References, Functions, and Parameter Passing

References

- Pointer syntax is confusing

  ```
  *(*(&p->(*x).y)) = 3;  // huh?
  ```

- But pointers are essential for managing memory and writing efficient code
- Wouldn't it be nice if there were a way to use pointers without all of the confusing syntax?
- References let us to do exactly that!
- A reference variable is declared using & instead of *
- A reference stores a memory address, just like a pointer, but dereferencing a reference requires no special syntax (the compiler automatically dereferences it for you)

References

```
struct Student { long id; string name; };

Student student;
long * id = &student.id;
string * name = &student.name;
*id = GetNextID();
*name = "Fred Flintstone";

// IS THE SAME AS

Student student;
long & id = student.id;
string & name = student.name;
id = GetNextID();
name = "Fred Flintstone";
```

References

- The major disadvantage of references compared to pointers is that a reference variable must be initialized with a valid address when it's declared, and can never be changed to point at anything else

```
// THIS WILL WORK

Student fred, barney;
long * id = &fred.id;
string * name = &fred.name;
*id = 32;
*name = "fred";
id = &barney.id;
name = &barney.name;
*id = 33;
*name = "barney";
```

```
// THIS COMPILES, BUT DOESN'T
// HAVE THE INTENDED EFFECT

Student fred, barney;
long & id = fred.id;
string & name = fred.name;
id = 32;
name = "fred";
id = barney.id;
name = barney.name;
id = 33;
name = "barney";
```

References

- A reference can be converted into a pointer using the address-of operator &
- When applied to a reference, & returns the address stored in the reference, not the address of the reference itself

```
int x;
int & ref = x;
int * ptr1 = &x;
int * ptr2 = &ref;

// ptr1 == ptr2
```

Returning References from Functions

```
// THIS IS OK

Student student;

Student & GetStudent() {
    return student;
}

GetStudent().id = 32;
GetStudent().name = "fred";
```

```
// DISASTER WAITING TO HAPPEN

Student & GetStudent() {
    Student student;
    return student;
}

GetStudent().id = 32;
GetStudent().name = "fred";
```

Parameter Passing Modes

- Pass by value
- Pass by pointer
- Pass by reference

Pass By Value

- "Pass by value" makes a new copy of the parameter value on the runtime stack
- If the called function modifies the parameter's value, the value of the actual parameter is not affected

```
void DoIt(int x) {
   x = -99;
}

int main() {
   int j = 32;
   DoIt(j);
   cout << "j = " << j << endl;
   return 0;
}
```

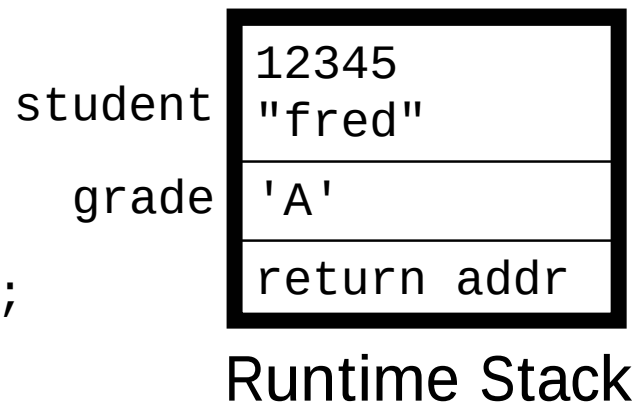
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

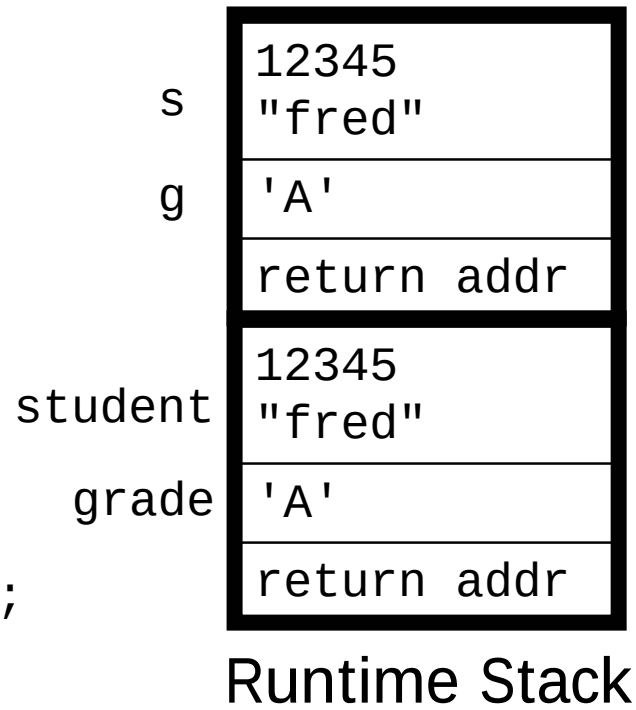
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

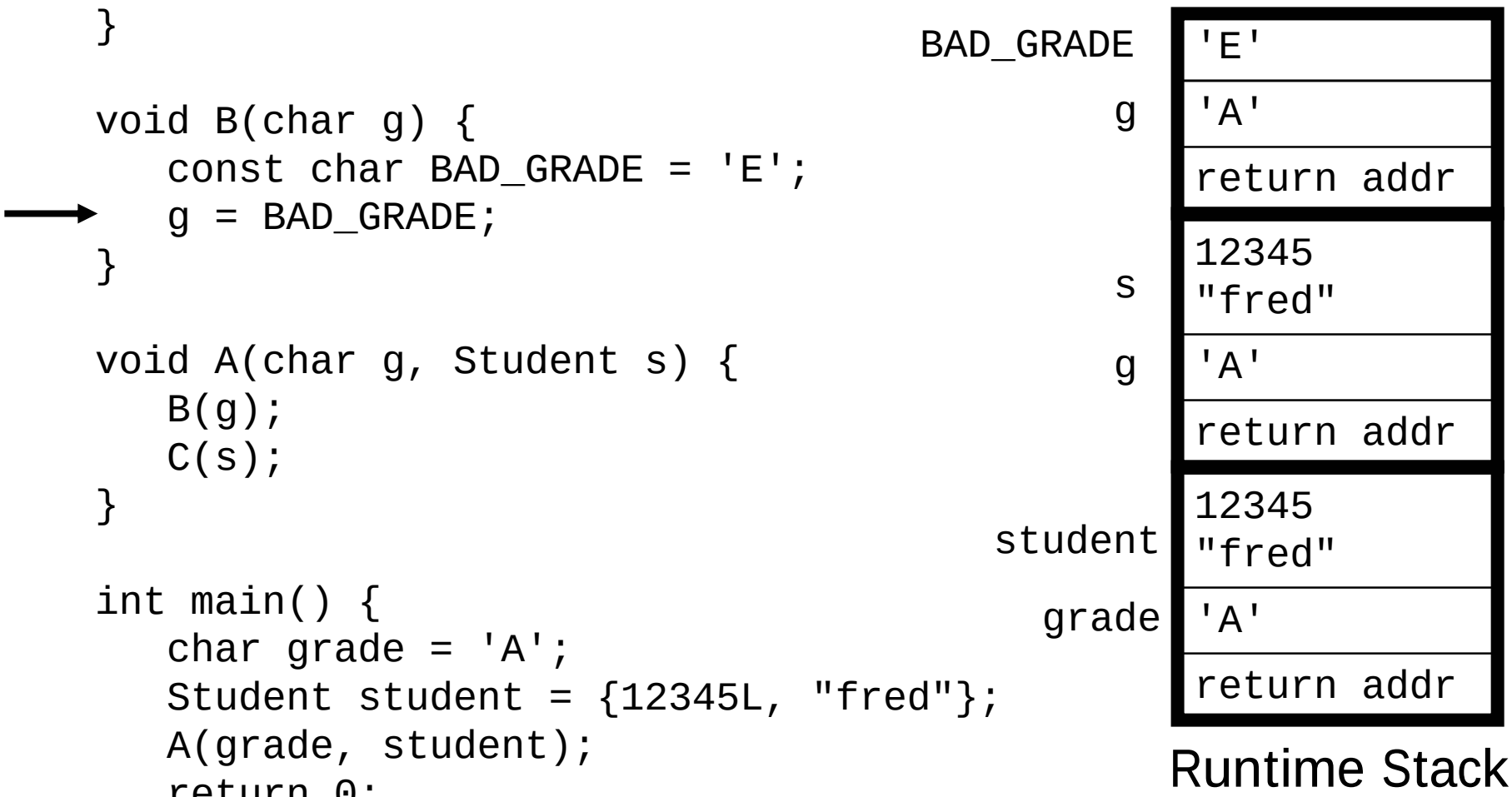
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



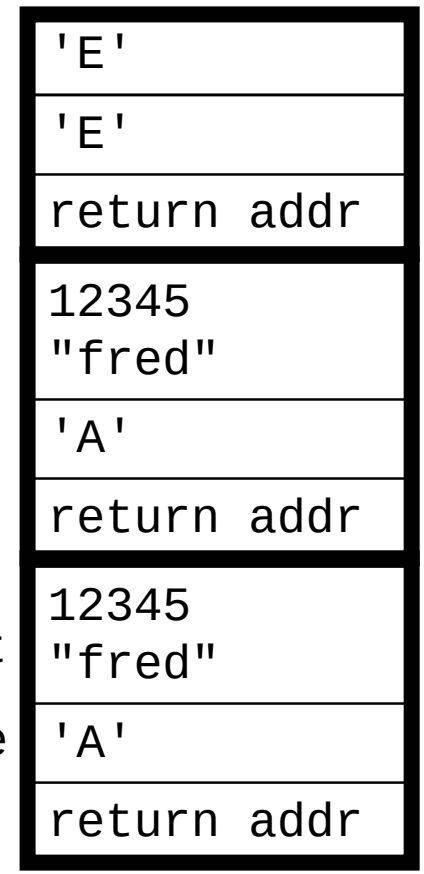
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

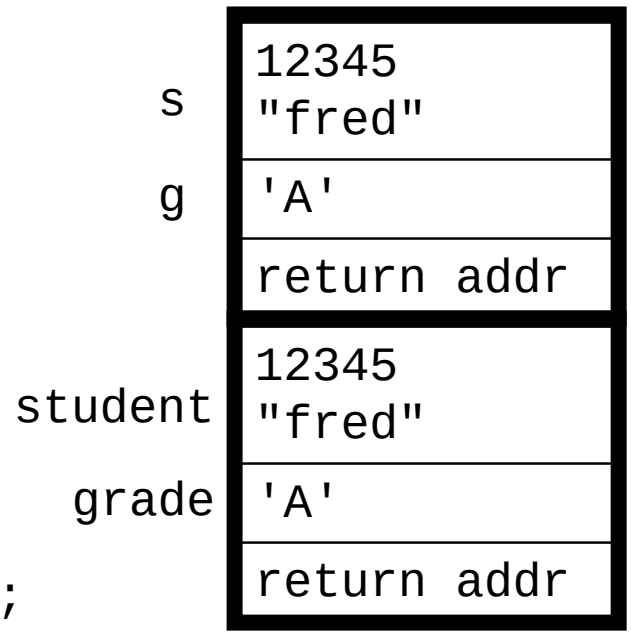
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

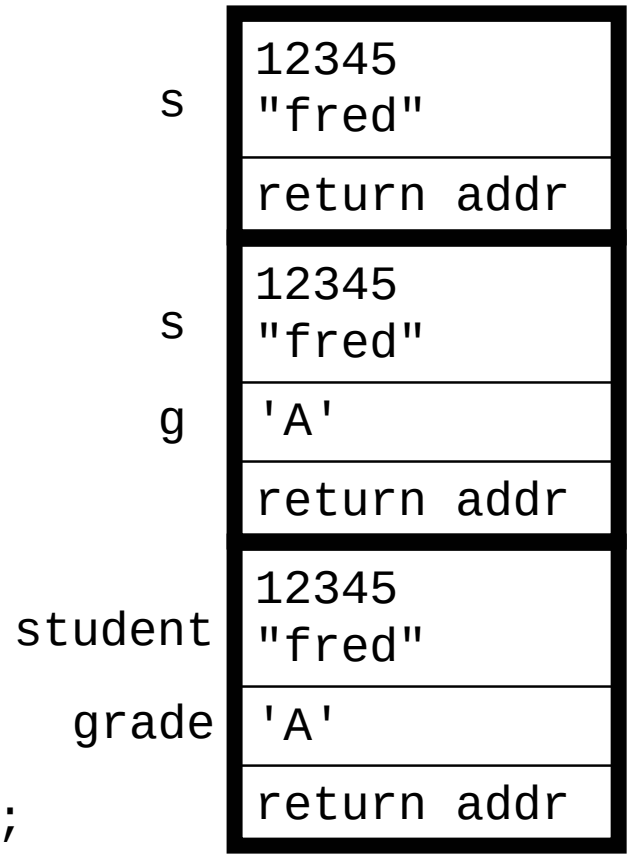
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

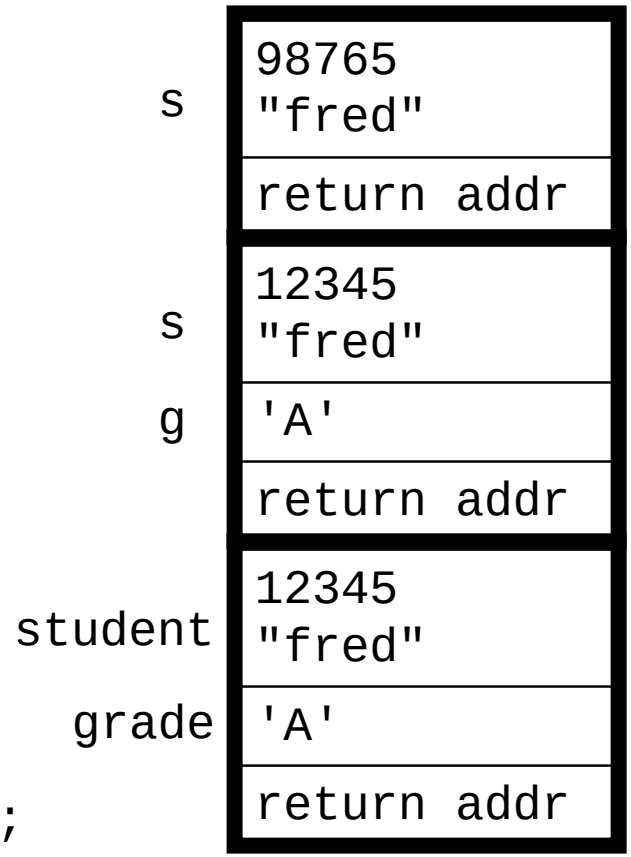
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

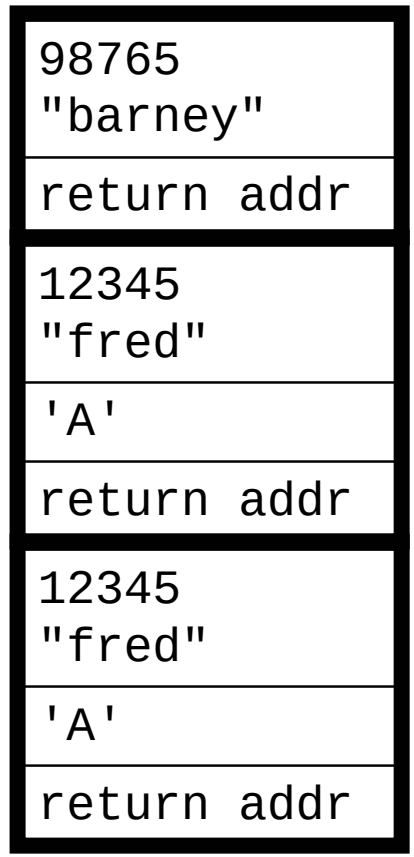
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

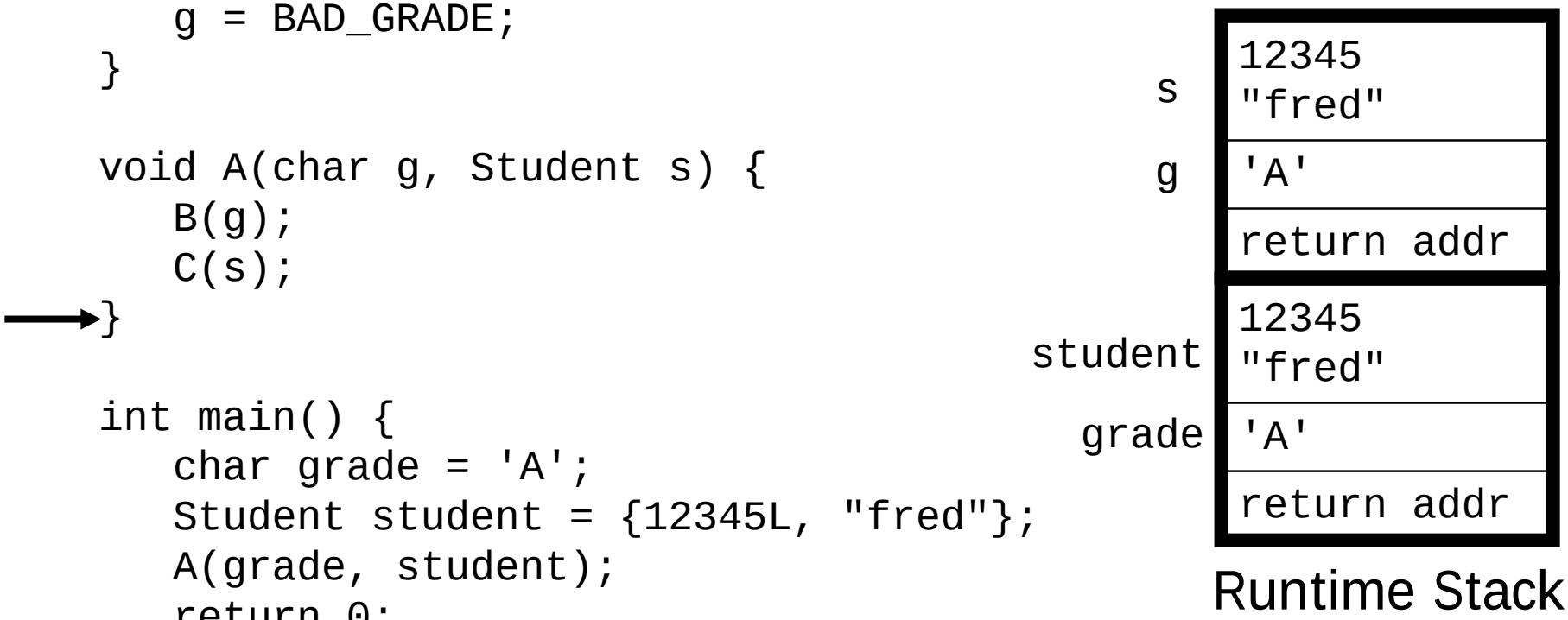
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



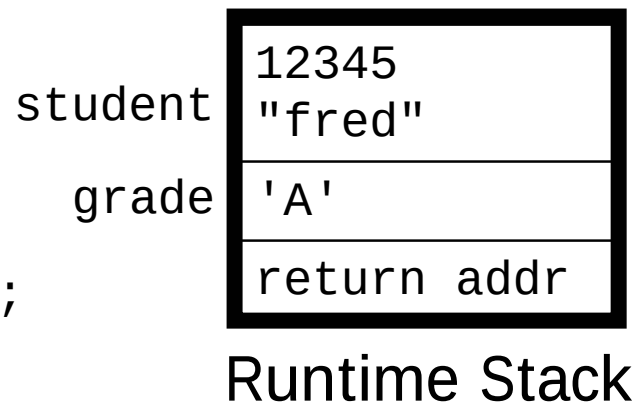
```
struct Student { long id; string name; };

void C(Student s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char g, Student s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Pass By Pointer

- "Pass by pointer" allows the function to change the value of the actual parameter
- Instead of passing the actual parameter's value, pass its address (or pointer)
- The function can change the actual parameter's value through the pointer

```
void DoIt(int * x) {
   *x = -99;
}

int main() {
   int j = 32;
   DoIt(&j);
   cout << "j = " << j << endl;
   return 0;
}
```

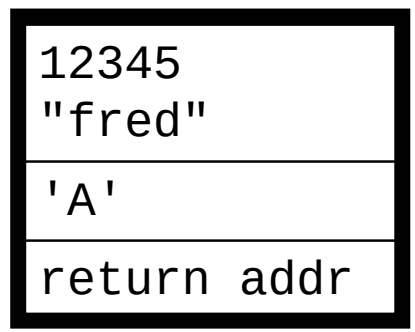
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



Runtime Stack

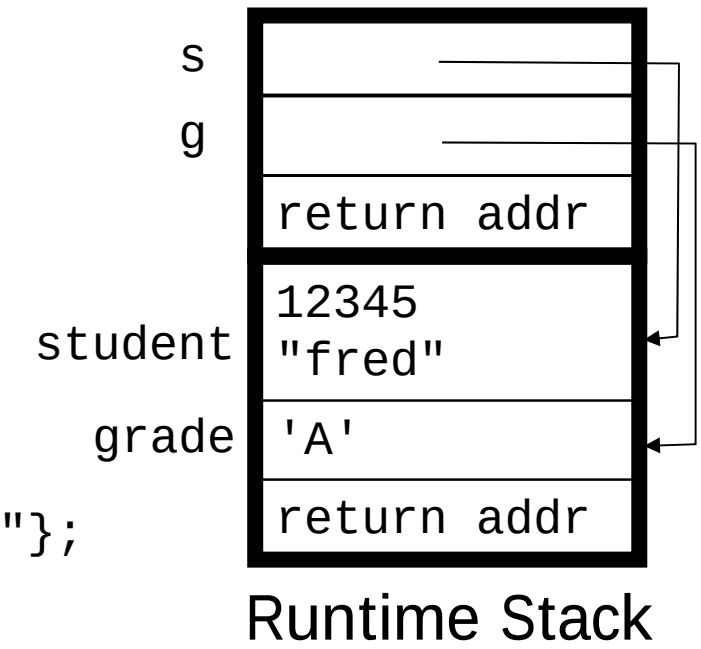
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



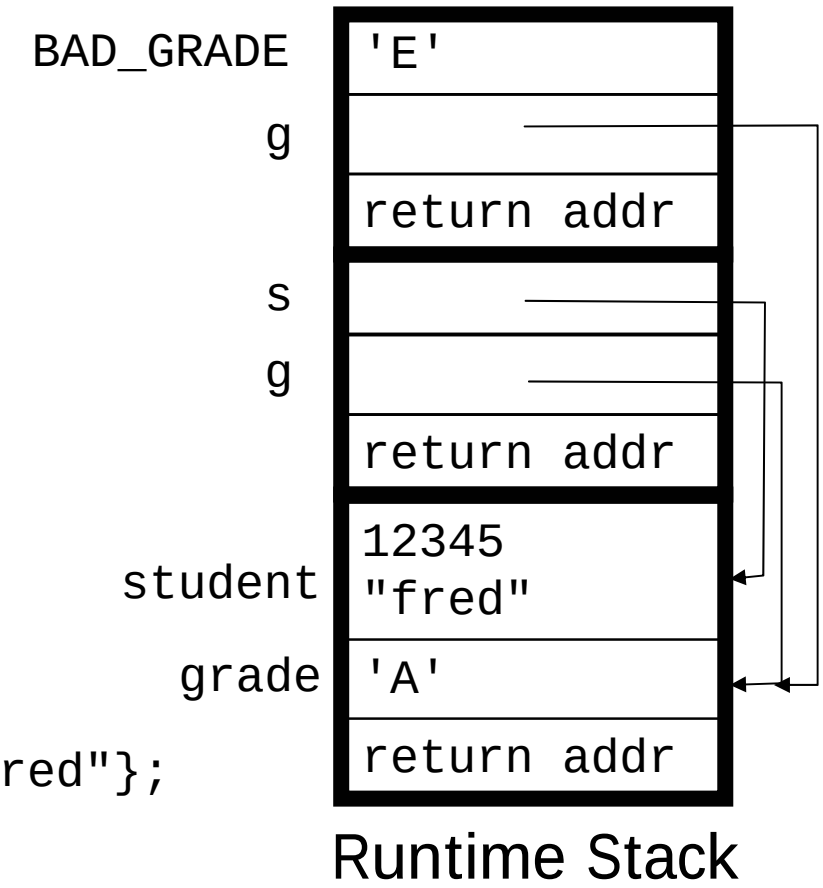
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



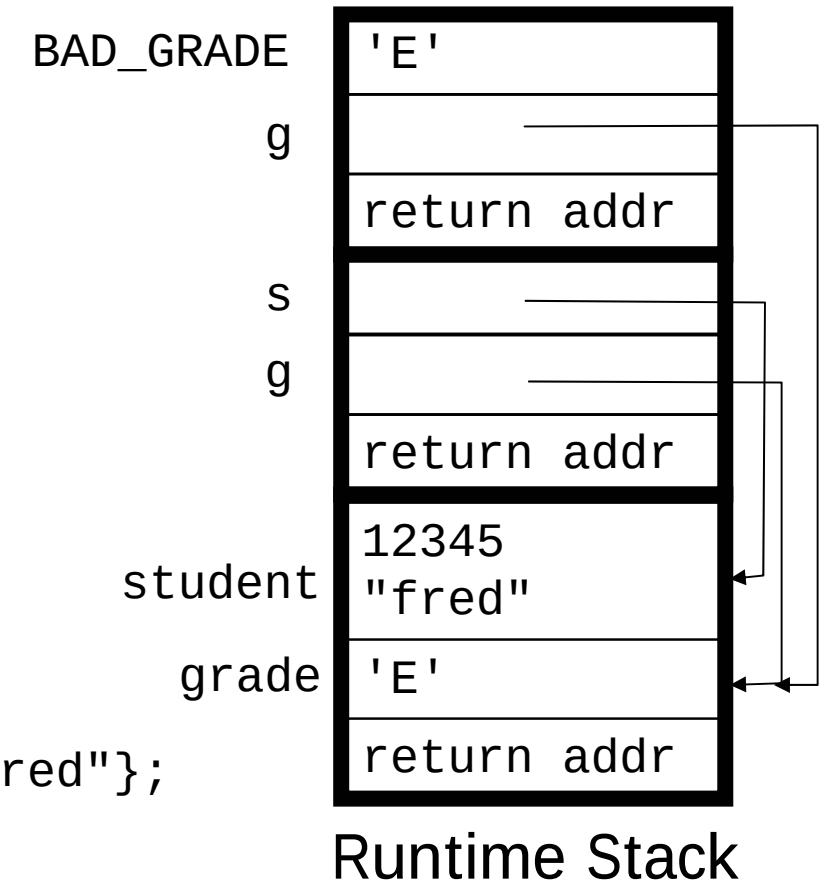
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



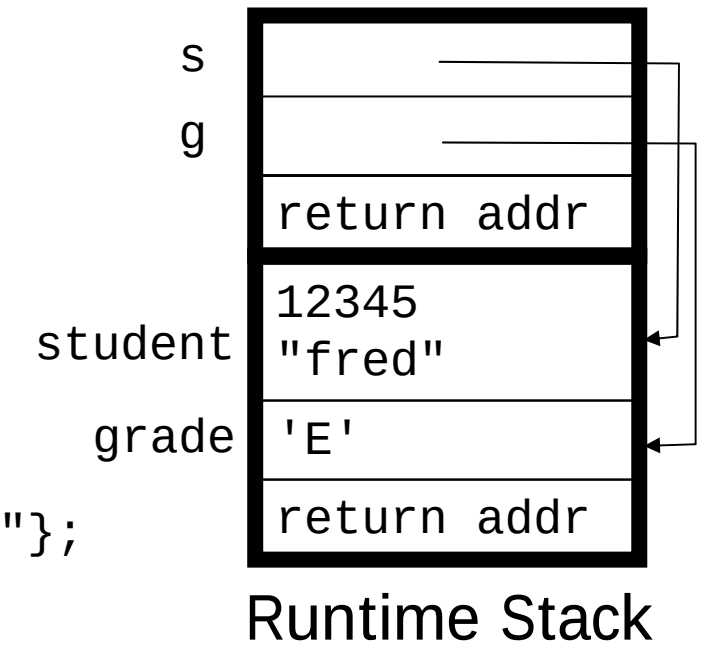
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



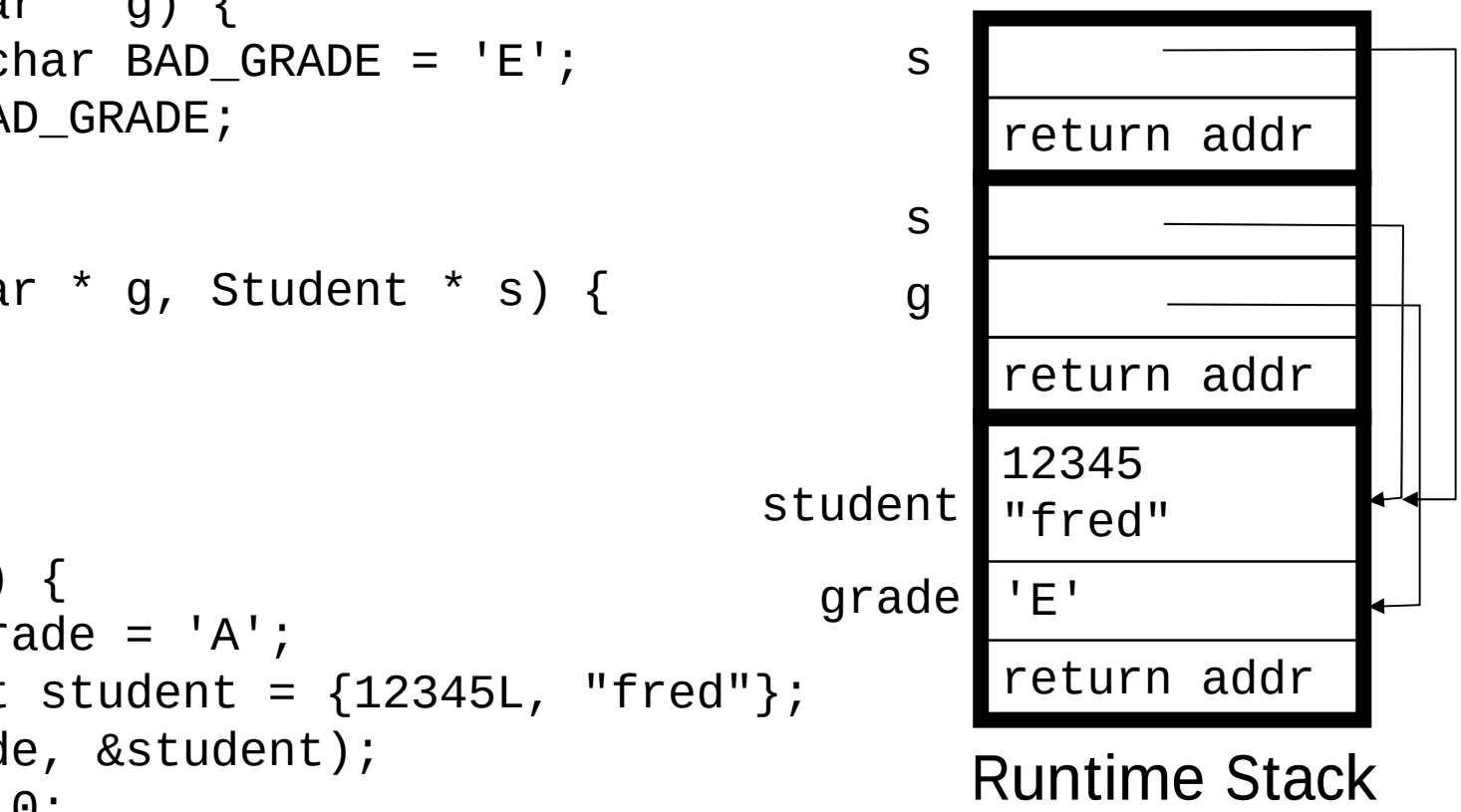
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



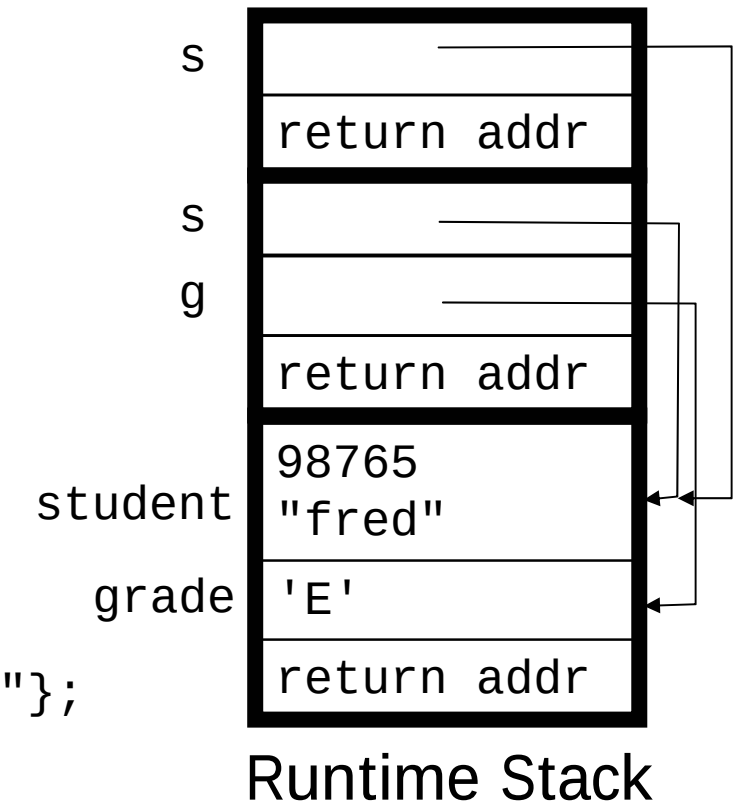
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



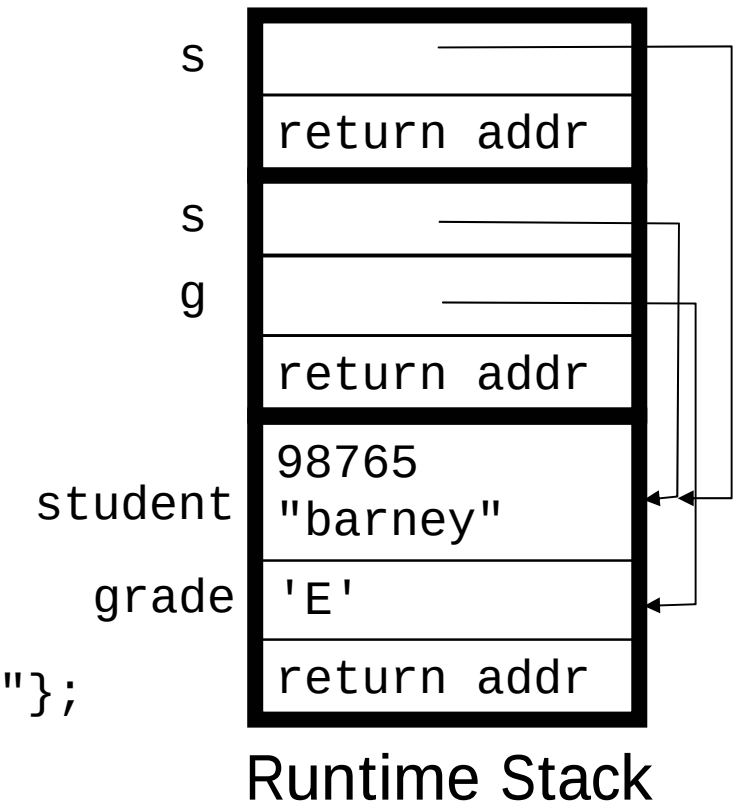
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



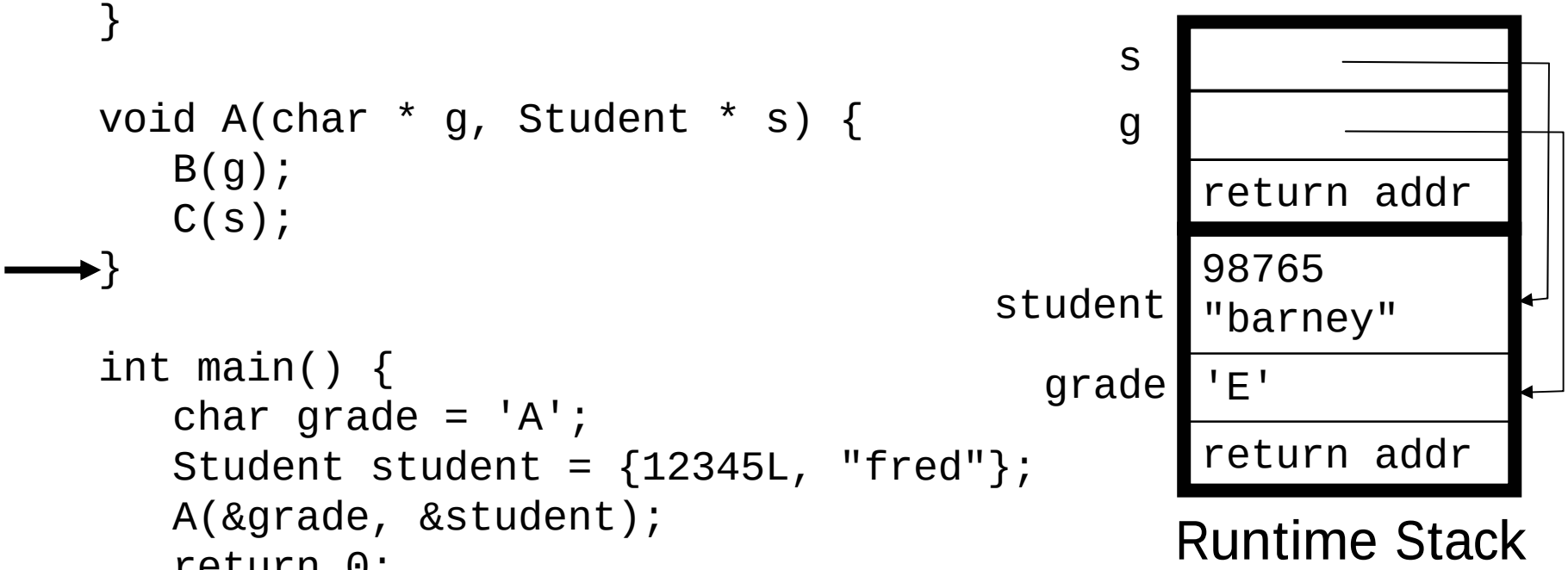
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



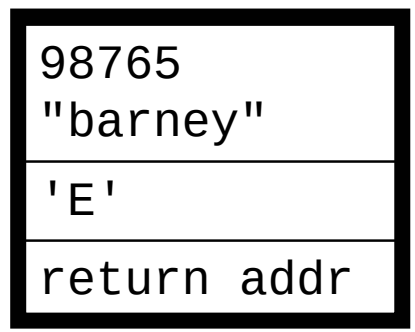
```
struct Student { long id; string name; };

void C(Student * s) {
   s->id = 98765L;
   s->name = "barney";
}

void B(char * g) {
   const char BAD_GRADE = 'E';
   *g = BAD_GRADE;
}

void A(char * g, Student * s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(&grade, &student);
   return 0;
}
```



Runtime Stack

Pass By Reference

- A disadvantage of "pass by pointer" is that the syntax required to pass and dereference pointers is cumbersome (lots of &, *, ->)
- "Pass by reference" works the same as "pass by pointer", but the syntax is more convenient

```
void DoIt(int & x) {
   x = -99;
}

int main() {
   int j = 32;
   DoIt(j);
   cout << "j = " << j << endl;
   return 0;
}
```

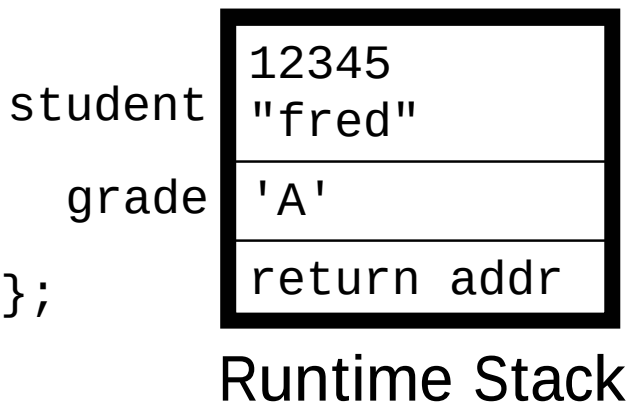
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

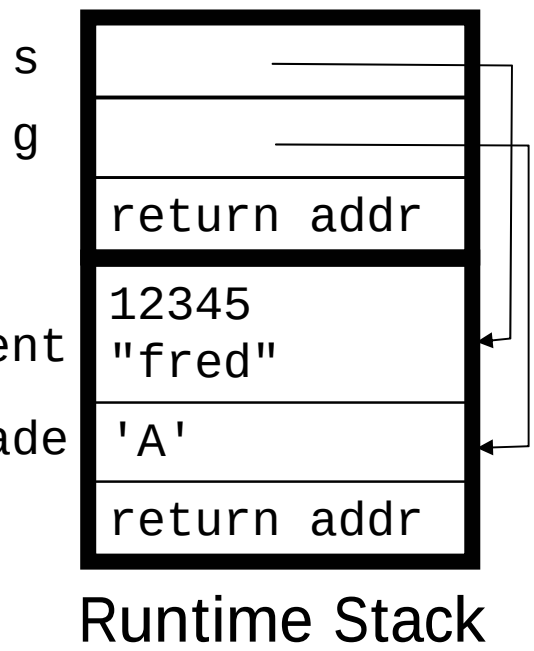
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



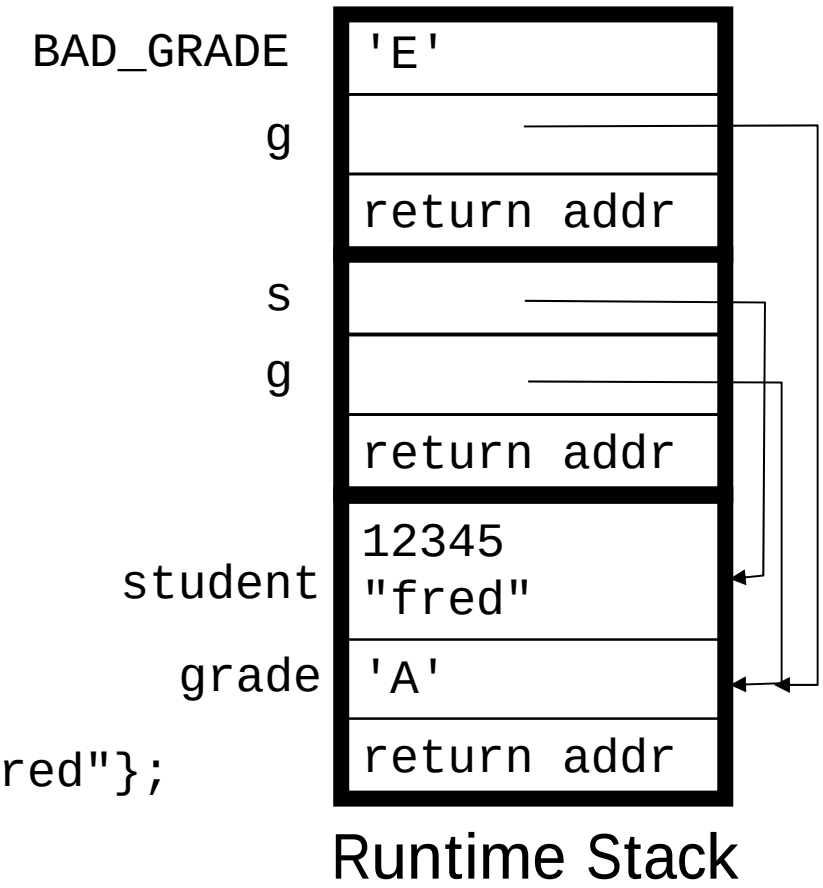
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



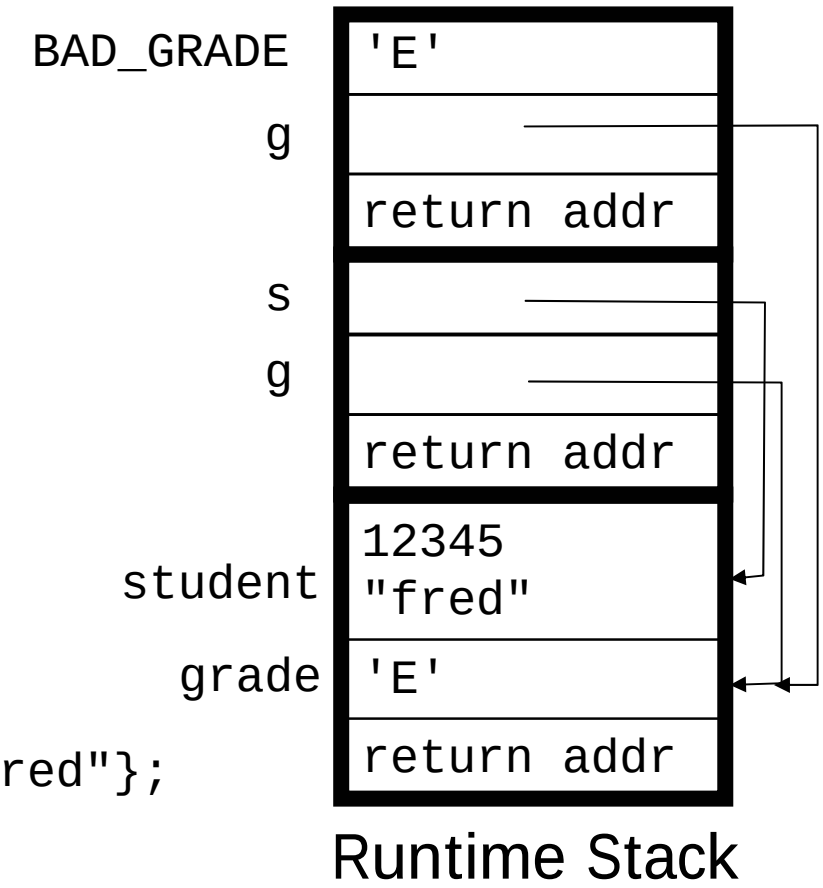
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



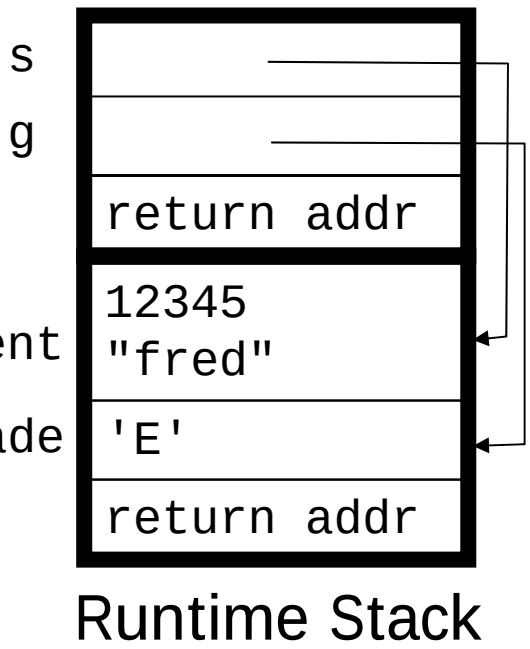
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



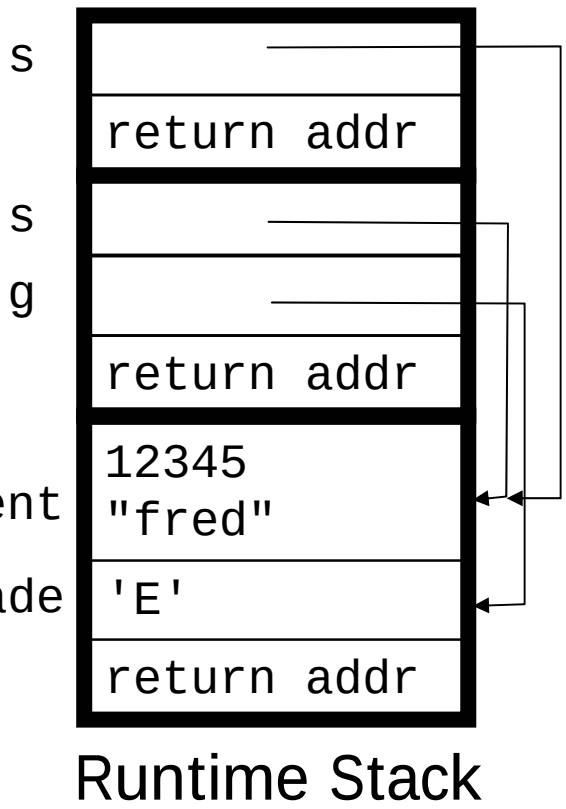
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



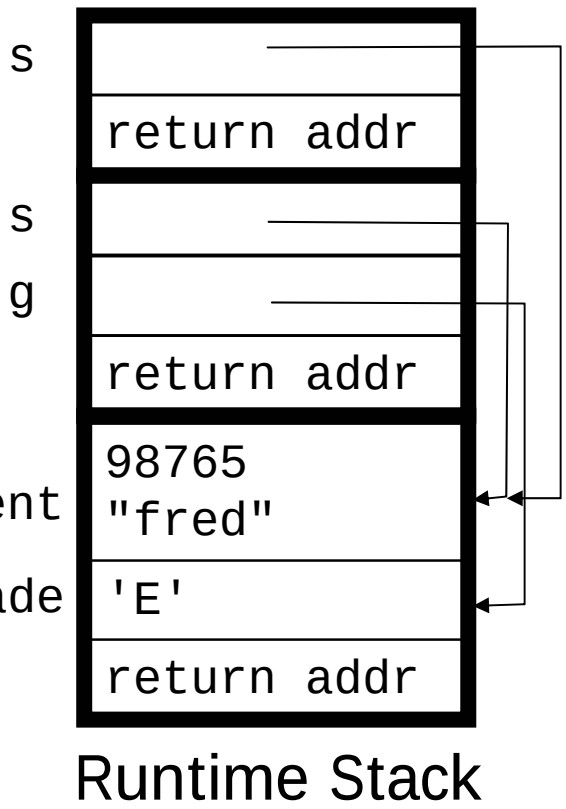
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



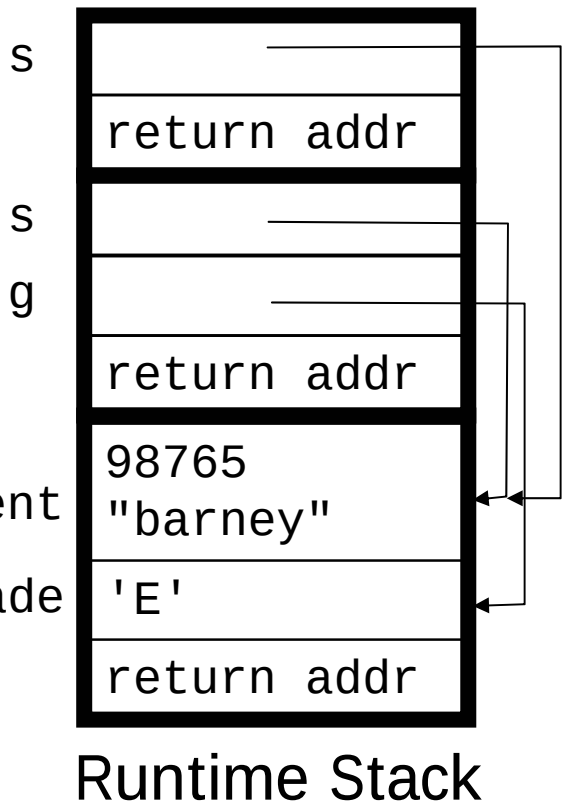
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



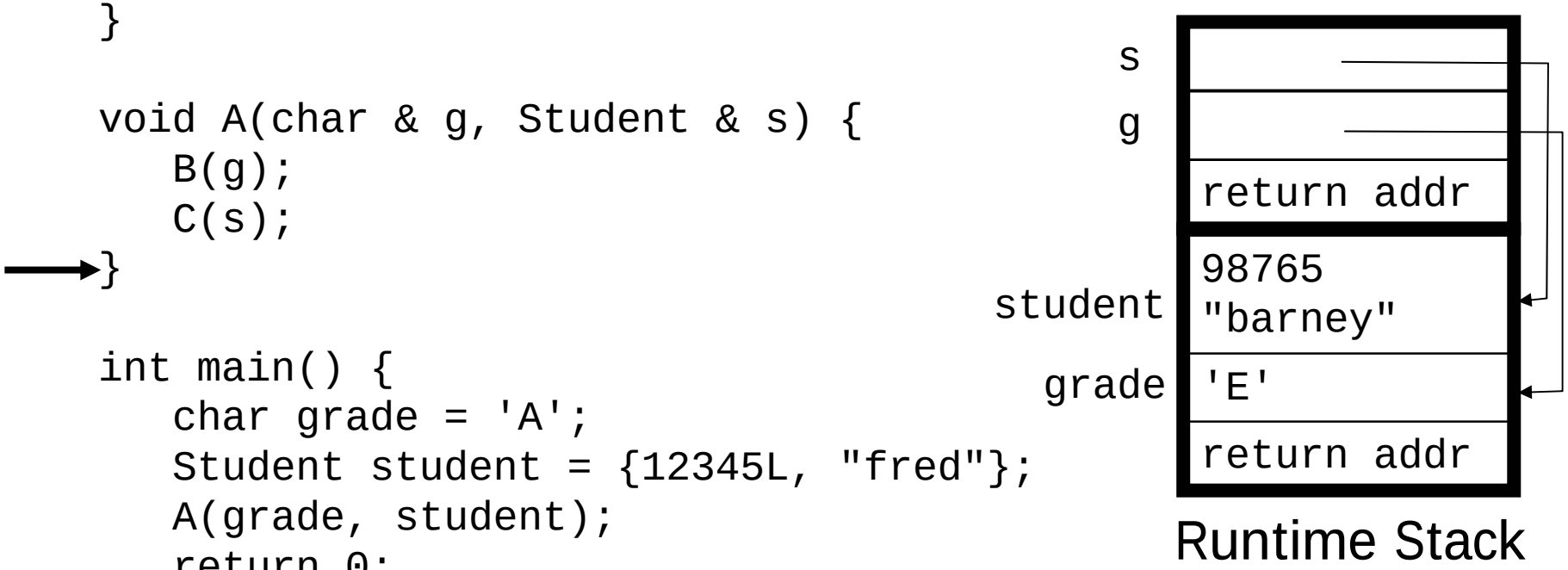
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



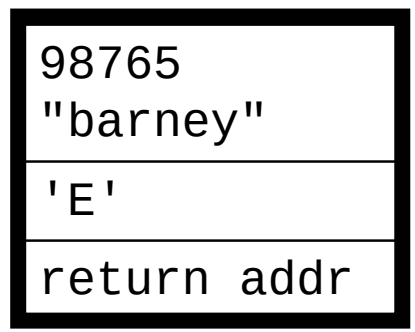
```
struct Student { long id; string name; };

void C(Student & s) {
   s.id = 98765L;
   s.name = "barney";
}

void B(char & g) {
   const char BAD_GRADE = 'E';
   g = BAD_GRADE;
}

void A(char & g, Student & s) {
   B(g);
   C(s);
}

int main() {
   char grade = 'A';
   Student student = {12345L, "fred"};
   A(grade, student);
   return 0;
}
```



Runtime Stack

Array Parameters

- In the case of large structures, arrays, and objects, passing by pointer or reference is much faster than passing by value (remember, pass by value makes a copy)
- For this reason, C++ always passes arrays by pointer

```
struct Student { long id; string name; };

void Initialize(Student s[5]) {
   for (int x=0; x < 5; ++x) {
      s[x].id = 0L;
      s[x].name = "";
   }
}

int main() {
   Student students[5];
   Initialize(students);
   return 0;
}
```

Array Parameters

```
void Initialize(Student s[5]) {
   for (int x=0; x < 5; ++x) {
      s[x].id = 0L;
      s[x].name = "";
   }
}

void Initialize(Student s[], int length) {
   for (int x=0; x < length; ++x) {
      s[x].id = 0L;
      s[x].name = "";
   }
}

void Initialize(Student * s, int length) {
   for (int x=0; x < length; ++x) {
      s[x].id = 0L;
      s[x].name = "";
   }
}
```

Const Parameters

- Arrays are automatically passed by pointer, but structures and objects are not
- Large structures and objects should be passed by pointer or reference
  - If the function needs to modify the structure or object, you need pass by pointer or reference anyway
  - If the function should not modify the structure or object, mark the parameter as "const" to prevent the function from changing it

```
void PrintStudent(const Student & s) {
   cout << "id: " << s.id << ", name: " << s.name << endl;
}

void PrintStudent(const Student * s) {
   cout << "id: " << s->id << ", name: " << s->name << endl;
}
```

Function Declarations and Definitions

```
// function declaration (or function prototype)
//
float Hypotenuse(float a, float b);


// function definition
//
float Hypotenuse(float a, float b) {
    return sqrt(a*a + b*b);
}
```

Function Declarations and Definitions

- In a CPP file, a function must be declared ABOVE the point at which it is first called
- This can be accomplished in two ways:
  - Place a function prototype somewhere above the first call
  - Place the function definition somewhere above the first call
    - the definition also serves as a declaration

Function Declarations and Definitions

```
// WHY WON'T THIS WON'T COMPILE?

void SheLovesMe(int x) {
    if (x == 0)
        cout << "She loves me!!!" << endl;
    else
        SheLovesMeNot(--x);
}

void SheLovesMeNot(int x) {
    if (x == 0)
        cout << "She loves me - NOT!" << endl;
    else
        SheLovesMe(--x);
}
```

Function Declarations and Definitions

```
// THIS WORKS

void SheLovesMeNot(int x);

void SheLovesMe(int x) {
    if (x == 0)
        cout << "She loves me!!!" << endl;
    else
        SheLovesMeNot(--x);
}

void SheLovesMeNot(int x) {
    if (x == 0)
        cout << "She loves me - NOT!" << endl;
    else
        SheLovesMe(--x);
}
```