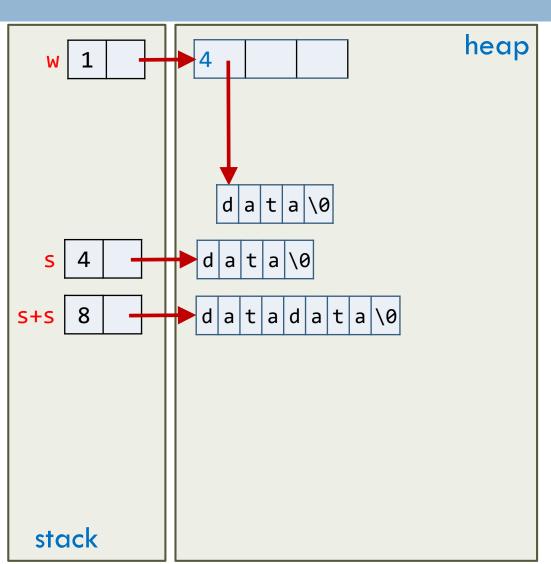
35



Motivation for RVO

```
std::vector<Str> f98() {
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
 w.push_back(s+s);
 w.push_back(s);
  return w;
std::vector<Str> v = f98();
```

36



```
37
std::vector<Str> f98() {
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
  w.push_back(s+s);
  w.push_back(s);
  return w;
                                S+S
std::vector<Str> v = f98();
```

```
heap
              datadata\0
           data\0
          data\0
          datadata\0
stack
```

```
std::vector<Str> f98() {
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
 w.push_back(s+s);
 w.push_back(s);
  return w;
std::vector<Str> v = f98();
```

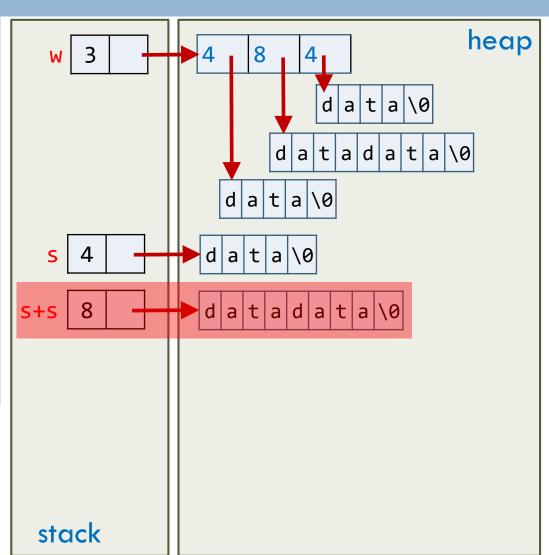
38

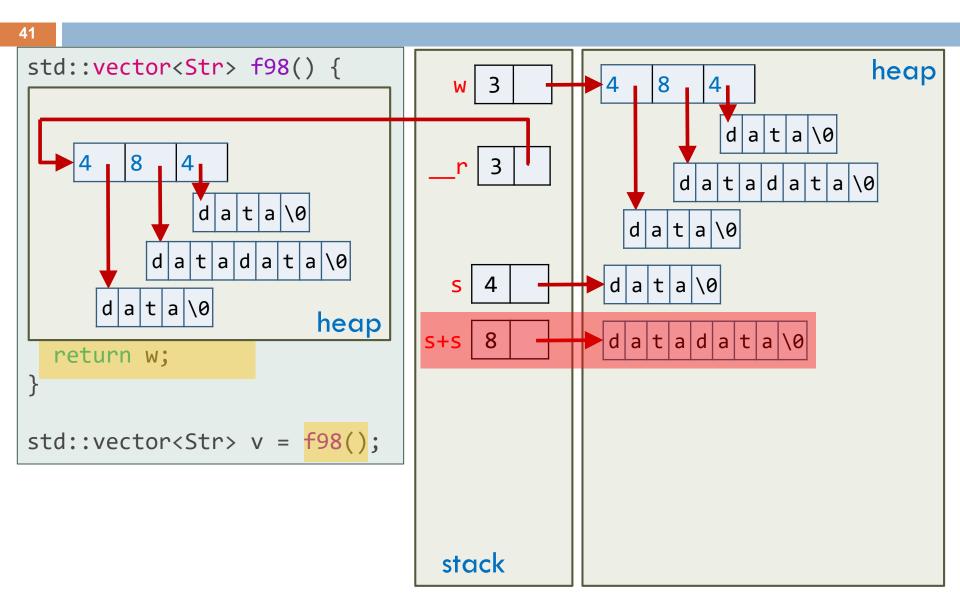
```
heap
              datadata\0
           data\0
          data\0
          datadata
stack
```

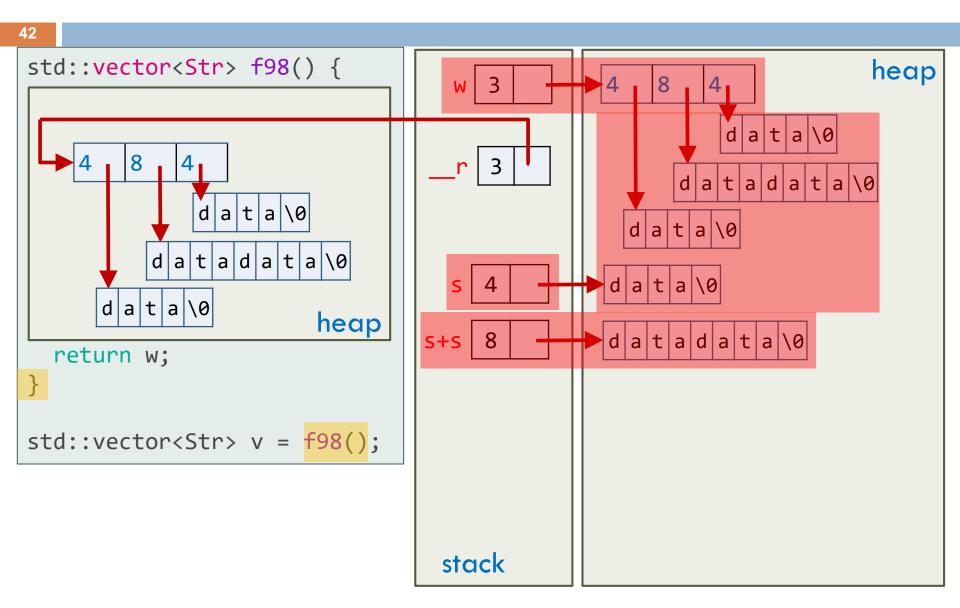
```
Str operator+(Str const& lhs, Str const& rhs) {
  Str tmp(lhs);
  tmp += rhs;
  return tmp;
template <typename T>
class vector {
public:
 // insert a copy for elem
  void push_back(T const& elem);
```

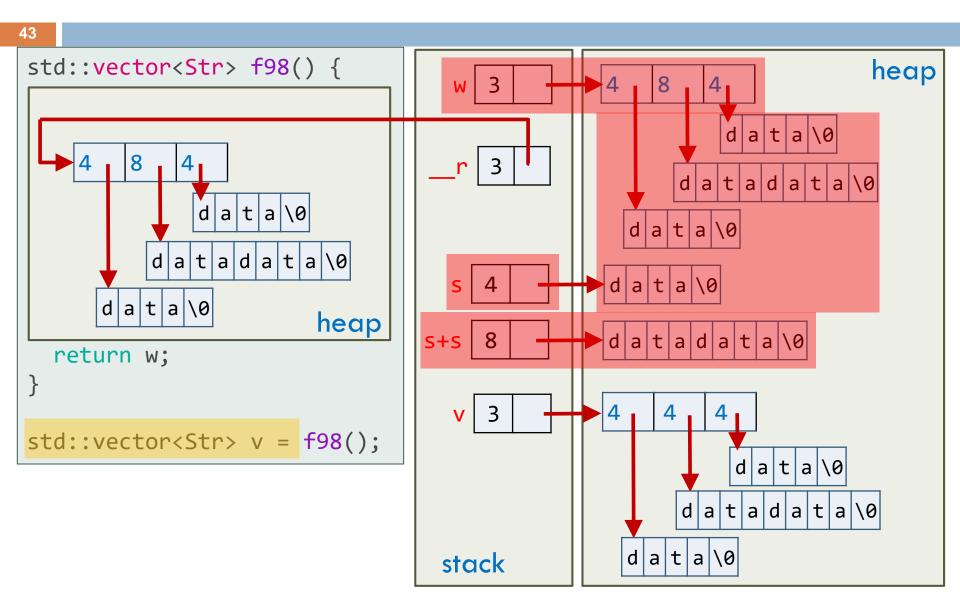
```
std::vector<Str> f98() {
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
 w.push_back(s+s);
 w.push_back(s);
  return w;
std::vector<Str> v = f98();
```

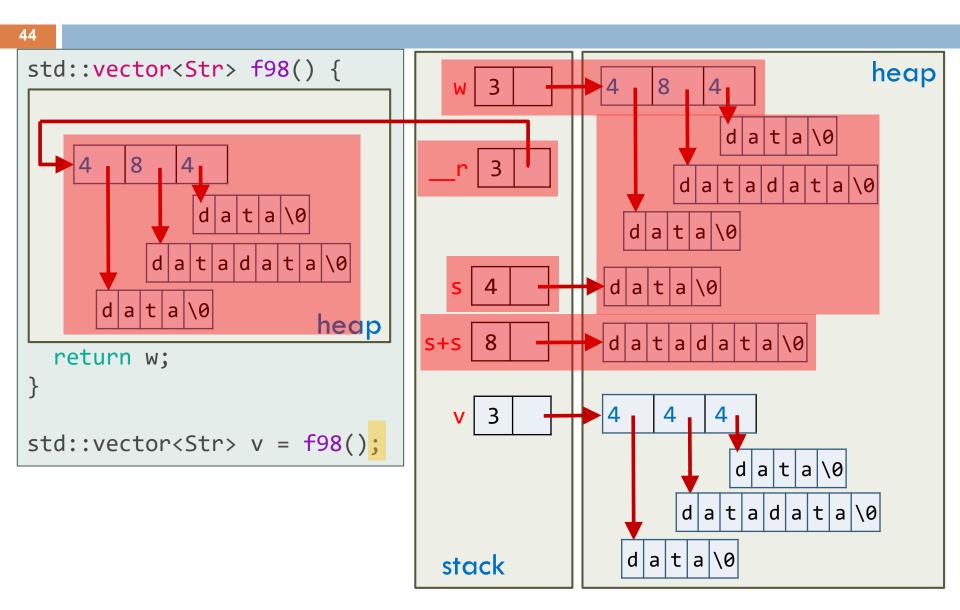
40











```
45
void f98(vector<Str>& __r) {
                                                                    heap
                                     0
  __r.vector<Str>();
  _r.reserve(3);
  Str s = "data";
  __r.push_back(s);
  __r.push_back(s+s);
  __r.push_back(s);
                                               d a t a \0
  return;
std::vector<Str> v; // no ctor
f98(v);
                                  stack
```

46

```
void f98(vector<Str>& __r) {
                                                                     heap
  __r.vector<Str>();
  __r.reserve(3);
  Str s = "data";
  __r.push_back(s);
                                                 d a t a \0
  __r.push_back(s+s);
  __r.push_back(s);
                                               d a t a \0
  return;
std::vector<Str> v; // no ctor
f98(v);
                                  stack
```

```
void f98(vector<Str>& __r) {
                                                                 heap
 __r.vector<Str>();
 __r.reserve(3);
 Str s = "data";
 __r.push_back(s);
                                              data\0
 __r.push_back(s+s);
 __r.push_back(s);
                                             data\0
                                 S
  return;
                                             datadata\0
                              s+s 8
std::vector<Str> v; // no ctor
f98(v);
                                stack
```

```
48
void f98(vector<Str>& __r) {
                                                                heap
                                   2
 __r.vector<Str>();
 __r.reserve(3);
 Str s = "data";
                                                 datadata\0
   _r.push_back(s);
                                             data\0
 __r.push_back(s+s);
  __r.push_back(s);
                                            data\0
                                S
  return;
                                            datadata\0
                              s+s 8
std::vector<Str> v; // no ctor
f98(v);
                               stack
```

```
49
void f98(vector<Str>& __r) {
                                                                heap
                                   2
 __r.vector<Str>();
 __r.reserve(3);
 Str s = "data";
                                                 datadata\0
 __r.push_back(s);
                                             data\0
 __r.push_back(s+s);
 __r.push_back(s);
                                            data\0
                                S
  return;
                                            datadata\0
std::vector<Str> v; // no ctor
f98(v);
                               stack
```

50

```
void f98(vector<Str>& __r) {
                                                               heap
                                   3
                                                8
 __r.vector<Str>();
 __r.reserve(3);
                                                     data\0
 Str s = "data";
                                                 datadata\0
 __r.push_back(s);
                                             data\0
 __r.push_back(s+s);
 __r.push_back(s);
                                            data\0
                                S
  return;
                                            datadata\0
std::vector<Str> v; // no ctor
f98(v);
                               stack
```

```
51
void f98(vector<Str>& __r) {
                                                                heap
                                   3
                                                8
 __r.vector<Str>();
 __r.reserve(3);
                                                     data\0
 Str s = "data";
                                                 datadata\0
 __r.push_back(s);
                                             data\0
 __r.push_back(s+s);
 __r.push_back(s);
                                            data\0
  return;
                                            datadata\0
                              s+s 8
std::vector<Str> v; // no ctor
f98(v);
                               stack
```

52

```
void f98(vector<Str>& __r) {
                                                                heap
                                                8
 __r.vector<Str>();
 __r.reserve(3);
                                                     data\0
 Str s = "data";
                                                 datadata\0
 __r.push_back(s);
                                             data\0
 __r.push_back(s+s);
 __r.push_back(s);
  return;
                                            datadata\0
std::vector<Str> v; // no ctor
f98(v);
                               stack
```

53

```
std::vector<Str> f() {
                                                                    heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
  w.push_back(s+s);
  w.push_back(s);
  return w;
std::vector<Str> v;
v = f();
                                  stack
```

```
54
std::vector<Str> f() {
                                                                     heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
  w.push_back(s+s);
  w.push_back(s);
                                                d a t a \0
  return w;
std::vector<Str> v;
V = f();
                                  stack
```

```
55
std::vector<Str> f() {
                                                                    heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
                                                 data\0
  w.push_back(s+s);
  w.push_back(s);
                                               d a t a \0
  return w;
std::vector<Str> v;
V = f();
                                  stack
```

```
56
std::vector<Str> f() {
                                                                 heap
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                              data\0
 w.push_back(s+s);
 w.push_back(s);
                                             data\0
  return w;
                                             datadata\0
                               S+S
std::vector<Str> v;
                                      X
V = f();
                                stack
```

```
57
std::vector<Str> f() {
                                                                   heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
                                                     |a|t|a|d|a|t|a|\0|
                                                    d
  w.push_back(s);
                                                data\0
  w.push_back(s+s);
  w.push_back(s);
                                              data\0
  return w;
                                              datadata\0
                               S+S
std::vector<Str> v;
                                       X
V = f();
                                 stack
```

```
58
std::vector<Str> f() {
                                                                    heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
                                                      |a|t|a|d|a|t|a|\0|
                                                     d
  w.push_back(s);
                                                data\0
  w.push_back(s+s);
  w.push_back(s);
                                               d a t a \0
  return w;
                                               datadata
std::vector<Str> v;
                                        X
V = f();
                                  stack
```

```
59
std::vector<Str> f() {
                                                                  heap
                                                  8
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
                                               data\0
  w.push_back(s+s);
  w.push_back(s);
                                              d a t a \0
  return w;
                                      X
                                              datadata\0
                               S+S
std::vector<Str> v;
                                       X
V = f();
                                 stack
```

```
60
std::vector<Str> f() {
                                                                   heap
                                                  8
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
                                               data\0
  w.push_back(s+s);
 w.push_back(s);
                                              d a t a \0
  return w;
                                              datadata\0
std::vector<Str> v;
                                       X
V = f();
                                 stack
```

```
61
std::vector<Str> f() {
                                                                    heap
                                                   8
  std::vector<Str> w;
  w.reserve(3);
                                                          |a|t|a|\0|
  Str s = "data";
  w.push_back(s);
                                                data\0
  w.push_back(s+s);
  w.push_back(s);
                                               d a t a \0
                                  S
  return w;
                                               datadata\0
std::vector<Str> v;
                                        X
V = f();
                                  stack
```

```
62
std::vector<Str> f() {
                                                                   heap
                                                  8
  std::vector<Str> w;
  w.reserve(3);
                                                         |a|t|a|\0|
  Str s = "data";
  w.push_back(s);
                                                data\0
  w.push_back(s+s);
  w.push_back(s);
  return w;
                                              datadata\0
std::vector<Str> v;
                                       X
V = f();
                                 stack
```

```
63
std::vector<Str> f() {
                                                                 heap
                                                 8
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                              data\0
 w.push_back(s+s);
 w.push_back(std::move(s));
                                      X
                                             data\0
  return w;
                                             datadata\0
std::vector<Str> v;
                                      X
V = f();
                                stack
```

```
64
std::vector<Str> f() {
                                                                 heap
                                                 8
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                              data\0
 w.push_back(s+s);
 w.push_back(std::move(s));
                                             data\0
  return w;
                                             datadata\0
std::vector<Str> v;
                                      X
V = f();
                                stack
```

```
65
std::vector<Str> f() {
                                                                  heap
                                                  8
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                               data\0
 w.push_back(s+s);
 w.push_back(std::move(s));
                                             data\0
  return w;
                                             datadata\0
std::vector<Str> v;
                                                  8
                                                      4
                                                          |t|a|\0|
                                                         a
V = f();
                                                   datadata\0
                                                a t a \0
                                stack
```

```
66
std::vector<Str> f() {
                                                                  heap
  std::vector<Str> w;
  w.reserve(3);
  Str s = "data";
  w.push_back(s);
  w.push_back(s+s);
 w.push_back(std::move(s));
                                              data\0
  return w;
                                              datadata
std::vector<Str> v;
                                                  8
                                                      4
                                                           |t|a|\0|
V = f();
                                                   datadata\0
                                                 a t a \0
                                 stack
```

```
67
std::vector<Str> f() {
                                                                 heap
                                                 8
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                              data\0
 w.push_back(s+s);
 w.push_back(std::move(s));
                                             data\0
  return w;
                                             datadata\0
std::vector<Str> v;
v = f();
                                stack
```

```
68
std::vector<Str> f() {
                                                                 heap
                                                 8
  std::vector<Str> w;
 w.reserve(3);
 Str s = "data";
 w.push_back(s);
                                              data\0
 w.push_back(s+s);
 w.push_back(std::move(s));
                                             data\0
  return w;
                                             datadata\0
std::vector<Str> v;
V = f();
                                stack
```

Next Lecture(s)

- Rvalue references
- Move constructors
- Move assignments
- □ std::move
- □ std::swap
- std::forward
- Smart pointers