# C++ Programming

Instructor: Rita Kuo
Office: CS 520E
Phone: Ext. 4405
E-mail: rita.kuo@uvu.edu

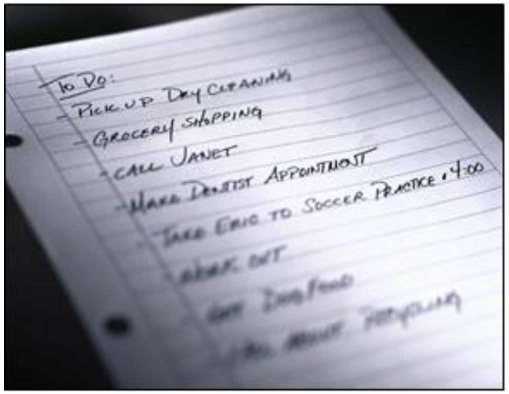
# Mapping zyBooks Chapters

| Topic | zyBooks Chapter |
|---|---|
| Structures | 8.1, 8.2 |
| Arrays of Structures | 8.2 |
| Structures and vectors | 8.3, 8.4 |

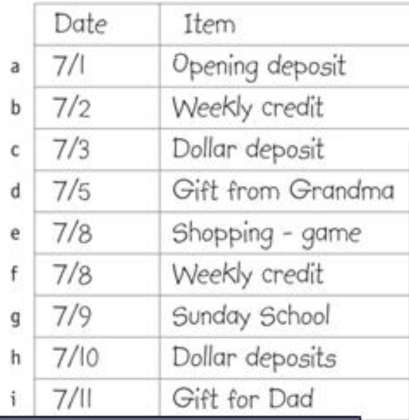
# Struct

# Review - List in Real Life

- It is often the case in computer science that you will process a list of items



To Do:
- Pick up Dry Cleaning
- Grocery Shopping
- Call Janet
- Make Dentist Appointment
- Take Eric to Soccer Practice · 4:00
- Work out
- Get Dog Food
- [illegible] about Recycling



| | Date | Item | In/Out | Balance |
|---|---|---|---|---|
| a | 7/1 | Opening deposit | + 16.00 | $ 16.00 |
| b | 7/2 | Weekly credit | + 3.00 | $ 19.00 |
| c | 7/3 | Dollar deposit | | |
| d | 7/5 | Gift from Grandma | | |
| e | 7/8 | Shopping - game | | |
| f | 7/8 | Weekly credit | | |
| g | 7/9 | Sunday School | | |
| h | 7/10 | Dollar deposits | | |
| i | 7/11 | Gift for Dad | | |

"ONLINE COUPON" Redeemable at participating KFC Restaurants

**3 PC CHICKEN & IND. FRIES $6.98**

Includes 3pcs of chicken, ind. fries.

Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitution. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

**3 PC CHICKEN & IND. FRIES $6.98**

Includes 3pcs of chicken, ind. fries.

Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitution. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

**4 PC CHICKEN & IND. FRIES $8.98**

Includes 4 pcs of chicken, ind. fries.

Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

**TWO CAN DINE BIG CRUNCH® OR ZINGER® $13.98**

Includes 2 Big Crunch® or 2 Zinger® sandwiches, 2 ind. fries and reg. soft drinks.

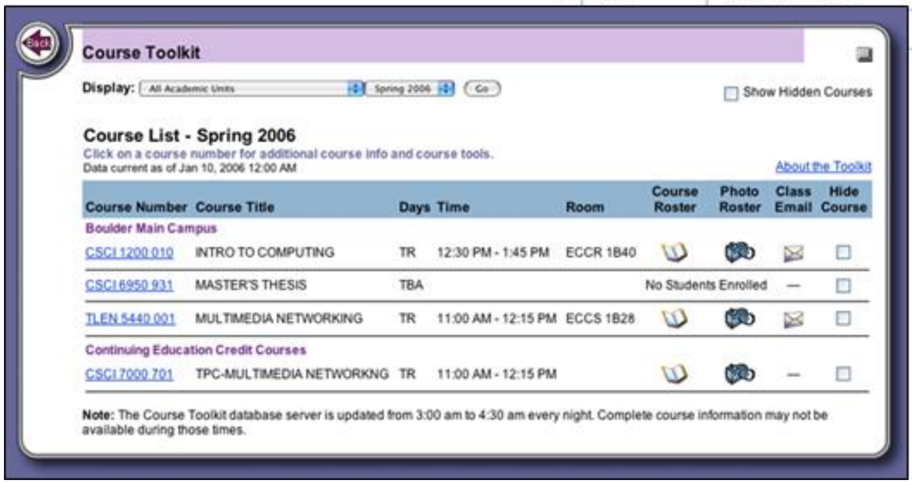
Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Pepsi ®, Pepsi Globe ® - PepsiCo, Inc. Used under license. Expires Oct 6th, 2013.

"ONLINE COUPON" Redeemable at participating KFC Restaurants

**14 PC FAMILY SPECIAL $28.98**

Includes 14 pcs of chicken, lg. fries, and 2 med. salads

Limit of one coupon per customer, per visit. Redeemable at participating KFC Restaurants. Not valid with any other offer. Not valid for on-line orders. Premium charge for white meat where available. Please present coupon when ordering. All orders subject to applicable taxes. No Substitutions. Expires Oct 6th, 2013.



Course Toolkit

Display: All Academic Units | Spring 2006 | Go

□ Show Hidden Courses

Course List - Spring 2006

Click on a course number for additional course info and course tools.

Data current as of Jan 10, 2006 12:00 AM

About the Toolkit

| Course Number | Course Title | Days | Time | Room | Course Roster | Photo Roster | Class Email | Hide Course |
|---|---|---|---|---|---|---|---|---|
| Boulder Main Campus | | | | | | | | |
| CSCI 1200 010 | INTRO TO COMPUTING | TR | 12:30 PM - 1:45 PM | ECCR 1B40 | 📖 | 📷 | ✉ | □ |
| CSCI 6950 931 | MASTER'S THESIS | TBA | | | No Students Enrolled | | — | □ |
| TLEN 5440 001 | MULTIMEDIA NETWORKING | TR | 11:00 AM - 12:15 PM | ECCS 1B28 | 📖 | 📷 | ✉ | □ |
| Continuing Education Credit Courses | | | | | | | | |
| CSCI 7000 701 | TPC-MULTIMEDIA NETWORKNG | TR | 11:00 AM - 12:15 PM | | 📖 | 📷 | — | □ |

Note: The Course Toolkit database server is updated from 3:00 am to 4:30 am every night. Complete course information may not be available during those times.

# Review - Variables

- Types
  - Specify what kind of data it will hold
  - Basic data types are **integers** (`short`, `int`, `long`), **real numbers** (`float` or `double`), or **characters** (`char`).
  - `int`: hold integer values, i.e., whole numbers such as 7, -11, 0, etc.
    - The largest `int` value is typically 2,147,483,647 but can be as small as 32,767
  - `float`: can store numbers with digits after the decimal point, such as 379.125
    - Slower than `int` in arithmetic operation
    - Is often an approximation of the number. E.g., `0.1` in a float variable might be 0.09999999999999987 stored in the system.
  - **Variables must be declared before they can be used**

# Review - Array

- Array declaration
  - Syntax: `type name[length]`
    - `type` is the data type of the array
    - `name` is the name you use to reference the array
    - `length` is the number of elements in the array, which is always one more than the last index of the array
  - Example, create an array to hold a 1000 integers

    ```
    int a[1000];
    ```

  - Another way to initialize array, which makes your code more maintainable, is to use a preprocessor constant

    ```
    const int SIZE = 1000;
    int a[SIZE];
    ```

  - Like other variables in C, declaring them does not initialize them

# Structures

# Structure and Array

- Array
  - All elements of an array have the same type
  - To select an array element, we specify its position (as an index)
- Structure
  - The elements (members) aren’t required to have the same data type
  - The members of a structure have names; to select a particular member, we specify its name, not its position

# Structure

- Define a structure
  - □ struct *structure_name* {

    ```
    struct structure_name {
      member_type1 member_name1;
      member_type2 member_name2;
      ...
    };
    ```

  - □ Example

    ```
    struct point_t {
      double x;
      double y;
    };
    ```

# Structure

- Declaring a structure variable
  - `struct` *structure_name variable_name*;
  - Example

    ```
    struct point_t pt1;
    struct point_t pt2;
    ```
- Initializing Structure Variables
  - Prepare a list of values to be stored in the structure and enclose it in braces
  - Example:

    ```
    struct point_t pt1 = {200.0, 200.0};
    ```
  - The values in the initializer must appear in the same order as the members of the structure.

# Structure

- Accessing Structure Members
  - Structure member operator: ., also called the dot operator
  - Example:
    `cout << "(" << pt1.x << ", " << pt2.x << ")";`
  - Designated Initializers
    `struct point_t pt1 = {.y = 200.0, .x = 100.0};`
- Where to define structures?
  - Generally defined in a header file along with function prototype
  - Can defined them at the top of `.c` file

https://stackoverflow.com/questions/228684/

# Structure

- Create `point.h` with the following codes

```
#ifndef POINT_H
#define POINT_H

struct point_t {
        double x;
        double y;
};

// function prototypes

#endif
```

# Structures as Arguments and Return Values

- Structures as Arguments
  - Pass a structure to a function require making a copy of all members in the structure
  - Example:

point.c

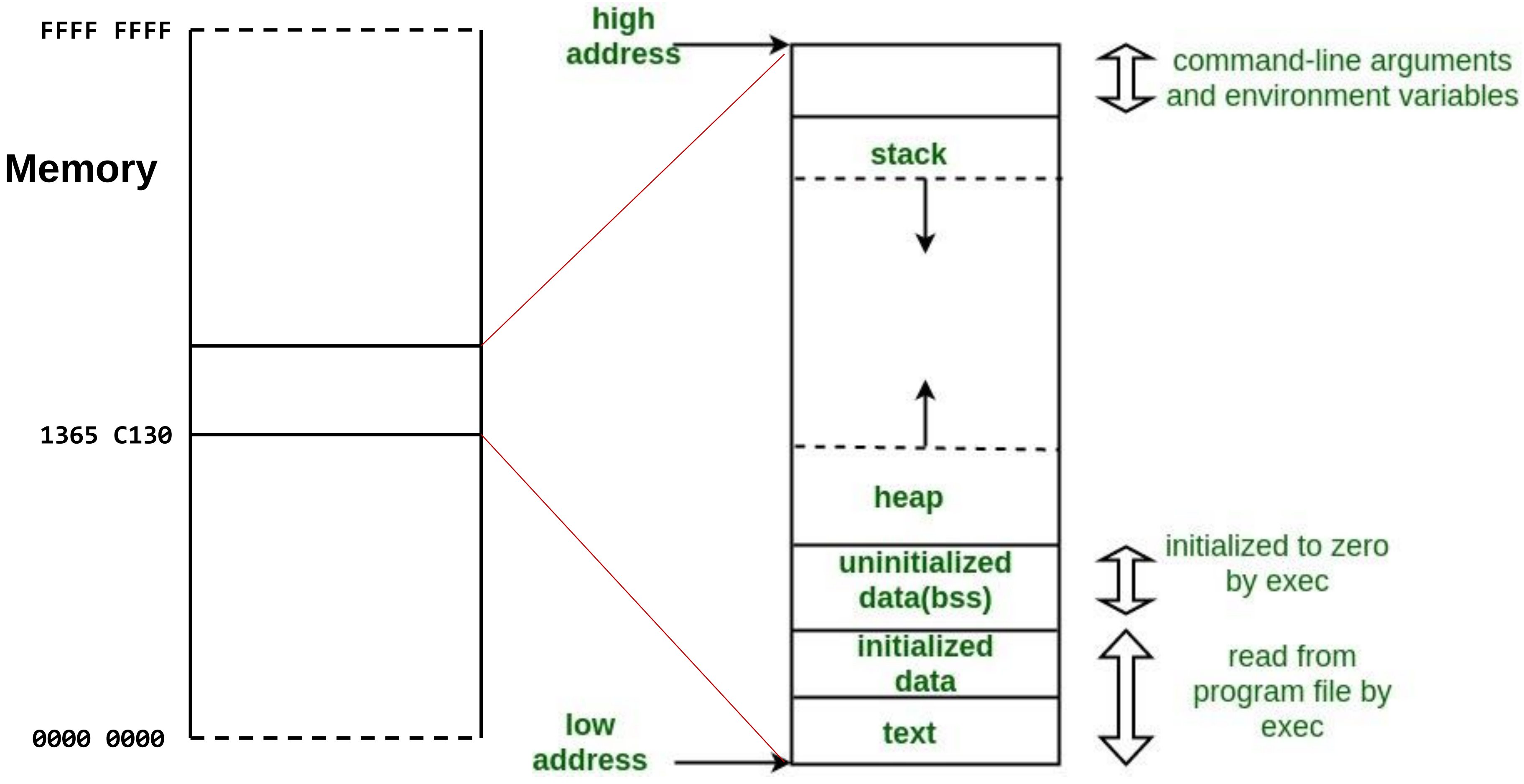
```
double distance(struct point_t p, struct point_t q)
{
        return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}
```

pts.c

```
int main(void)
{
         struct point_t pt1 = {0.0, 0.0};
         struct point_t pt2 = {3.0, 4.0};
         double d;

         d = distance(pt1, pt2);

         return 0;
}
```

# Review - Memory Layout

- Memory
  - Variables correspond to locations in the computer's memory



https://www.geeksforgeeks.org/memory-layout-of-c-program/

# Structures as Arguments and Return Values

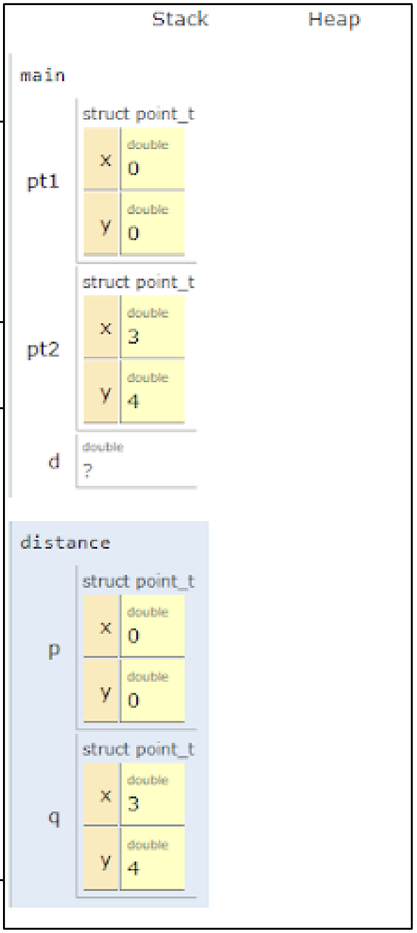
- Structures as Arguments
  - □ Pass a structure to a function require making a copy of all members in the structure
  - □ Example:

point.c

```
double distance(struct point_t p, struct point_t q)
{
        return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}
```

pts.c

```
int main(void)
{
        struct point_t pt1 = {0.0, 0.0};
        struct point_t pt2 = {3.0, 4.0};
        double d;

        d = distance(pt1, pt2);

        return 0;
}
```



# Structures as Arguments and Return Values

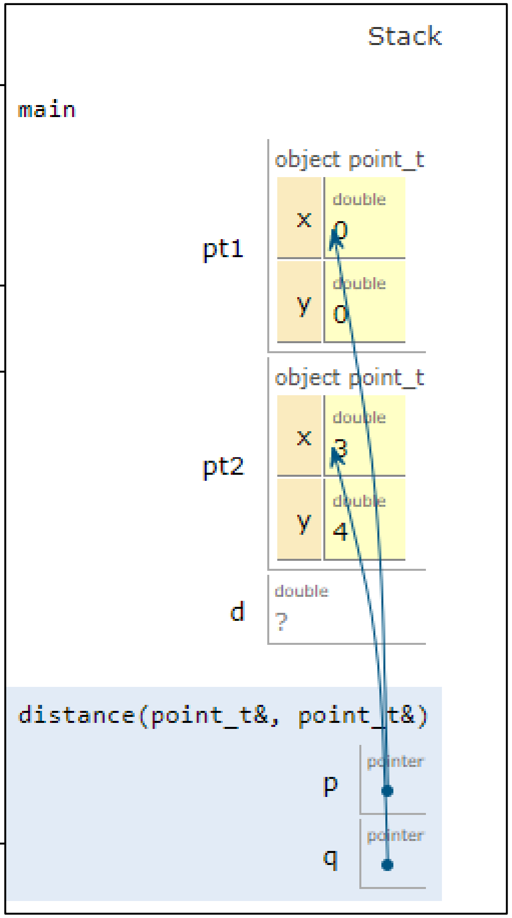
- Structures as Arguments
  - Pass the `struct` parameters by references

point.c

```c
double distance(struct point_t &p, struct point_t &q)
{
        return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}
```

pts.c

```c
int main(void)
{
         struct point_t pt1 = {0.0, 0.0};
         struct point_t pt2 = {3.0, 4.0};
         double d;

         d = distance(pt1, pt2);

         return 0;
}
```



# Structures as Arguments and Return Values

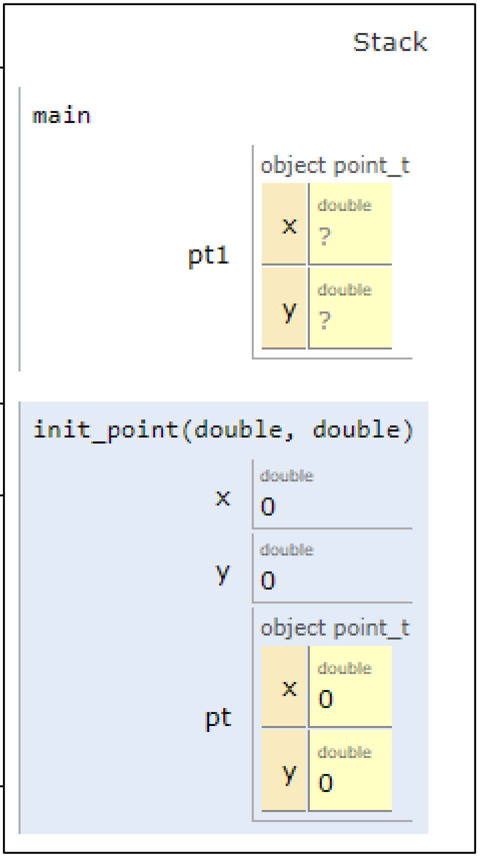
- Structures as Arguments
  - □ Pass a structure to a function require making a copy of all members in the structure
  - □ Example:

point.c

```
struct point_t init_point(double x, double y)
{
        struct point_t pt;
        pt.x = x;
        pt.y = y;
        return pt;
}
```

pts.c

```
int main(void)
{
        struct point_t pt1 = init_point(0.0, 0.0);

        return 0;
}
```



# Structures as Arguments and Return Values

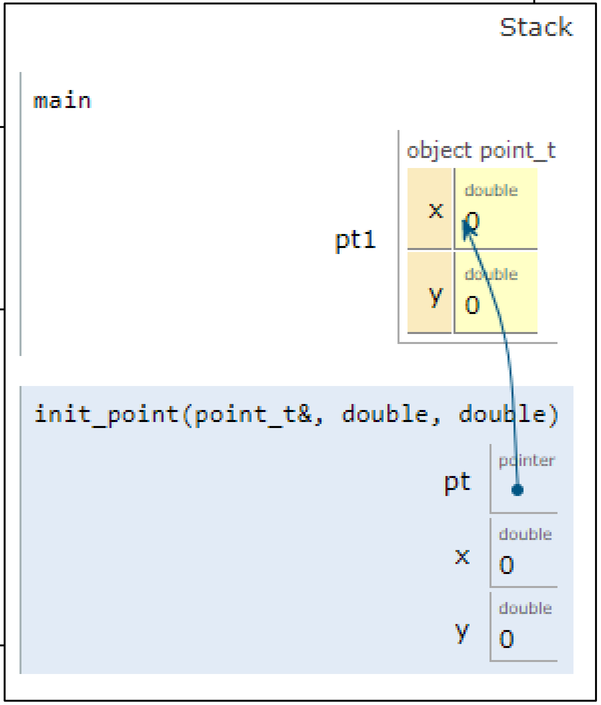
- Structures as Arguments
  - Another way to solve the problem
  - We typically pass `structs` to functions by reference.

point.c

```
void init_point(struct point_t &pt, double x, double y)
{
        pt.x = x;
        pt.y = y;
}
```

pts.c

```
int main()
{
  struct point_t pt1;
  init_point(pt1, 0.0, 0.0);

  return 0;
}
```



# Nested Structure

- Nesting one kind of structure inside another
- Example

```
struct person_name {
  string first;
  char middle_initial;
  string last;
};

struct student {
  struct person_name name;
  int id, age;
  char gender;
};

int main()
{
  struct student std1 = {{"John", 'A', "Doe"}, 32, 21};
  cout << "Student's first name is " << std1.name.first << endl;

  return 0;
}
```

# Arrays of Structures

# Review - Array

- Array declaration
  - Syntax: `type name[length]`
    - `type` is the data type of the array
    - `name` is the name you use to reference the array
    - `length` is the number of elements in the array, which is always one more than the last index of the array
  - Example, create an array to hold a 1000 integers

    ```
    int a[1000];
    ```
  - Another way to initialize array, which makes your code more maintainable, is to use a preprocessor constant

    ```
    const int SIZE = 1000;
    int a[SIZE];
    ```
  - Like other variables in C, declaring them does not initialize them

# struct Arrays

- Declaration
  - □ `struct` *structure_name variable_name*`[`*size*`];`
  - □ Example:

```
struct point_t pts[10];
```

- Visit elements in the struct array
  - □ As same as visiting elements in array with primary data types
  - □ Example:

```
struct point_t origin = {0.0, 0.0};
 for (i = 0; i < 10; i++) {
   cout << "distance to origin: "
        << distance(pts[i], origin) << endl;
 }
```

# struct Arrays

- Example
    - Find the distance of an array of points to the origin

pts.c

```
int main()
{
  const int NPOINTS = 5;
  int i;
  double dist[NPOINTS];
  struct point_t pts[NPOINTS] =
    {{3.5, 7.8}, {3.0, 4.0}, {2.0, 8.5},
     {6.3, 8.9}, {5.0, 5.0}};

  distance_to_origin(pts, NPOINTS, dist);

  for (i = 0; i < NPOINTS; i++)
    cout << dist[i] << "\t";
  cout << endl;

  return 0;
}
```

# struct Arrays

- Example
  - Find the distance of an array of points to the origin

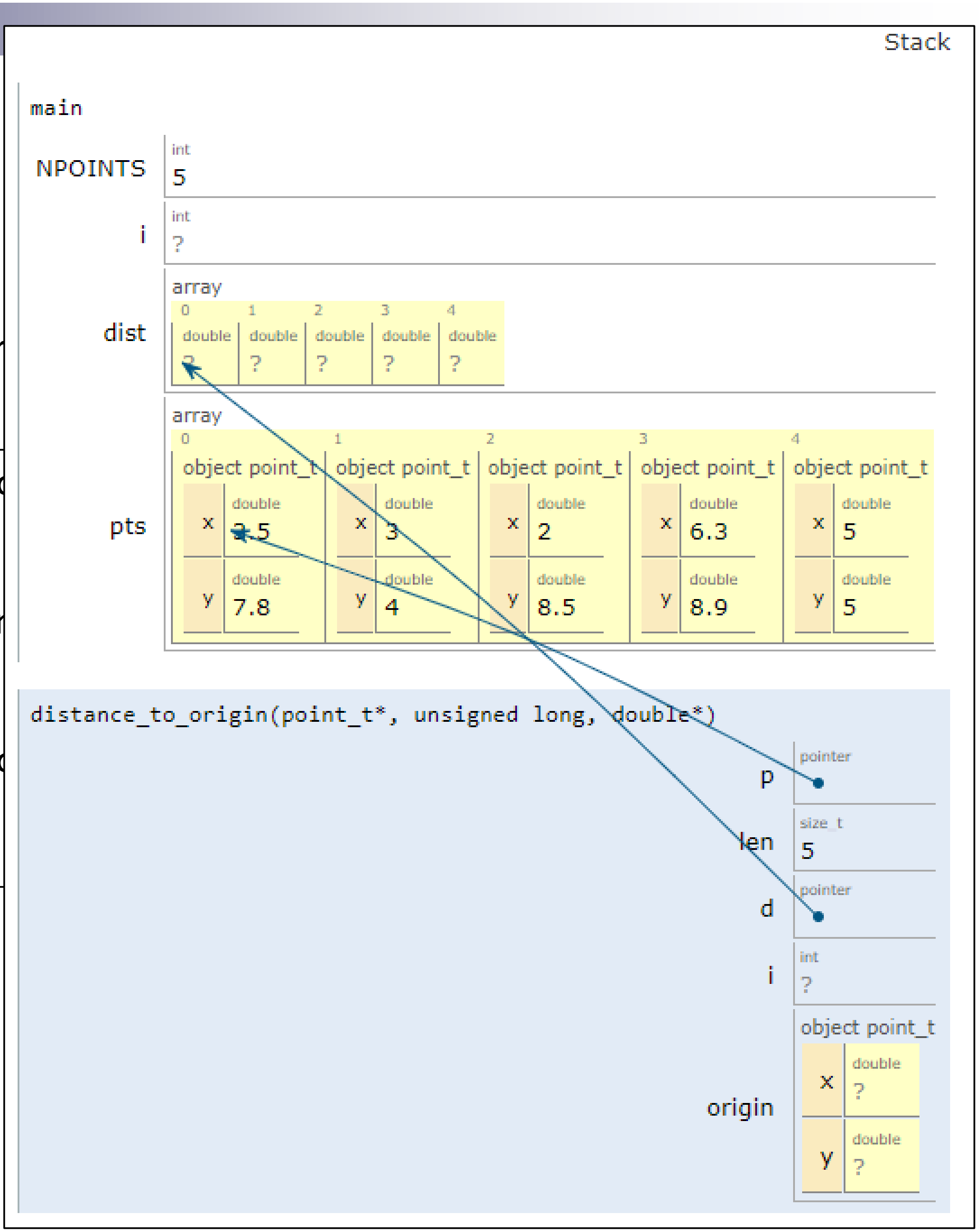
point.c

```
void distance_to_origin(struct point_t p[], size_t len, double d[])
{
        int i;
        struct point_t origin = {0.0, 0.0};

        for (i = 0; i < len; i++) {
               d[i] = distance(p[i], origin);
        }
}
```

# struct Arrays

- Example
  - Find the distance of an ar

```
void distance_to_origin(struc
{
        int i;
        struct point_t origi

        for (i = 0; i < len; 
                d[i] = distanc
        }
}
```



Stack

main

NPOINTS: int 5

i: int ?

dist: array

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| double<br>? | double<br>? | double<br>? | double<br>? | double<br>? |

pts: array

|  | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
|  | object point_t | object point_t | object point_t | object point_t | object point_t |
| x | double<br>3.5 | double<br>3 | double<br>2 | double<br>6.3 | double<br>5 |
| y | double<br>7.8 | double<br>4 | double<br>8.5 | double<br>8.9 | double<br>5 |

distance_to_origin(point_t*, unsigned long, double*)

p: pointer

len: size_t 5

d: pointer

i: int ?

origin: object point_t

x: double ?

y: double ?

# Structures and `vectors`

# struct vectors

- Example
  - Find the distance of an array of points to the origin

pts.c

```
int main()
{
  const int NPOINTS = 5;
  int i;
  double dist[NPOINTS];
  vector<struct point_t> pts =
    {{3.5, 7.8}, {3.0, 4.0}, {2.0, 8.5},
     {6.3, 8.9}, {5.0, 5.0}};

  distance_to_origin(pts, NPOINTS, dist);

  for (i = 0; i < NPOINTS; i++)
    cout << dist[i] << "\t";
  cout << endl;

  return 0;
}
```

# struct vectors

- Example
  - Find the distance of an array of points to the origin

point.c

```
void distance_to_origin(vector<struct point_t> &p, size_t len,
                                                          double d[])
{
  int i;
  struct point_t origin = {0.0, 0.0};

  for (i = 0; i < len; i++) {
    d[i] = distance(p.at(i), origin);
  }
}
```

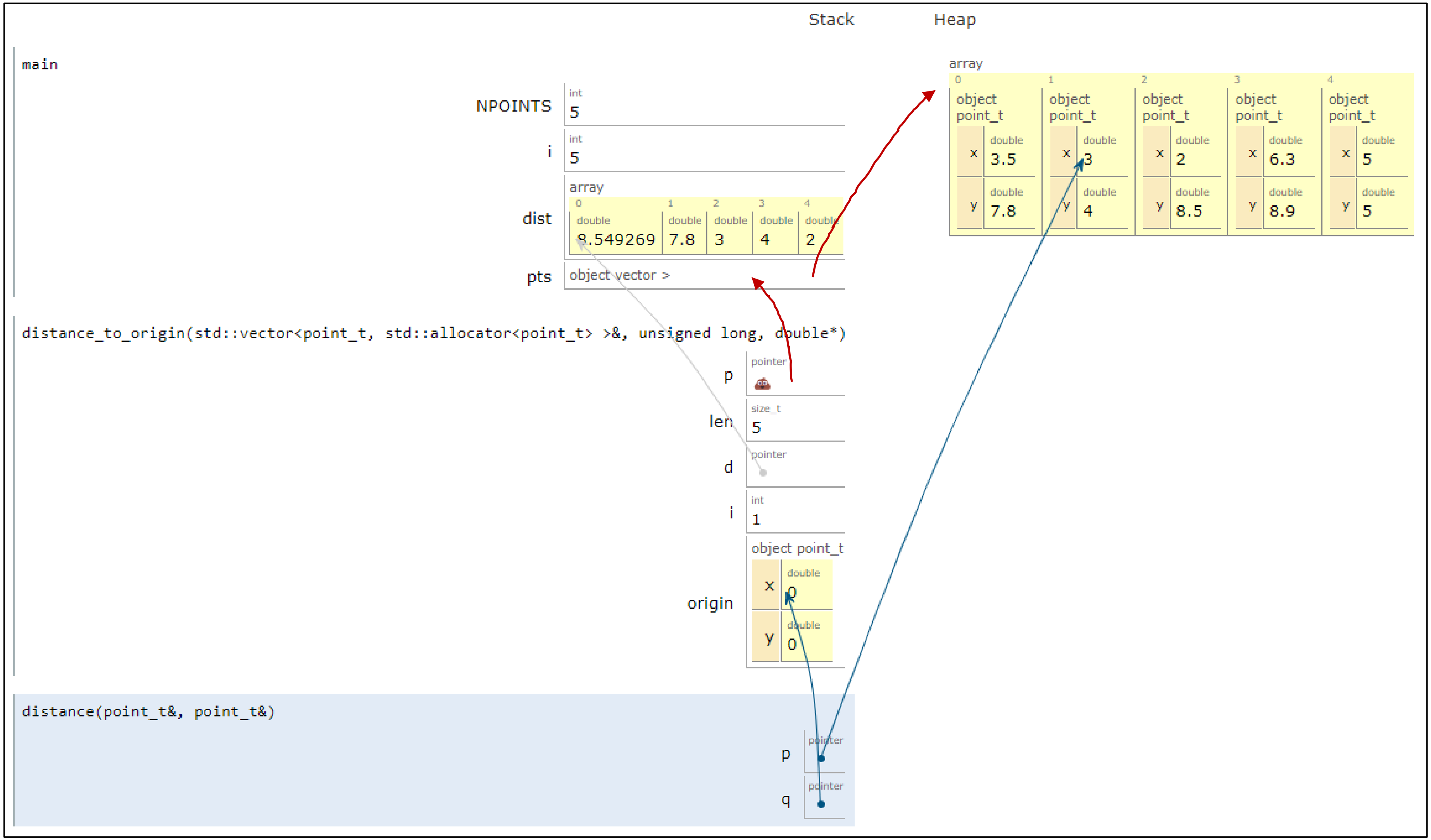
# struct vectors

- Example
  - Find the distance of an array of points to the origin

point.c

```
void distance_to_origin(vector<struct point_t> &p, size_t len,
                                                         double d[])
{
  int i;
  struct point_t origin = {0.0, 0.0};

  for (i = 0; i < len; i++) {
    d[i] = distance(p.at(i), origin);
  }
}
```

# struct vectors



Stack

Heap

main

NPOINTS int 5

i int 5

dist array

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| double | double | double | double | double |
| 8.549269 | 7.8 | 3 | 4 | 2 |

pts object vector >

distance_to_origin(std::vector<point_t, std::allocator<point_t> >&, unsigned long, double*)

p pointer

len size_t 5

d pointer

i int 1

origin object point_t

| | |
|---|---|
| x | double<br>0 |
| y | double<br>0 |

distance(point_t&, point_t&)

p pointer

q pointer

array

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| object point_t | object point_t | object point_t | object point_t | object point_t |
| x double 3.5 | x double 3 | x double 2 | x double 6.3 | x double 5 |
| y double 7.8 | y double 4 | y double 8.5 | y double 8.9 | y double 5 |

Figure 8.4.1

# struct vectors

- Example: Seat structure

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct Seat {
   string firstName;
   string lastName;
   int amountPaid;
};
```

Figure 8.4.1

# struct vectors

- Example:
  - □ Seat structure

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct Seat {
   string firstName;
   string lastName;
   int amountPaid;
};
```

Figure 8.4.1

# struct vectors

- Example:
  - Seat structure related functions - Seat structure only

```
void SeatMakeEmpty(Seat& seat) {
   seat.firstName   = "empty";
   seat.lastName    = "empty";
   seat.amountPaid  = 0;
}

bool SeatIsEmpty(Seat seat) {
   return(seat.firstName == "empty");
}

void SeatPrint(Seat seat) {
   cout << seat.firstName << " ";
   cout << seat.lastName << ", ";
   cout << "Paid: " << seat.amountPaid << endl;
}
```

Figure 8.4.1

# struct vectors

- Example:
  - □ Seat structure related functions - for Seat vector

```
void SeatsMakeEmpty(vector<Seat>& seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      SeatMakeEmpty(seats.at(i));
   }
}

void SeatsPrint(vector<Seat> seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      cout << i << ": ";
      SeatPrint(seats.at(i));
   }
}
```