# Structure

**Ahsan Ayub**

Ph.D. Student, Department of Computer Science

Graduate Research Assistant, CEROC

**CSC 1300: Introduction to Programming**

Tuesday, November 9, 2021

# Simple Program

Write a program to store three books' information (name, price, and number of pages).

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

int main()
{
    string names[3];
    double prices[3];
    unsigned int pages[3];

    for(int i = 0; i < 3; i++)
        cin >> names[i] >> prices[i] >> pages[i];

    for(int i = 0; i < 3; i++)
        cout << names[i] << "\t" << prices[i] << "\t" << pages[i] << endl;
}
```

# Introducing Structure

```
struct book
{
    string name;
    double price;
    int pages;
};
```

# Why use Structure?

- A collection of variables of different data types under a single name
- For example: We want to store some information about a person: name, date of birth, and salary.

```
struct person
{
    string name;
    string dob;
    double salary;
};
```

# Declaration of Structure

```
struct <structure_name>
{
    structure element1;
    structure element2;
    ...
    ...
};
```

# Declaration of Structure (back to our example)

```
struct Book
{
    string name;
    int pages;
    double price;
};
```

# Define a Structure Variable

```
<structure_name> <name_of_the_variable>
```

# Define a Structure Variable

```
Book b;
```

# Define a Structure Variable

```
Book b1, b2;
```

# Define a Structure Variable

```
Book b1, b2, b3;
```

# How to access members of a structure?

```
Book b1, b2, b3;

b1.name = “Intro to C++”;
b1.pages = 364;
b1.price = 157.64;

b2.name = “Intro to Calculus”;
b2.pages = 618;
b2.price = 113.99;

b3.name = “Intro to Physics”;
b3.pages = 571;
b3.price = 132.46;
```

# Simple Program w/ Structure

Write a program to store a book's information (name, price, and number of pages).

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

struct Book
{
    string name;
    double price;
    int pages;
};

int main()
{
    Book b;        // Declare the structure variable

    getline(cin, b.name);
    cin >> b.price;
    cin >> b.pages;

    cout << b.name << "\t" << b.price << "\t" << b.pages << endl;

    return 0;
}
```

# Simple Program w/ Structure (Debug Mode)

VARIABLES

Locals

b: {...}

name: "Intro to C++"

price: 89.459999999999993

pages: 172

books_static.cpp > main()

```
10
11  int main()
12  {
13      Book b;
14
15      b.name = "Intro to C++";
16      b.price = 89.46;
17      b.pages = 172;
18
19      cout << b.name << "\t" << b
```

# Let’s a define a structure to hold x, y points

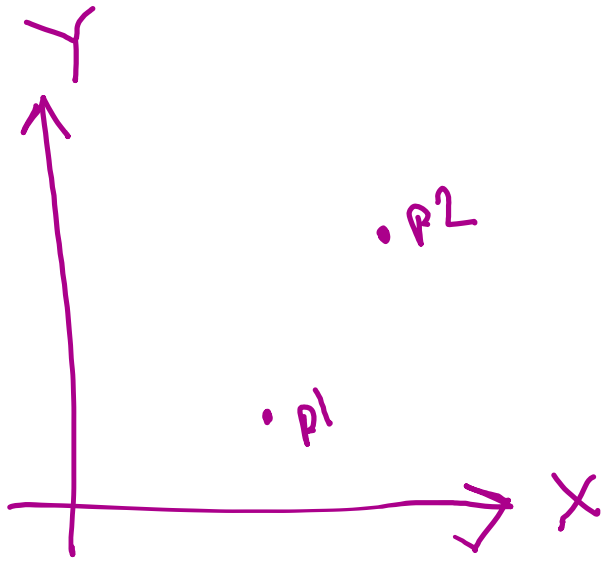
```
struct Points
{
   int x;
   int y;
};
```

# Let’s a define a structure to hold x, y points

```
Points p1, p2;

p1.x = 3;
p1.y = 2;

p2.x = 5;
p2.y = 6;
```



# Let’s a define a structure to hold x , y points

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
double distance;
distance = pow((p2.x - p1.x), 2);
distance += pow((p2.y - p1.y), 2);
distance = sqrt(distance);
```

# Let’s a define a structure to hold x, y points

```
#include <iostream>
#include <cmath>
using namespace std;

struct Points
{
    int x;
    int y;
};
```

```
int main()
{
    Points p1, p2;

    p1.x = 3;
    p1.y = 2;
    p2.x = 5;
    p2.y = 6;

    double distance;
    distance = pow((p2.x - p1.x), 2);
    distance += pow((p2.y - p1.y), 2);
    distance = sqrt(distance);

    cout << distance << endl;
    return 0;
}
```

# Let’s dig deep with addresses and memory allocation of that program!

```
Points p1, p2;

p1.x = 3;
p1.y = 2;

p2.x = 5;
p2.y = 6;

printf("%p\n", &p1.x);
printf("%p\n", &p1.y);
printf("%p\n", &p2.x);
printf("%p\n", &p2.y);
```

C language literals to display addresses

```
Points p1, p2;

p1.x = 3;
p1.y = 2;

p2.x = 5;
p2.y = 6;

cout << &p1.x << endl;
cout << &p1.y << endl;
cout << &p2.x << endl;
cout << &p2.y << endl;
```

C++ language literals to display addresses

# Let’s dig deep with addresses and memory allocation of that program!

```
Points p1, p2;

p1.x = 3;
p1.y = 2;

p2.x = 5;
p2.y = 6;

printf("%p\n", &p1.x);
printf("%p\n", &p1.y);
printf("%p\n", &p2.x);
printf("%p\n", &p2.y);
```

C language literals to display addresses

```
Points p1, p2;

p1.x = 3;
p1.y = 2;

p2.x = 5;
p2.y = 6;

cout << &p1.x << endl;
cout << &p1.y << endl;
cout << &p2.x << endl;
cout << &p2.y << endl;
```

C++ language literals to display addresses

```
0x7ff7bf7fe3e0
0x7ff7bf7fe3e4
0x7ff7bf7fe3d8
0x7ff7bf7fe3dc
```

# Structures and Functions

Structure variables can be passed to a function and returned in a similar way as normal arguments.

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

struct Points
{
    int x;
    int y;
};

void DisplayPoints(Points p)
{
    cout << p.x << “\t” << p.y << endl;
}

int main()
{
    Points point;
    point.x = 10;
    point.y = 20;
    DisplayPoints(point);
    return 0;
}
```

```
??
```

# Structures and Functions

Structure variables can be passed to a function and returned in a similar way as normal arguments.

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

struct Points
{
    int x;
    int y;
};

void DisplayPoints(Points p)
{
    cout << p.x << “\t” << p.y << endl;
}

int main()
{
    Points point;
    point.x = 10;
    point.y = 20;
    DisplayPoints(point);
    return 0;
}
```

```
10       20
```

# Structures and Functions

Structure variables can be passed to a function and returned in a similar way as normal arguments.

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

struct Points
{
    int x;
    int y;
};

Points BuildPoint(int a, int b)
{
    Points temp;
    temp.x = a;
    temp.y = b;
    return temp;
}

void DisplayPoints(Points p)
{
    cout << p.x << "\t" << p.y << endl;
}

int main()
{
    Points point;
    point = BuildPoint(5, 7);
    DisplayPoints(point);
    return 0;
}
```

??

# Structures and Functions

Structure variables can be passed to a function and returned in a similar way as normal arguments.

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

struct Points
{
    int x;
    int y;
};

Points BuildPoint(int a, int b)
{
    Points temp;
    temp.x = a;
    temp.y = b;
    return temp;
}

void DisplayPoints(Points p)
{
    cout << p.x << "\t" << p.y << endl;
}

int main()
{
    Points point;
    point = BuildPoint(5, 7);
    DisplayPoints(point);
    return 0;
}
```

```
5	7
```

# Output Tracing

Structure variables can be passed to a function and returned in a similar way as normal arguments.

```
1  #include <iostream>
2  using namespace std;
3
4  struct Points
5  {
6      int x;
7      int y;
8  };
9
10 Points BuildPoint(int a, int b)
11 {
12     Points temp;
13     temp.x = a;
14     temp.y = b;
15     return temp;
16 }
17
18 void DisplayPoints(Points p)
19 {
20     cout << p.x << "\t" << p.y << endl;
21 }
22
23 int main()
24 {
25     DisplayPoints(BuildPoint(5, 7));
26     return 0;
27 }
```

```
??
```

# Output Tracing

Structure variables can be passed to a function and returned in a similar way as normal arguments.

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

struct Points
{
    int x;
    int y;
};

Points BuildPoint(int a, int b)
{
    Points temp;
    temp.x = a;
    temp.y = b;
    return temp;
}

void DisplayPoints(Points p)
{
    cout << p.x << "\t" << p.y << endl;
}

int main()
{
    DisplayPoints(BuildPoint(5, 7));
    return 0;
}
```

```
5	7
```

# Pointers to Structure

A pointer variable can be created not only for native types like (int, double etc.) but also they can also be created for user defined types like structure.

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

struct Points
{
    int x;
    int y;
};

int main()
{
    // p is the pointer variable of structure
    Points *p, point;
    // p is pointing to a regular structure variable
    p = &point;
    // Access the structure data with pointer and
    // -> notation. (*p).x and (*p).y would work too.
    p->x = 10;
    p->y = 20;
    // Displaying the values of the elements
    cout << p->x << "\t" << p->y << endl;
    return 0;
}
```

```
10        20
```

# Pointers to Structure (Debug Mode)

VARIABLES
Locals
p: 0x00007ff7bfeffc18
x: 10
y: 20
point: {...}
x: 10
y: 20

struct_pointers.cpp > main()

```
3
4   struct Points
5   {
6       int x;
7       int y;
8   };
9
10  int main()
11  {
12      // p is the pointer variable of structure
13      Points *p, point;
14      // p is pointing to a regular structure varia
15      p = &point;
16      // Access the structure data with pointer and
17      // -> notation. (*p).x and (*p).y would work
18      p->x = 10;
19      p->y = 20;
20      // Displaying the values of the elements
21      cout << p->x << "\t" << p->y << endl;
22      return 0;
```

# Simple Program w/ Structure: Array to Structures

Write a program to store 3 books' information (name, price, and number of pages).

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

struct Book
{
    string name;
    double price;
    int pages;
};

int main()
{
    Book b[3];     // Declare an array to structure

    for(int i = 0; i < 3; i++) {
         getline(cin, b[i].name);
         cin >> b[i].price;
         cin >> b[i].pages;
         cin.ignore();
     }
     for(int i = 0; i < 3; i++)
        cout << b[i].name << "\t" << b[i].price << "\t" << b[i].pages << endl;
}
```

# Array to Structure (Debug Mode)

```
4   struct Book
5   {
6       string name;
7       double price;
8       int pages;
9   };
10
11  int main()
12  {
13      Book b[3];
14
15      b[0].name = "Intro to C++";
16      b[0].price = 89.64;
17      b[0].pages = 114;
18
19      b[1].name = "Intro to Python";
20      b[1].price = 101.47;
21      b[1].pages = 156;
22
23      b[2].name = "Intro to C#";
24      b[2].price = 95.12;
25      b[2].pages = 247;
26
27      for(int i = 0; i < 3; i++)
28          cout << b[i].name << "\t" << b[i].price << "\t" << b[i].pages << endl;
```

# Array to Structure (Debug Mode)

VARIABLES

Locals

```
b: [3]
  [0]: {...}
    name: "Intro to C++"
    price: 89.64
    pages: 114
  [1]: {...}
    name: "Intro to Python"
    price: 101.47
    pages: 156
  [2]: {...}
    name: "Intro to C#"
    price: 95.120000000000005
    pages: 247
i: 0
```

WATCH

CALL STACK PAUSED ON STEP

books_array!main books_arra...

dyld!start Unknown Source

books_array.cpp > main()

```cpp
int main()
{
    Book b[3];

    b[0].name = "Intro to C++";
    b[0].price = 89.64;
    b[0].pages = 114;

    b[1].name = "Intro to Python";
    b[1].price = 101.47;
    b[1].pages = 156;

    b[2].name = "Intro to C#";
    b[2].price = 95.12;
    b[2].pages = 247;

    for(int i = 0; i < 3; i++)
        cout << b[i].name << "\t" << b[i].price << "\t" << b[i].pages << endl;

    return 0;
}
```

# Array to Structure (Another way to initialize)

```
struct Book
{
   string name;
   double price;
   int pages;
};

int main()
{
    Book b[3] = {
                    {“Intro to C++”, 89.64, 114},
                    {“Intro to Python”, 101.47, 156},
                    {“Intro to C#”, 95.12, 247}
                 };
}
```

# Allocate Struct Memory w/ `malloc`

```
struct Points
{
    int x;
    int y;
};

int main()
{
    Points point;
}
```

```
struct Points
{
    int x;
    int y;
} point;
```

# Allocate Struct Memory w/ `malloc`

```
struct Points
{
    int x;
    int y;
};

int main()
{
    Points point;
    Points *p = (points *) malloc(sizeof(point));
    p->x = 10;
    p->y = 20;
    cout << p->x << endl;
    cout << p->y << endl;
}
```

```
struct Points
{
    int x;
    int y;
} point;

int main()
{
    Points *p = (points *) malloc(sizeof(point));
    p->x = 10;
    p->y = 20;
    cout << p->x << endl;
    cout << p->y << endl;
}
```

```
10
20
```

# Allocate Struct Memory w/ `malloc` (Debug)

VARIABLES

Locals

p: 0x000060000000c000

x: 0

y: 0

dynamic_struct.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  struct Points
5  {
6      int x;
7      int y;
8  } point;
9
10 int main()
11 {
12     Points *p = (Points *) malloc(sizeof(point));
13
14     p->x = 10;
15     p->y = 20;
16
17     cout << p->x << "\t" << p->y << endl;
18     return 0;
19 }
```

# Allocate Struct Memory w/ `malloc` (Debug)

VARIABLES
Locals
p: 0x000060000000c000
x: 10
y: 0

dynamic_struct.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  struct Points
5  {
6      int x;
7      int y;
8  } point;
9
10 int main()
11 {
12     Points *p = (Points *) malloc(sizeof(point));
13
14     p->x = 10;
15     p->y = 20;
16
17     cout << p->x << "\t" << p->y << endl;
18     return 0;
19 }
```

# Allocate Struct Memory w/ `malloc` (Debug)

VARIABLES
Locals
p: 0x000060000000c000
x: 10
y: 20

dynamic_struct.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  struct Points
5  {
6      int x;
7      int y;
8  } point;
9
10 int main()
11 {
12     Points *p = (Points *) malloc(sizeof(point));
13
14     p->x = 10;
15     p->y = 20;
16
17     cout << p->x << "\t" << p->y << endl;
18     return 0;
19 }
```

# Allocate an Array of Struct Memory w/ `malloc`

```
1  #include <iostream>
2  using namespace std;
3
4  struct Points
5  {
6      int x;
7      int y;
8  } point;
9
10 int main()
11 {
12     Points *p = (Points *) malloc(10 * sizeof(point));
13
14     for(int i = 0; i < 10; i++)
15     {
16         p[i].x = i;
17         p[i].y = i * 10;
18     }
19
20     for(int i = 0; i < 10; i++)
21         cout << p[i].x << "\t" << p[i].y << endl;
22
23     free(p); // Free the allocated memory
24     return 0;
25 }
```

# Allocate an Array of Struct Memory w/ `malloc`

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

struct Points
{
    int x;
    int y;
} point;

int main()
{
    Points *p = (Points *) malloc(10 * sizeof(point));

    for(int i = 0; i < 10; i++)
    {
        p[i].x = i;
        p[i].y = i * 10;
    }

    for(int i = 0; i < 10; i++)
        cout << p[i].x << "\t" << p[i].y << endl;

    free(p); // Free the allocated memory
    return 0;
}
```

o/p →

```
0       0
1       10
2       20
3       30
4       40
5       50
6       60
7       70
8       80
9       90
```

# Allocate an Array of Struct Memory w/ `malloc`

```cpp
#include <iostream>
using namespace std;

struct Points
{
    int x;
    int y;
} point;

int main()
{
    Points *p = (Points *) malloc(10 * sizeof(point));

    for(int i = 0; i < 10; i++)
    {
        (*(p+i)).x = i;          // same as p[i].x = i;
        (*(p+i)).y = i * 10;     // same as p[i].y = i * 10;
    }

    for(int i = 0; i < 10; i++)
        cout << p[i].x << "\t" << p[i].y << endl;

    free(p); // Free the allocated memory
    return 0;
}
```

o/p →

```
0       0
1       10
2       20
3       30
4       40
5       50
6       60
7       70
8       80
9       90
```

# Remarks

- Reference
  - Structures in C++. Programiz. https://www.programiz.com/cpp-programming/structure
  - Let us C by Yashavant P. Kanetkar